The Value of Consensus in Online Stochastic Scheduling

Russell Bent and Pascal Van Hentenryck

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-03-22
October 2003

The Value of Consensus in Online Stochastic Scheduling

Russell Bent and **Pascal Van Hentenryck**
Brown University
Providence, RI 02912
{rbent,pvh}@cs.brown.edu

Abstract

This paper reconsiders online packet scheduling in computer networks, where the goal is to minimize weighted packet loss and where the arrival distributions of packets, or approximations thereof, are available for sampling. Earlier work proposed an expectation approach, which chooses the next packet to schedule by approximating the expected loss of each decision over a set of scenarios. The expectation approach was shown to significantly outperform traditional approaches ignoring stochastic information.

This paper proposes a novel stochastic approach for online packet scheduling, whose key idea is to select the next packet as the one which is scheduled first most often in the optimal solutions of the scenarios. This consensus approach is shown to outperform the expectation approach significantly whenever time constraints and the problem features limit the number of scenarios that can be solved before making a decision. More importantly perhaps, the paper shows that the consensus and expectation approaches can be integrated to combine the benefits of both approaches. These novel online stochastic algorithms are problem-independent, they should apply to other online applications as well, and they shed new light on why existing online stochastic algorithms behave well.

Introduction

Online packet scheduling is a class of important problems arising in communication networks. These problems may significantly vary in scope and difficulty but they are often challenging since they may require online solutions to complex combinatorial optimization problems. This paper reconsiders the online packet scheduling model from (Chang, Givan, & Chong 2000) where jobs from different classes arrive stochastically and must be scheduled within a fixed amount of time or dropped. Each job has a weight and the goal is to minimize the weighted packet loss or, alternatively, to maximize the sum of the weights of the scheduled packets. The arrival distributions of the packets, which are quite complex in our experiments, are available for sampling.

The offline version of this packet scheduling problem is relatively simple and can be solved in polynomial time. However, it is sufficiently rich to demonstrate and study the practical issues arising in online optimization. Indeed, the main purpose of (Chang, Givan, & Chong 2000) in studying this problem was to demonstrate that exploiting stochastic information may provide significant benefits in online packet scheduling. They proposed an *expectation* approach which chooses the next packet to schedule by approximating the expected loss of each packet class over the distribution. The approximation samples the distributions to build scenarios from available and future packets. The expectation algorithm was shown to significantly outperform traditional approaches which ignore stochastic information such as greedy heuristics and algorithms optimizing the decisions based on known packets. However, the expectation algorithm may be quite demanding computationally, since it solves each scenario for each class of packets. As a consequence, it may not scale well when the number of classes increases and it may not be applicable when decisions must be taken within strong time constraints.

This paper reconsiders online packet scheduling and proposes a *consensus* approach which was inspired by our earlier work on online vehicle routing where decisions must be taken quickly and the optimization algorithms are time-consuming (Bent & Van Hentenryck 2001; 2003). At each time t, the key idea underlying the consensus approach is to solve each scenario once and to select the packet which is scheduled first the most often in the optimal solutions of the scenarios. This consensus algorithm outperforms the expectation approach significantly when time limits restrict the number of scenarios which can be solved before making the decision. Indeed, in the consensus approach, solving a scenario produces global information on all possible actions, not on a single action as in the expectation approach. Moreover, the paper demonstrates that the *consensus* and *expectation* approaches synergize and can be hybridized to preserve most of their respective advantages. In particular, the hybridization provides the quality of the expectation algorithm when the number of available samples is large and the quality of the consensus approach what there is only time to solve a few scenarios. The key contributions of this paper can thus be summarized as follows:

1. it proposes a consensus approach to online packet scheduling which (1) reduces the computational burden of the expectation method and (2) significantly outperforms it when time constraints limit the number of scenarios that can be solved in between two events;

2. it proposes an hybrid algorithm which almost always outperforms the expectation and consensus approaches and combines their benefits;
3. it provides experimental justification for the consensus approach in online vehicle routing since these applications can only solve 2-10 scenarios in between events.
4. it provides experimental evidence that consensus approaches quickly converge to their optimal solution values in terms of the number of scenarios.
5. it proposes two stochastic approaches which are essentially generic and may apply to many other online optimization problems.

The rest of the paper is organized as follows. It first formulates online packet scheduling precisely as a scheduling problem and describes the online stochastic optimization framework. The next section discusses the relationships to POMDPs, since the packet scheduling problem was initially modelled as a POMDP. The paper then describes the polynomial time offline algorithm, the oblivious online algorithms, and the online stochastic approaches. It reports the experimental results, describes related work, and give the conclusion and future directions.

Problem Description

The online packet scheduling problem was initially proposed by (Chang, Givan, & Chong 2000) to study the benefits of stochastic information on the quality of the schedules. Its offline version can be specified as follows. We are given a set *Jobs* of jobs partitioned into a set of classes C. Each job j is chararacterized 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 $H = [\underline{H}, \overline{H}]$ 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

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

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 $w(\sigma)$.

In the online version, jobs are not known a priori but new jobs arrive at each time step, at most one per class. As in (Chang, Givan, & Chong 2000), we assume that the arrival distribution of the jobs, or an approximation thereof, can be sampled. In other words, we assume the existence of a black-box which can be queried to return samples of future job requests. Once again, the goal is to find a schedule of maximal weight. Observe that, although classes are not significant in the offline version of the problem, they are used to reduce the complexity of the expectation algorithm. Note also that, in (Chang, Givan, & Chong 2000), the arrival distribution of jobs of a given class is governed by an independent Hidden Markov Model (HMM).

```
onlineOptimization(H)
J ← ∅;
w ← 0;
for t ∈ H
do J ← availableRequests(J, t) ∪ newRequests(t);
   j ← chooseRequest(J, t);
   serveRequest(j, t);
   w ← w + w(j);
   J ← J \ {j};
```

Figure 1: The Generic Online Algorithm

Online Stochastic Optimization

As mentioned earlier, our goal is to design and evaluate various approaches to online stochastic optimization. In particular, the main focus of this paper is on determining how to use sampling in order to improve the quality of the online solutions under various time constraints. Hence the techniques considered in this paper apply to a variety of online stochastic optimization problems and are not tailored to the packet scheduling problem. They only rely of the existence of an algorithm to solve (or approximate) the offline problem and on the ability to sample the distribution. More precisely, the approaches presented herein are expressed in terms of the two black-boxes:

1. a function OPTIMALSCHEDULE(J, H) which, given a set J of jobs and a schedule horizon H, returns an optimal schedule of J over H;
2. a function GETSAMPLE(H) which, given a schedule horizon H, returns a set of jobs to be scheduled over H by sampling the arrival distribution.

When facing another online stochastic optimization problems, it suffices to replace, or implement, these two functions for the application at hand.

Moreover, all the approaches described in this paper shares the same overall structure which is depicted in Figure 1 and expressed in general terms to emphasize its independence from the packet scheduling problem. The approaches only differ 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 j which is then served and removed from the set of available requests. In the case of packet scheduling, J denotes the set of jobs to schedule, function AVAILABLEREQUEST(J, t) returns the set of jobs that are available at time t, i.e.,

$$\text{AVAILABLEREQUEST}(J, t) = \{j \in J \mid t \le a(j) + d\}$$

and function SERVEREQUEST(j, t) simply schedules j at time t ($\sigma(t) \leftarrow j$). Subsequent sections specify the offline algorithms, i.e., their implementations of function CHOOSEREQUEST. However, we first relate this framework to POMDPs.

Relationships to POMDPs

POMDPs have been used to represent and solve a broad class of problems with inherent uncertainty and the packet scheduling problem was initially modeled as a POMDP by (Chang, Givan, & Chong 2000). It is thus appropriate to relate this work to this line of research. There is a wide variety of approaches to solve or approximate general POMDPs in the expected sense. Of particular relevance to this work is the sampling method of (McAllester & Singh 1999) (see also (Kearns, Mansour, & Ng 1999a; 1999b)). This approach computes an expectimax function on the POMDP. In other words, it determines the action/policy that achieves the best reward in the expected sense. Informally speaking, computing this function involves a randomized tree exploration: the method tries an action, samples the distribution to find a next state, tries another action, and repeats. Of course, to evaluate the best possible expected benefit of an action a at a node, all possible subsequent actions must be tried. The best of these subsequent actions determines the best possible benefit of a.

An important theoretical result of expectimax sampling is that the actual expected value of an action can be formally approximated given enough samples and a sufficient depth. However, the resulting algorithm is exponential wrt the number of actions and the depth. Indeed, as pointed out by (Chang, Givan, & Chong 2000), when a guaranteed approximation is desired, the method is intractable for many problems, including the packet scheduling problem, even when dynamic programming and other techniques are used to reduce computation. When the parameters are chosen to ensure tractability, the algorithm is no longer competitive. However, (Chang, Givan, & Chong 2000) observed that the randomness in the POMDP is independent of the actions taken. The expectimax tree can then be collapsed to a single level, where each action is evaluated against a number of scenarios obtained through sampling. Algorithm E, which is presented later in the paper, was proposed in (Chang, Givan, & Chong 2000) as a general algorithm for POMDPs with this property and applied to the packet scheduling problem to show the value of stochastic information. The new algorithms presented herein would apply to the same class of POMDPs and may be of interest to other classes of POMDPs as well.

Offline Scheduling Algorithm

An interesting aspect of the offline scheduling problem is that it is solvable in polynomial time (Chang, Givan, & Chong 2000). The quadratic algorithm is loosely based upon earlier work in (Villareal & Bulfin 1983). Although a detailed understanding of the algorithm is not necessary for the rest of the paper, Figure 2 gives the pseudo-code for the optimal algorithm for completeness. The algorithm builds the optimal schedule σ from right to left by considering each job successively, starting with the job of highest weight and, in case of ties, of latest arrival date (line 2). For each job j, the algorithm finds the right-most time t before the deadline $a(j) + d$ where j can be scheduled (line 3). If such time t exists, there are two possibilities. If $t \ge a(j)$, then the algorithm simply schedules j at time t (line 6). Otherwise, the algorithms tries to shuffle the schedule to accommodate j after its arrival date by shifting existing jobs left (line 7). The shuffling successively swaps job j (initially scheduled at time t) with jobs on its right until j is scheduled after $a(j)$ or no such swap is feasible. Once a schedule is built, the algorithm applies a postprocessing step to make sure that jobs with highest weights and, in case of ties, earliest arrival dates, are scheduled as early as possible (line 8). This postprocessing step is slightly more specific than in (Chang, Givan, & Chong 2000) to ensure that the offline schedule is most appropriate for an online use. Note that, in Figure 2, we use $\sigma(t) = \perp$ to denote that no job is scheduled at time t at this stage. Moreover, FEASIBLESWAP(σ, t_1, t_2) holds if

$$a(j_1) \le t_2 \le a(j_1) + d \ \wedge\ a(j_2) \le t_1 \le a(j_2) + d.$$

where $j_1 = \sigma(t_1)$ and $j_2 = \sigma(t_2)$. We also use $min\ S$ (resp. $max\ S$) to denote the minimum (resp. maximum) of a set S. By convention, if S is empty, both expressions return $\perp$.

OPTIMALSCHEDULE(*Jobs*, H)
1 $\sigma(t) \leftarrow \perp$ $(t \in H)$;
2 **for** $j \in Jobs$ ordered by decreasing $\langle w(j), a(j)\rangle$
3 **do** $p \leftarrow max\{t \in H \mid t \le a(j) + d \;\&\; \sigma(t) = \perp\}$;
4 **if** $p \ne \perp$
5 **then if** $p \ge a(j)$
6 **then** $\sigma(p) \leftarrow j$;
7 **else** $\sigma \leftarrow$ SHUFFLE(σ, j, p);
8 POSTPROCESS$(\sigma, Jobs, H)$;
9 **return** σ;

SHUFFLE(σ, j, p)
1 $\sigma' \leftarrow \sigma$;
2 $\sigma(p) \leftarrow j$;
3 **while** $p < a(j)$
4 **do** $q \leftarrow min\{t \mid p + 1 \le t \le p + d \;\&\; a(\sigma(t)) \le p\}$;
5 **if** $q \ne \perp$
6 **then** SWAP(σ, p, q);
7 $p \leftarrow q$;
8 **else return** σ';
9 **return** σ;

POSTPROCESS(σ, H)
1 **for** $p \in H, q \in H : p < q$
2 **do if** FEASIBLESWAP(σ, p, q) &
3 $\langle w(\sigma(p)), a(\sigma(p))\rangle < \langle w(\sigma(q)), a(\sigma(q))\rangle$
4 **then** SWAP(σ, p, q);

Figure 2: The Optimal Packet Scheduling Algorithm

Online Oblivious Algorithms

This section describes two existing online algorithms which are not using stochastic information.

Greedy (G): This heuristic schedules the job with highest weight and, in case of ties, the job with the earliest deadline. (Chang, Givan, & Chong 2000) refers to this heuristic as SP. It can be specified formally as

CHOOSEREQUEST-G(J, t)
1 $\sigma \leftarrow$ OPTIMALSCHEDULE$(J, [t, t])$;
2 **return** $\sigma(t)$;

Local Optimal (LO): This algorithm chooses the next job to schedule at time t by finding the optimal solution for the available jobs at t. The first job in the optimal solution is selected. (Chang, Givan, & Chong 2000) refers to this algorithm as CM. It can be specified formally as

CHOOSEREQUEST-LO(J, t)
1 $\sigma \leftarrow$ OPTIMALSCHEDULE$(J, [t, t + d])$;
2 **return** $\sigma(t)$;

Note that the optimal schedule cannot expand beyond time $t + d$ by definition of the time windows.

Online Stochastic Algorithms

We now consider a variety of online approaches using stochastic information to make more informed decisions. We assume that each approach has the time to execute the offline algorithm *nbSamples* times for samples which extend Δ time steps in the future. Both *nbSamples* and Δ are parameters of the online stochastic algorithms. Note that function OPTIMALSCHEDULE is quadratic in the schedule horizon in the worst case.

Expectation (E): This is the primary method proposed by (Chang, Givan, & Chong 2000). It is referred to as SPM in that paper. Informally speaking, the method generates future jobs by sampling and evaluates each possible seleec-tion against that sample. A naive implementation can be specified as follows:

CHOOSEREQUEST-NE(J, t)
1 **for** $j \in J$
2 **do** $f(j) \leftarrow 0$;
3 **for** $i \leftarrow 1 \ldots nbSamples/|J|$
4 **do** $R \leftarrow J \ \cup$ GETSAMPLE$([t + 1, t + \Delta])$;
5 $T \leftarrow [t + 1, t + 1 + \Delta + d]$;
6 **for** $j \in J$
7 **do** $\sigma \leftarrow$ OPTIMALSCHEDULE$(R \setminus \{j\}, T)$;
8 $f(j) \leftarrow f(j) + w(j) + w(\sigma)$;
9 **return** $argmax(j \in J) \ f(j)$;

Lines 1-2 initialize the evaluation function $f(j)$ for each job j. The algorithm then generates a number of samples for future requests (lines 3-4). For each such sample, it computes the set R of all available and sampled jobs at time t (line 4) and an appropriate time horizon for the scenarios (line 5) The algorithm then considers each available job j successively, it implicitly schedules j at time t, and applies the optimal offline algorithm (line 7) using $R \setminus \{j\}$ and the time horizon. The evaluation of job j is updated in line 8 by incrementing it with its weight and the score of the corresponding optimal offline schedule σ. All scenarios are evaluated for all jobs and the algorithm then returns the job $j \in J$ with the highest evaluation.

The above implementation is very ineffective when applied to the packet scheduling problem. Indeed, the algorithm can be considerably improved by considering only one job per class (i.e., the one with the earliest arrival date), since other choices are guaranteed not to produce better schedules. The jobs are called the *representative* jobs in the following. The representative job with the highest evaluation is then chosen. This reduces the number of iterations of the inner loop from $|J|$ to C, which is typically much smaller. The implementation of E can be specified as follows:

CHOOSEREQUEST-E(J, t)
1 **for** $c \in C$
2 **do** $j(c) \leftarrow argmin(j \in J : c(j) = c) \ a(j)$;
3 $f(j(c)) \leftarrow 0$;
4 **for** $i \leftarrow 1 \ldots nbSamples/|C|$
5 **do** $R \leftarrow J \ \cup$ GETSAMPLE$([t + 1, t + \Delta])$;
6 $T \leftarrow [t + 1, t + 1 + \Delta + d]$;
7 **for** $c \in C$
8 **do** $\sigma \leftarrow$ OPTIMALSCHEDULE$(R \setminus \{j(c)\}, T)$;
9 $f(j(c)) \leftarrow f(j(c)) + w(j(c)) + w(\sigma)$;
10 $c^* = argmax(c \in C) \ f(j(c))$;
11 **return** $j(c^*)$;

Lines 1-3 initializes the representative jobs $j(c)$ for each class c and their evaluations $f(j(c))$. The algorithm then proceeds as before but only considers representative jobs.[1]

To select the job to schedule at time t, algorithm E calls OPTIMALSCHEDULE $nbSamples * |C|$ times for the packet scheduling problem. More generally, algorithm E applies OPTIMALSCHEDULE $nbSamples * |R|$ times for a problem where R is the number of requests to consider. This is a significant drawback when time is limited and/or when the number of requests is large. This is precisely why online vehicle routing algorithms (Bent & Van Hentenryck 2001; 2003) do not use this method, since the number of requests is very large (about 50 to 100), time between decisions is limited, and the optimization algorithm is demanding computationally.

Consensus (C): We now turn to a novel online algorithm. Algorithm C uses stochastic information in a fundamental different way and can be viewed as an abstraction of

[1] It is possible to design an alternative sampling method that utilizes independent samples for each request. However empirical studies in (Chang, Givan, & Chong 2000) and by these authors on this problem show this approach is less successful in general.

the sampling method used in online vehicle routing with stochastic customers (Bent & Van Hentenryck 2001; 2003). Instead of evaluating each possible request at time t with respect to each sample, its key idea underlying algorithm C is to execute the optimization algorithm on the available and sampled jobs and to count the number of times a job is scheduled at time t in each resulting solution. The the job with the highest count is selected. The motivation behind algorithm C is the least-commitment heuristic (Stefik 1981) often used in planning and scheduling. By choosing a job which is selected first the most often, algorithm C takes a decision which is consistent with the optimal solutions of many samples. Algorithm C can be specifies as follows:

CHOOSEREQUEST-C(J, t)
1 **for** $j \in J$
2   **do** $c(j) \leftarrow 0$;
3 **for** $i \leftarrow 1 \ldots nbSamples$
4   **do** $R \leftarrow J \cup$ GETSAMPLE$([t+1, t+\Delta])$;
5       $\sigma \leftarrow$ OPTIMALSCHEDULE$(R, [t, t+\Delta+d])$;
6       $c(\sigma(t)) \leftarrow c(\sigma(t)) + 1$;
7 **return** $argmax(j \in J)\ c(j)$;

Observe line 5 which calls the offline algorithm with all available and sampled jobs and a schedule horizon starting at t and line 6 which increments the number of times job $\sigma(t)$ is scheduled first. Line 7 simply returns the job with the largest count. Algorithm C has several appealling features. First, it does not partition the available samples between the job classes, which is a significant advantage when the number of samples is small and/or when the number of classes is large. Second, by virtue of the least-commitment approach, it avoids the conceptual complexity of dealing with classes of jobs. The main strength of algorithm C is to identify (quickly and implicitly) the most promising requests. Its limitation is that it ignores the score of the schedules and may fail to discriminate between requests with similar or close consensus scores but different weights.

Consensus+Expectation (C+E$_k$): Algorithms E and C both have advantages and limitations. Algorithm C+E$_k$ attemps to combine their strengths while minimizing their drawbacks. Its key idea is to run algorithm C first to identify a small set of k promising requests and to run the naive version of E to discriminate between them in a precise fashion. Letting $nbSamples = S_c + kS_e$, the algorithm can be specified as follows:

CHOOSEREQUEST-C+E$_k(J, t)$
1 **for** $j \in J$
2   **do** $c(j) \leftarrow 0$;
3 **for** $i \leftarrow 1 \ldots S_c$
4   **do** $R \leftarrow J \cup$ GETSAMPLE$([t+1, t+\Delta])$;
5       $\sigma \leftarrow$ OPTIMALSCHEDULE$(R, [t, t+\Delta+d])$;
6       $c(\sigma(t)) \leftarrow c(\sigma(t)) + 1$;
7 $P = argmax(k)(j \in J)\ c(j)$;
8 **for** $j \in P$
9   **do** $f(j) \leftarrow 0$;
10 **for** $i \leftarrow 1 \ldots S_e$
11   **do** $R \leftarrow P \cup$ GETSAMPLE$([t+1, t+\Delta])$;
12       $T \leftarrow [t+1, t+1+\Delta+d]$;
13   **for** $j \in P$
14   **do** $\sigma \leftarrow$ OPTIMALSCHEDULE$(R \setminus \{j\}, T)$;
15             $f(j) \leftarrow f(j) + w(j) + w(\sigma)$;
16 **return** $argmax(j \in P)\ f(j)$;

Low		Medium		High	
Weight	Rank	Weight	Rank	Weight	Rank
5	1	600	2	1000	3
10	4	800	5	2000	6
20	7	500	8	1500	9
1	10	700	11	1200	12
15	13	750	14	3000	15
50	16	550	17	2500	18
30	19	650	20	3500	21
25	22	400	23	4000	24
40	25	450	26	5000	27
3	28	575	29	4500	30

Table 1: Job Classes

Observe line 7 which collects in P the k jobs with the largest counts. Th rest of the algorithm simply applies algorithm E on the jobs in P. Algorithm C+E$_k$ applies OPTIMALSCHEDULE $S_c + kS_e = nbSamples$ times.

Experimental Results

This section compares the various approaches on the packet scheduling problems. More precisely, it evaluates the impact of various choices of parameters on the quality of the schedule. In practice, these parameters depend on the specifics of the application (e.g., the available time to make a decision) and the problem features (e.g., the number of classes and the time windows).

The experimental results are based on the problems of (Chang, Givan, & Chong 2000). In these problems, the arrival distributions are specified by independent HMMs, one for each job class. Each HMM has three states which correspond to low, medium, and high job arrival rates. The transitions between the states are drawn uniformly in the interval [0.9, 1.0]. In the low/medium/high arrival rate, the probability of generating a job is drawn uniformly from the intervals [0.0,0.01], [0.2,0.5], and [0.7, 1.0] respectively. To create problems that are challenging enough, it is important to normalize the HMMs. First, each set of classes (low, medium, and high) should have roughly the same number of jobs generated in the long run. Thus, each HMM is simulated for a long period of time and the job generation probabilities of the HMMS are normalized such that $E[low] \approx E[medium] \approx E[high]$. Furthermore, too few and too many jobs make the problems easy, so the job generation probabilities are further normalized so that roughly 1.5 tasks arrive at each time unit.

The classes of jobs are also divided in three categories corresponding to low, medium, and high weights. Table 1 depicts the weights of the various jobs classes are shown. A rank number is also associated with each class to show

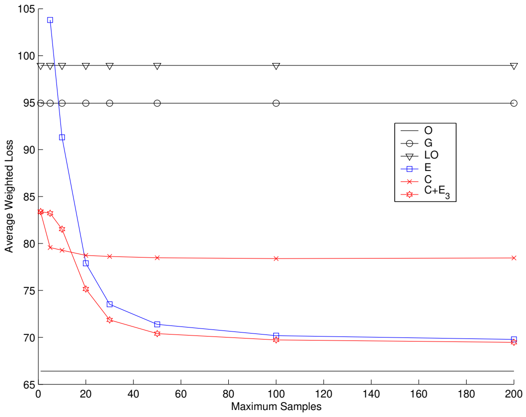

Figure 3: Impact on Sample Sizes

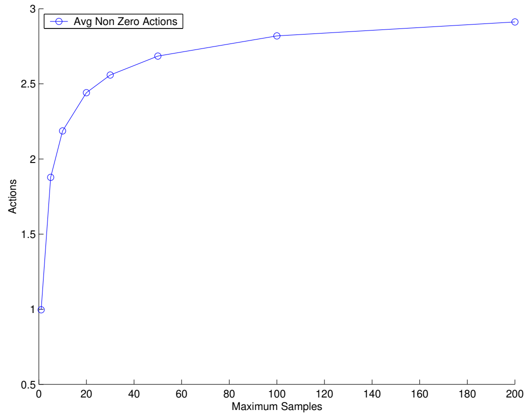

Figure 5: Number of Promising Actions wrt Sample Sizes

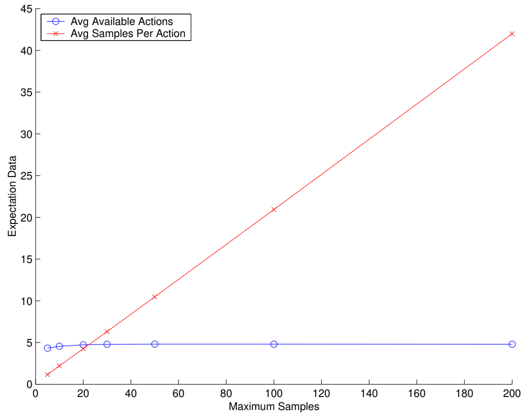

Figure 4: Samples per Actions wrt Sample Sizes

which weights match which classes. A problem with k classes includes the classes with rank numbers $1..k$. For example, problems with 7 classes use the classes with ranks $1..7$ and weights $(5, 10, 20, 600, 800, 1000, 2000)$. These are exactly the same problems as those in (Chang, Givan, & Chong 2000).

All experimental results are given for an online schedule consisting of 200,000 time steps. It is thus unrealistic to sample the future for so many steps (since the algorithm is quadratic in the time horizon). In general, the sampling horizon Δ (which corresponds to the depth of sampling in the POMDP algorithms) is chosen to be 50 (which seems to be an excellent compromise between time and quality when $d = 20$). The impact of the sampling horizon is also analyzed in detail.

Impact of Sample Size

Figure 3 compares the various approaches on the 7-class problem from (Chang, Givan, & Chong 2000) in terms of the sample sizes. In these problems, the time window size d is 20 and the sampling horizon Δ is 50. Since these problems have a small number of classes, they should be favorable to algorithm E (e.g., when compared to vehicle routing problems which have many more actions and which are more demanding computationally). The results depict the packet loss for each of the algorithms as a function of the sample sizes (i.e., small values denote a better algorithm) as well as the offline solution which denotes the theoretical best solution. The results are interesting in many ways. First, they clearly indicate the value of stochastic information, since algorithms E, C, and C+E$_3$ are clearly superior to the oblivious algorithms. Second, observe that algorithm LO is dominated by the greedy heuristic. This confirms once again that local optimization may not be a good approach to online problems, since it may lead to solutions that cannot accommodate future requests easily (Bent & Van Hentenryck 2001; 2003). Finally, and perhaps most importantly, the results demonstrate the value of consensus on the packet scheduling problems. *In particular,* C *is the best method when only few samples can be solved before decision time and* C+E$_k$ *dominates* E *for all sample sizes and significantly so when only few samples are considered. It is also worth pointing out that the consensus approach reaches its best value very quickly. In other words, it does not need many samples to identify the most promising packets to schedule.*

Figure 4 depicts the number of average actions considered with respect to the number of samples (and the number of samples available per action) for algorithm E. The number of actions may vary with the sample sizes because E's decisions may differ with the sample size. They indicate that there are about 5 classes of packets to consider at each time step for all sample sizes. The number of average classes to consider is slightly lower when there are few samples to work with. Figure 5 depicts the number of promising actions suggested by Algorithm C as the sample size increases, i.e., the number of packet classes which are scheduled first at time t. The figure indicates that there are at most 3 classes to consider, justifying why there is no need to go beyond algorithm C+E$_3$ on average on these problems.

Impact of the Sampling Horizon

Figures 6, 7, and 8 evaluates the impact of the sampling horizon Δ. They consider the 7-class problem with $d = 20$ and *nbSamples=20*, and evaluates the impact of varying Δ. Fig-

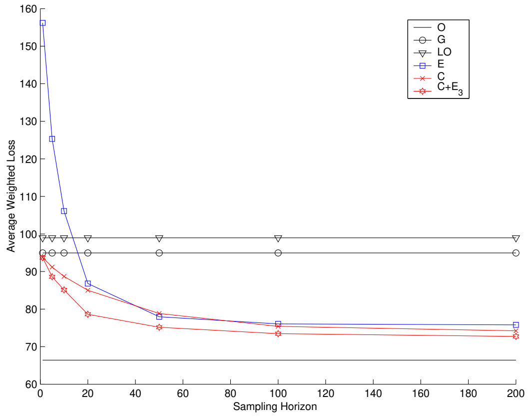

Figure 6: Impact of Sample Horizon on the Weighted Loss

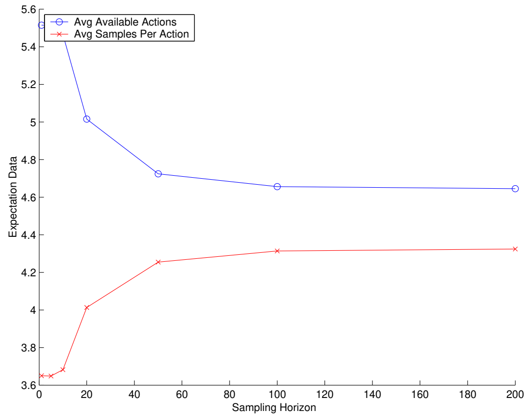

Figure 7: Samples per Actions wrt the Sampling Horizon

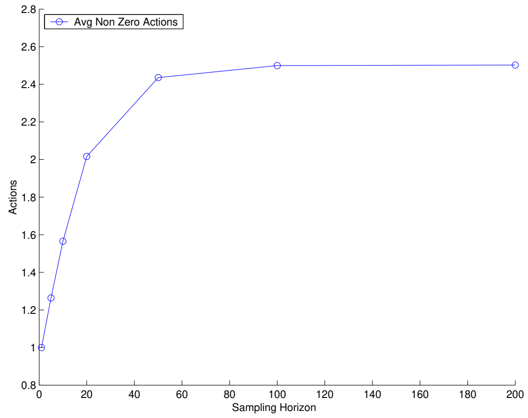

Figure 8: Number of Promising Actions wrt the Sampling Horizon

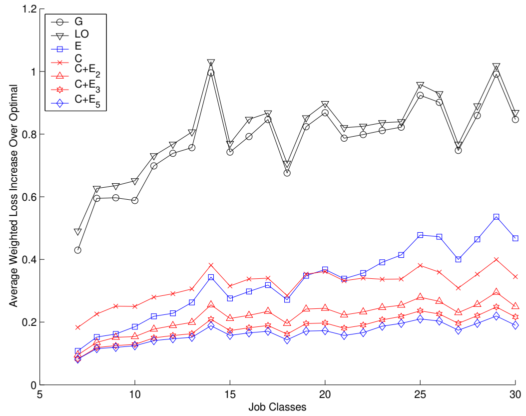

Figure 9: Impact of the Number of Actions

ure 6 shows that a reasonable schedule horizon is critical for E. For low schedule horizons, E is worse than the oblivious algorithms and it only becomes comparable to C when $\Delta = 20$. More generally, our results indicates that E needs Δ to be not smaller than d to perform reasonably. Note also that there is little change after $\Delta = 50$ and that C+E_3 is very robust wrt the sampling horizon. Figure 7 shows the average number of actions for E as a function of the sampling horizon. Note how the number of actions sharply decreases initially when the schedule horizon is increased from 0 to 20 and then 40 time steps. Similarly, Figure 8 depicts the number of promising actions identified by C. Once again, the number of actions sharply increases when Δ moves from 0 to 20 and then 40. It is basically stable after that.

Once again, these results indicate the value of consensus which makes the approach much less sensitive to the sampling horizon and always outperforms the oblivious algorithms. This is not the case of Algorithm E *which exhibits pathological behavior for low sampling horizons.*

Impact of the Number of Classes

Figure 9 depicts the impact of the number of packet classes on the behaviour of the algorithms. The stochastic algorithms given a fixed number of samples (*nbSamples*=30) and the sampling horizon is 50. The figure describes the quality of the algorithms (i.e., the weighted loss) as the number of classes grow and depicts their average weighted loss as a percentage over the theoretical offline optimum. The number of samples (*nbSamples*=30) was chosen so that Algorithm E has at least one sample per action (i.e., when there are 30 classes). The results show that, as the number of classes increases, Algorithm C gets closer to Algorithm E, outperforms it as soon as the number of classes reaches 20, and then produces significant improvements. Note also that Algorithm C+E_3 outperforms C and E but the distance wrt C decreases as the number of classes increases. *Once again, these results shed light on when the consensus approach (and its hybridization with the expectation approach) is beneficial. They indicate that the benefits of the consensus approach are more dramatic as the number of classes increases.* Note also that algorithm C remains between 20 and 25% from the optimum. Algorithm E starts as 10% off the optimum but it quality quickly decreases to 40% or more as the number of classes increases. The results also indicate that the hybrid algorithms are extremely robust as the number of classes increases.

Related Work

Online algorithms in computer science have been addressed for numerous years. A good source for an overview of such problems and techniques for solving them can be found in (Fiat & Woeginger 1998). Research in online algorithms has traditionally focused on techniques that are oblivious to any information about the future and many results are concerned with the worst case in the form of the competitive ratio (Karlin *et al.* 1988). It has only been recently that researchers have begun to address online problems where information about future uncertainty is known and how such information may increase the performance of algorithms. This includes scheduling problems (Chang, Givan, & Chong 2000), vehicle routing problems (Bent & Van Hentenryck 2003), (Bent & Van Hentenryck 2001), (Cambell & Savelsbergh 2002) and elevator dispatching (Nikovski & Branch 2003) to name a few. Research on these problems has varied widely, but the unifying theme is that knowing probabilistic information about the future significantly increases performance. The consensus approach proposed in this paper was motivated by online stochastic vehicle routing proposed in (Bent & Van Hentenryck 2003; 2001). In these algorithms, routing plans are generated continously during plan execution. They are ranked by a consensus function and the chosen plan is used to make the departure decisions. In these routing problems, only 2 to 10 scenarios can be solved in between two events and the number of events is large (e.g., 50 different requests).

Conclusion

This paper reconsidered the online packet scheduling in computer networks proposed in (Chang, Givan, & Chong 2000) and made a number of contributions. First, it proposed a consensus approach to sampling and showed experimentally how the consensus approach outperforms the expectation approach from (Chang, Givan, & Chong 2000) when the number of samples that can be solved at decision time is small. Second, the paper shows how the consensus and expectation approaches may synergize to produce an effective hybrid algorithm outperforming both approaches. The hybrid algorithm uses the consensus approach to select the most promising actions which are then evaluated using the expectation approach.

More generally, the paper confirms the value of consensus in online stochastic optimization. It indicates the consensus is very effective whenever the number of samples that can be solved before decision time is small, which occurs in many practical problems and justifies the design of online vehicle routing algorithms (Bent & Van Hentenryck 2003; 2001). It also indicated that the consensus methodology synergizes with the expectation approach even when large number of samples are available.

There are numerous directions for future directions. First, it would be interesting to scale these algorithms to more complex packing scheduling where the offline problems is no longer polynomial, as well as more complicated scheduling applications. Second, these algorithms are dependent to a certain degree on the sampling accuracy. It would be interesting to integrate learning techniques to approximate the underlying distributions of the job arrivals (and perhaps to change the distributions over time), as well as to determine the robustness of the approaches and how well the approach they adapt to changes. Finally, it would be interesting to determine if the consensus approach, and its hybridization with the expectation algorithm, are relevant, and bring benefits for, general POMDPs and whether theroretical results can be obtained on the consensus approach for some classes of problems.

References

Bent, R., and Van Hentenryck, P. 2001. Scenario Based Planning for Partially Dynamic Vehicle Routing Problems with Stochastic Customers. *Operations Research*. (to appear).

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

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

Chang, H.; Givan, R.; and Chong, E. 2000. On-line Scheduling Via Sampling. *Artificial Intelligence Planning and Scheduling (AIPS)* 62–71.

Fiat, A., and Woeginger, G. 1998. *Online Algorithms: The State of the Art*.

Karlin, A.; Manasse, M.; Rudolph, L.; and Sleator, D. 1988. Competitive Snoopy Caching. *Algorithmica* 3:79–119.

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

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

McAllester, D., and Singh, S. 1999. Approximate Planning for Factored POMDPs Using Belief State Simplification. *Uncertainity in AI (UAI)*.

Nikovski, D., and Branch, M. 2003. Marginalizing Out Future Passengers in Group Elevator Control. *Uncertainity in Artificial Intelligence (UAI)*.

Stefik, M. 1981. Planning with Constraints (MOLGEN: Part 1). *Artificial Intelligence* 16:111–139.

Villareal, F., and Bulfin, R. 1983. Scheduling a Single Machine to Minimize the Weighted Number of Tardy Jobs. *IEEE Transactions* 337–343.