Online Stochastic and Robust Optimization

Russell Bent and Pascal Van Hentenryck

Brown University, Box 1910, Providence, RI 02912

Abstract. This paper considers online stochastic optimization problems where uncertainties are characterized by a distribution that can be sampled and where time constraints severely limit the number of offline optimizations which can be performed at decision time and/or in between decisions. It reviews our recent progress in this area, proposes some new algorithms, and reports some new experimental results on two problems of fundamentally different nature: packet scheduling and multiple vehicle routing (MVR). In particular, the paper generalizes our earlier generic online algorithm with precomputation, least-commitment, service guarantees, and multiple decisions, all which are present in the MVR applications. Robustness results are also presented for multiple vehicle routing.

1 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.

This research considers an online stochastic optimization framework which assumes the distribution of future requests, or an approximation thereof, is a black-box available for sampling. This is typically the case in many applications, where either historical data and predictive models are available. The framework assumes that the uncertaintly does not depend on decisions, an assumption which holds in a great variety of applications and has significant computational advantages. Indeed, there is no need to explore trees of scenarios and/or sequences of decisions. In addition, this research focus primarily on online stochastic optimization under time constraints, which assumes that the time to make a decision is severely constrained, so that only a few offline optimizations can be performed at decision time and/or in between decisions. Online problems of this kind arise in many applications, including vehicle routing, taxi dispaching, packet scheduling, and online deliveries.

The paper reviews our recent progress in that area, proposes some new algorithms, generalizes existing ones to accommodate a variety of significant functionalities, and reports some new experimental results. All results are presented in a unified framework, abstracting the contributions spread accross multiple papers and crystallizing the intuition beyond the algorithmic design decisions.

The starting point is the generic online algorithm, initially proposed in [4], which can be instantiated to a variety of oblivious and stochastic approaches. When no time constraints are present, the generic algorithm naturally leads to the "traditional" expectation algorithm and to a novel hedging algorithm that provides an online counterpart to robust optimization [10]. When time constraints are present, the critical issues faced by the online algorithms is how to use their time wisely, since only a few scenarios can be optimized within the time constraints. The generic algorithm can then be instantiated to produce consensus [4] and regret [5], two algorithms which approximate expectation.

The generic online algorithm can be elegantly generalized to accommodate many features that are critical in practical applications. In particular, the paper shows how it can incorporate precomputation (to make immediate decisions), least-commitment (to avoid suboptimal decisions), service guarantees (to serve all accepted requests), and agregate decisions (to serve several requests simultaneously).

Various instantiations of the generic online algorithm are evaluated experimentally on two fundamentally different applications: packet scheduling and multiple vehicle routing. These two applications represent two extremes in the landscape of online stochastic optimization. Packet scheduling is of interest because of its simplicity: its offline problem is polynomial and the number of possible actions at each time step is small. As a consequence, it is possible to study how consensus and regret approximate expectation and hedging, as well as how all these algorithms behave under severe and less severe time constraints. Multiple vehicle routing is of interest because of its complexity: its offline problem is NP-hard and it features many of the modeling complexities of practical applications.

The rest of this paper is organized as follows. Sections 2 and 3 present the online stochastic framework and the generic online algorithm. Section 4 presents stochastic algorithms for loose time constraints and Section 5 shows how these algorithms can be approximated by consensus and regret under strict time constraints. Section 6 compares the algorithms on packet scheduling under various time constraints. Section 7 generalizes the online algorithm to incorporate precomputation, service guarantees, least-commitment and pointwise consensus/regret. Finally, Section 8 presents experimental results of the generalized algorithm to a complex multiple vehicle routing applications.

2 The Online Stochastic Framework

The Offline Problem The framework assumes a discrete model of time. 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.

The Online Problem The online algorithms have at their disposal a procedure to solve, or approximate, the offline problem. They have also access to the distribution of future requests. The distribution is seen as a black-box and is available for sampling. In practice, it may not be practical to sample the distribution for the entire time horizon and hence the sizes of the samples is an implementation parameter.

ONLINEOPTIMIZATION(H)
$R \leftarrow \emptyset$;
$w \leftarrow 0$;
for $t \in H$
do $R \leftarrow$ AVAILABLEREQUESTS$(R, t) \cup$ NEWREQUESTS(t);
  $r \leftarrow$ CHOOSEREQUEST(R, t);
  SERVEREQUEST(r, t);
  $w \leftarrow w + w(r)$;
  $R \leftarrow R \setminus \{r\}$;

Fig. 1. The Generic Online Algorithm

Time Constraints Practical applications often include severe time constraints on the decision time and/or on the time between decisions. To model this requirements, 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 [6] and partially observable Markov decision processes (POMDPs) [9]. The simplicity of the framework also allows us to prove some nice theoretical properties of the algorithms. These will be described in a forthcoming paper.

3 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 OPTIMALSOLUTION(R, t, Δ) that, given a set R of requests, a time t, and a number Δ, returns an optimal solution for R over $[t, t + \Delta]$;
2. a function GETSAMPLE$([t_s, t_e])$ that returns a set of requests over the interval $[t_s, t_e]$ by sampling the arrival distribution.

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)
$A \leftarrow$ READY(R, t);
return $argmax(r \in A)\ w(r)$;

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)
$\sigma \leftarrow$ OPTIMALSOLUTION(R, t);
return $\sigma(t)$;

4 Online Stochastic Algorithms without Time Constraints

This section reviews two algorithms exploiting stochastic information. The first algorithm optimizes expectation, while the second one hedges against the worst-case scenario. These algorithms are appropriate when time constraints are loose (i.e., when $\mathcal{O}$ is large enough to produce high-quality results).

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 $A \leftarrow$ READY(R, t);
2 **for** $r \in A$
3   **do** $f(r) \leftarrow 0$;
4 **for** $i \leftarrow 1 \ldots \mathcal{O}/|A|$
5   **do** $S \leftarrow R\ \cup$ GETSAMPLE($[t+1, t+\Delta]$);
6   **for** $r \in A$
7   **do** $f(r) \leftarrow f(r) + (w(r) + w($OPTIMALSOLUTION$(S \setminus \{r\}, t+1)))$;
8 **return** $argmax(r \in A)\ f(r)$;

Line 1 computes the requests which can be served at time t. Lines 2-3 initialize the evaluation function $f(j)$ for each request r. The algorithm then generates a number of samples for future requests (line 4). For each such sample, it computes the set R of all available and sampled requests at time t (line 5). The algorithm then considers each available request r successively (line 6), it implicitly schedules r at time t, and applies the optimal offline algorithm using $S \setminus \{r\}$ and the time horizon. 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 $r \in A$ with the highest evaluation. Observe Line 4 of Algorithm E which distributes the available offline optimizations across all available requests.

Hedging (H): Algorithm H is an online adaptation of robust optimization, whose key idea is to hedge against the worst-case scenario. In other words, the goal is to find, at each time step, a solution whose deviation with respect to the optimal solution is minimal over all scenarios.

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* DEVIATION(r, R, t)*, is defined as*

$$\mid w(\text{OPTIMALSOLUTION}(R,t)) - (w(r) + w(\text{OPTIMALSOLUTION}(R \setminus \{r\}, t+1))) \mid .$$

The differences between algorithms E and H are Line 7 which computes the maximum deviation for the action and Line 8 which selects the action with minimum deviation.

```
CHOOSEREQUEST-H(R, t)
1  A ← READY(R, t);
2  for r ∈ A
3      do f(r) ← 0;
4  for i ← 1 ... O/|A|
5      do S ← R ∪ GETSAMPLE([t + 1, t + Δ]);
6         for r ∈ A
7             do f(r) ← max(f(r), DEVIATION(r, R, t));
8  return argmin(r ∈ A) f(r);
```

Observe that algorithm E can be obtained from algorithm H by replacing line 7 with

$$f(r) \leftarrow f(r) + \text{DEVIATION}(r, R, t);$$

since E can be viewed as minimizing the average deviation.

5 Online Stochastic Algorithms under Time Constraints

This section studies online optimization under time constraints, i.e., when the number of optimizations at each time step t is small. As mentioned earlier, algorithms E and H distribute the available optimizations $\mathcal{O}$ across all requests (line 4). 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 algorithms do not yield much information. This is precisely why online vehicle routing algorithms [2] cannot use E and H, since the number of requests is very large (about 50 to 100), the time between decisions is relatively short, and optimization is computationally demanding. The section shows how algorithm E can be approximated and presents two approximation algorithms, consensus and regret. The regret algorithm can also be adaped to approximate algorithm H.

Consensus (C): The consensus algorithm C was introduced in [4] as an abstraction of the sampling method used in online vehicle routing [2]. Its key idea is to solve each scenario once and thus to examine $\mathcal{O}$ scenarios instead of $\mathcal{O}/|A|$. 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 ∈ R
2      do f(r) ← 0;
3  for i ← 1 . . . O
4      do S ← R ∪ getSample([t + 1, t + Δ]);
5         σ ← optimalSolution(S, t);
6         f(σ(t)) ← f(σ(t)) + w(σ);
7  return argmax(r ∈ 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. The regret algorithm shows how to gather that kind of information from the sample solutions without solving additional optimization problems.[1]

Regret (R): The key insight in 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). *A regret is a function that, given a request r, a set R ($r \in R$), a time t, and an optimal solution $\sigma =$ OPTIMALSOLUTION(R, t), over-approximates the deviation of r wrt R and t, i.e.,*

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

Moreover, there exists two functions f_o and f_r such that

- OPTIMALSOLUTION(R, t) *runs in time* $O(f_o(R, \Delta))$;
- REGRET(r, R, t, σ) *runs in time* $O(f_r(R, \Delta))$;
- $|A| f_r(R, \Delta)$ *is* $O(f_o(R, \Delta))$.

[1] The 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.

Intuitively, the complexity requirement states that the computation of the $|A|$ 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 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 offline 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** $i \leftarrow 1 \ldots \mathcal{O}$
4   **do** $S \leftarrow R \,\cup$ GETSAMPLE$([t+1, t+\Delta])$;
5         $\sigma \leftarrow$ OPTIMALSOLUTION(S, t);
6         $f(\sigma(t)) \leftarrow f(\sigma(t)) + w(\sigma)$;
7   **for** $r \in$ READY$(R, t) \setminus \{\sigma(t)\}$
8   **do** $f(r) \leftarrow f(r) + (w(\sigma) -$ REGRET$(\sigma, r, R, t))$;
9 **return** $argmax(r \in R)\ f(r)$;

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., $w(\sigma) -$ REGRET(σ, r, R, t). Hence every available request is given an evaluation for every sample at time t for the cost of a single offline optimization (asymptotically). Observe that algorithm R performs $\mathcal{O}$ offline optimizations at time t and that it is easy to adapt algorithm R to approximate algorithm H.

6 Packet Scheduling

This section reports experimental results on the online packet scheduling problem studied in [8]. 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 (MMs).

The Offline Problem 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

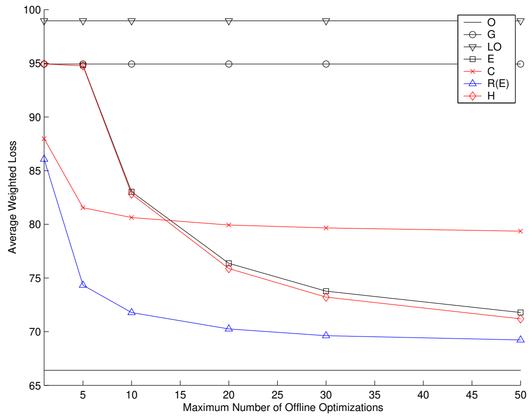

Fig. 2. The Regret Algorithm on Packet Scheduling

$[a(j), a(j) + d]$, 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

$$\forall\, t_1, t_2 \in H : t_1 \neq t_2 \;\rightarrow \sigma(t_1) \neq \sigma(t_2)$$
$$\forall\, t \in H : a(\sigma(t)) \leq t \leq a(\sigma(t)) + d.$$

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 [8, 4], where all the details can be found. In these problems, the arrival distributions are specified by independent MMs, one for each job class. The results are given for the reference 7-class problems and for an online schedule consisting of 200,000 time steps. Because it is unpractical 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 [5] and consists of swapping a constant number of packets in the optimal schedule.

Experimental Results Figure 2 depicts the average packet loss as a function of the number of available optimizations $\mathcal{O}$ for the various algorithms on a variety of 7-class problems. It also gives the optimal, a posteriori, packet loss (O). The results indicate the value of stochastic information as algorithms E and H significantly outperform the oblivious algoritms G and LO and bridge much of the gap between these algorithms and the optimal solution. Note that LO is worse than LO, illustrating the (frequent) pathological behavior of over-optimizing. Hedging slightly outperforms expectation, although the results are probably not statistically significant.

The results also indicate that consensus outperforms E and H whenever few optimizations are available (e.g., ≤ 15). The improvement is particularly significant when there are very few available optimizations. Consensus is dominated by E and H when the number of available optimizations increases, although it still produces significant improvements over the oblivious algorithms. This is of course pertinent, since E and H are 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 offline optimizations (strong time constraints) and expectation/hedging even when there are a reasonably large number of them, (weak time constraints). Reference [5] also shows this to be the case for complex online vehicle routing with time windows.

7 The Online Stochastic Algorithm Revisited

This section considers four important generalizations to the framework: precomputation, service guarantees, least-commitment, and multiple decisions.

Precomputation Some applications are characterized by very short decision times, either because of problem requirements or to produce solutions of higher quality. These applications however allow for some limited number of optimizations in between decisions. For instance, online vehicle routing and deliveries are applications exhibiting these features. The generic online algorithm can generalized to provide these functionalities. The key idea is to maintain a set of scenario solutions during execution. At decision time, these solutions can then be used to choose an appropriate request to serve. The set of solutions can then be updated to remove solutions that are incompatible with the selected decisions and to include newly generated solutions.

Figure 3 depicts the generalized online algorithm and shows how to instantiate it with consensus. The set of solutions Σ is initialized in Line 2. The request is selected in line 5 by function CHOOSEREQUEST which now receives Σ as input as well. Lines 9 and 10 remove the infeasible solutions and generates new ones. The function GENERATESOLUTIONS is also depicted in Figure 3. It is essentially the core of the CHOOSEREQUEST implementation in algorithms C and R with the logic to make decisions abtracted away. The decision code is what is left in the instantiations of function CHOOSEREQUEST. The figure also gives the implementation of CHOOSEREQUEST for algorithm C to illustrate the instantiations.

Service Guarantees Many applications require service guarantees. The algorithm may decide to accept or reject a new request but, whenever a request is accepted, the request must be served. The online algorithm can be enhanced to include service guarantees. It suffices to introduce a new function to accept/request new requests and to keep only those solutions which can accommodate the requests. Of course, to accept a request, at least one solution must be able to serve it in addition to the current requests. The new

```
onlineOptimization(H, R)
 1  w ← 0;
 2  Σ ← generateSolutions(R, 0);
 3  for t ∈ H
 4  do R ← availableRequests(R, t) ∪ newRequests(R, t);
 5     r ← chooseRequest(R, t, Σ);
 6     serveRequest(r, t);
 7     w ← w + w(r);
 8     R ← R \ {r};
 9     Σ ← {σ ∈ Σ | σ(t) = r};
10     Σ ← Σ ∪ generateSolutions(R, t);
```

```
generateSolution(R, t)
1  Σ ← ∅;
2  repeat
3        S ← R ∪ getSample([t + 1, t + Δ]);
4        σ ← optimalSolution(S, t);
5        Σ ← Σ ∪ {σ};
6    until until time t + 1
7  return Σ;
```

```
chooseRequest-C(R, t, Σ)
1  for r ∈ R
2  do f(r) ← 0;
3  for σ ∈ Σ
4  do f(r) ← f(r) + w(σ);
5  return argmax(r ∈ R) f(r);
```

Fig. 3. The Generic Online Algorithm with Precomputation

online generic algorithm with service guarantees is depicted in Figure 4. The changes are in lines 4-6. Function ACCEPTREQUESTS (line 4) selects the new requests to serve using the existing solutions Σ and function REMOVEINFEASIBLESOLUTIONS removes those solutions which cannot accommodate the new requests.

Least-Commitment In the packet scheduling application, it is always suboptimal not to serve a packet at each time step. However, in many online applications, it may be advisable not to serve a specific request, since this may reduce further choices and/or make this algorithm less adaptive. The ability to avoid or to delay a decision is critical in some vehicle routing applications, as shown later in the paper. It is easy to extend the framework presented so far to accommodate this feature. At every step, the algorithm may select a request $\perp$ which has no effect and no profit/cost. It suffices to use CHOOSEREQUEST$(R \cup \{\perp\}, t, \Sigma)$ in line 5 of the algorithm.

Multiple Decisions and Pointwise Consensus Many practical applications have the ability to serve several requests at the same time, since resources (e.g., machines or vehicles) are often available in multiple units. The online algorithm naturally generalizes to multiples decisions. Assume that a solution σ at time t returns a tuple

ONLINEOPTIMIZATION(H, R)
$w \leftarrow 0;$
$\Sigma \leftarrow$ GENERATESOLUTIONS$(R, 0);$
for $t \in H$
do $N \leftarrow$ ACCEPTREQUESTS$(R, t, \Sigma);$
  $\Sigma \leftarrow$ REMOVEINFEASIBLESOLUTIONS$(R, t, N, \Sigma);$
  $R \leftarrow$ AVAILABLEREQUESTS$(R, t) \cup N;$
  $r \leftarrow$ CHOOSEREQUEST$(R, t, \Sigma);$
  SERVEREQUEST$(r, t);$
  $w \leftarrow w + w(r);$
  $R \leftarrow R \setminus \{r\};$
  $\Sigma \leftarrow \{\sigma \in \Sigma \mid \sigma(t) = r\};$
  $\Sigma \leftarrow \Sigma \cup$ GENERATESOLUTIONS$(R, t);$

Fig. 4. The Generic Online Algorithm with Precomputation and Service Guarantees

$\sigma(t) = (r_1, \ldots, r_n) = (\sigma_1(t), \ldots, \sigma_n(t))$. It suffices to replace r in the online algorithm by a tuple $(r_1, \ldots, r_n)$ to obtain a generic algorithm over tuples of decisions. However, it is important to reconsider how to choose requests in this new context. A straighforward generalization of consensus would give

CHOOSEREQUEST-C(R, t)
for $e \in R^n$
  do $f(e) \leftarrow 0;$
for $i \leftarrow 1 \ldots \mathcal{O}$
  do $S \leftarrow R \cup$ GETSAMPLE$([t + 1, t + \Delta]);$
    $\sigma \leftarrow$ OPTIMALSOLUTION$(S, t);$
    $f(\sigma(t)) \leftarrow f(\sigma(t)) + w(\sigma);$
return $argmax(e \in R^n)\ f(e);$

Unfortunately, this generalized implementation of consensus is not particularly effective, especially when there are many requests and few scenarios. Indeed, the information about decisions is now distributed over tuples of requests instead of over individual requests and consensus does not capture the desirability of serving particular requests. This limitation can be remedied by evaluating the decisions independently accross all scenarios and by selecting the best coupling available among the solutions. This *point-wise consensus* can be formalized as follows:

CHOOSEREQUEST-PC(R, t)
for $r \in R, i \in 1..n$
  do $f_i(r) \leftarrow 0;$
for $i \leftarrow 1 \ldots \mathcal{O}$
  do $S \leftarrow R \cup$ GETSAMPLE$([t + 1, t + \Delta]);$
    $\sigma \leftarrow$ OPTIMALSOLUTION$(S, t);$
    for $i \in 1..n$
      do $f_i(\sigma_i(t)) \leftarrow f_i(\sigma_i(t)) + w(\sigma);$
$\sigma^* = argmax(\sigma \in \Sigma)\ \sum_{i=1}^{n}\ f_i(\sigma_i(t));$
return $\sigma^*(t);$

Note that pointwise consensus reduces to consensus when $n = 1$ and that pointwise regret could be derived in the same fashion.

8 Vehicle Routing

This section describes the applications of the online generic algorithm with precomputation, service guarantees, and least-commitment to a multiple vehicle routing applications. Contrary to the applications in [2] where the focus is on feasibility, the difficulty here lies in the lexicographic objective function, i.e., serving as many customers as possible and minimizing travel distance. The interesting result is that approximations of expectation perform remarkably in these two "orthogonal" applications.

The Problem The application is based on the model proposed in [11] where customers are distributed in a 20km×20km region and must be served by vehicles with uniform speed of 40 km/h. Service times for the customers are generated according to a log-normal distribution with parameters (.8777, .6647). With this distribution, the mean service time is 3 min. and the variance is 5 min. The service times were chosen to mimic the service times of long-distance courier mail services [11]. We use n to denote the expected number of customers and T to denote the time horizon (8 hours). Problems are generated with a degree of dynamism (DOD) (i.e, the ratio of known customers over stochastic customers) in the set $\{0\%, 5\%, \ldots, 100\%\}$. For a DOD x, there are $n(1-x)$ known customers. The remaining customers are generated using an exponential distribution with parameter $\lambda = \frac{nx}{T}$ for their inter-arrival times. It follows from the corresponding Poisson distribution (with parameter λT) that the expected number of unknown customers is nx, the expected number of customers is n, and the expected DOD is x. The results given here assume that 4 vehicles amd 160 customers. Each vehicle can serve at most 50 customers and the vehicle must return to the depot by the time horizon. The customers are generated using 2-D Gaussians centered at two points in the region. (Similar results are obtained under other distributions). The objective function consists in minimizing the number of missed customers and minimizing the travel distance. The experimental results are based on 15 instances and an average of 5 runs on each instances. See Reference [3] for a more comprehensive evaluation.

Setting of the Algorithms The online generic algorithm is run with the following settings. Initially, 25 different scenarios are created and optimized for 1 minute using large-scale neighborhood search (LNS) [12, 1]. These initial solutions are used to determine the first customer for each vehicle. An additional 25 scenarios are created and optimized for 1 minute with the first customers fixed. It was verified experimently that this second step improves the quality of the final solutions. Subsequent scenarios are optimized for about 10 seconds using LNS. The parameters for LNS are as follows: 30 for the maximum number of customers to remove at one time, 100 attempts at removing c customers without improvement before removing $c + 1$ customers, 15 for the determinism factor of the relatedness function, and 4 discrepancies. A simple insertion heuristic is used to decide whether a new request should be accommodated. The online algorithm uses precomputations to decide whether to accept requests immediately and to avoid

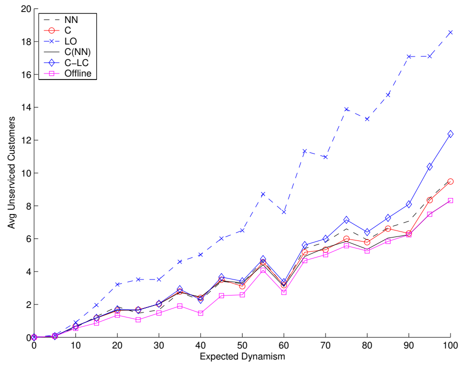

Fig. 5. Results on the Number of Serviced Customers

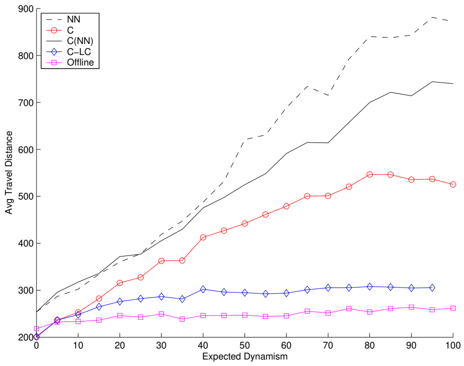

Fig. 6. Results on Travel Distance

delaying the dispatching of vehicles, service guarantees to serve all accepted requests, least-commitment to be able to postpone vehicle departures to accommodate future requests more effectively, and pointwise consensus to gather as much information as possible from the small number of scenarios available in this application.

Experimental Results The online generic algorithm is compared with the Nearest Neighbor (NN) heuristic proposed in [11] and generalized to providing guarantees on servicing customers. Whenever a request arrives, the NN algorithm is simulated to determine if it can accommodate the new request. If it cannot, the request is rejected. More generally, the results compare NN and the online algorithm instantiated with local optimization (LO), consensus (C), consensus with least-commitment (C-LC), and consensus using NN instead of LNS (C(NN)) to find solutions to the scenarios. The figures will also give the offline solution found using LNS, which represents the "best" solution the various online algorithm could hope to achieve.

Figure 5 describes the experimental results concerning the number of serviced customers for various degrees of dynamism. The results clearly indicate that the stochastic approaches are superior to LO which is unable to service as many customers. A detailed look at the trace of the decisions performed by LO indicate that it waits too long to deploy some of the vehicles. This is because optimal solutions use as few vehicles as possible to minimize travel distance and LO believes it can use fewer vehicles than necessary until late in the simulation. The remaining approaches service a comparable number of customers. With higher degrees of dynamism, the benefits of using a consensus function for ranking are clear, as it reduces the number of missed customers significantly compared to using travel distance. The online stochastic algorithm do not bring significant benefits in terms of serviced customers compared to NN. C(NN) is generally superior to NN, while C is roughly similar to NN (except for very high degrees of dynamism). Note that C-LC does not perform as well as C for these very high degrees of dynamism: It has a tendency to wait too long, which could be addressed easily by building some slack in C-LC.

Figure 6 depicts the results for the travel distance, which are extremely interesting. No results are given for LO, since it is far from being competitive for customer service. The results indicate that the stochastic instantiations of the online algorithm significantly reduce travel distance compared to NN. The results are particularly impressive for C-LC, whose travel distance is essentially not affected by the degree of dynamism. Observe also that the comparison between C(NN) and the other stochastic approaches tend to indicate that it seems beneficial for these problems to use a more sophisticated optimization algorithms on fewer samples than a weaker method on more samples.

Robustness It is natural to question how the algorithms behave when the stochastic information is noisy. This situation could arise from faulty historical data, predictions, and/or approximations in machine learning algorithms. Figure 7 shows some results when run on the 20% and 50% dynamism instances of M3 (32 and 80 expected new customers respectively). It is interesting to see that, in both cases, it is better to be optimistic when estimating the number of dynamic customers. For example, on 20% dynamism, C-LC is able to service roughly the same number of customers when it expects between 20 and 100 dynamic customers. However, it performs the best in terms of travel distance when it expects 50 dynamic customers, slightly more than the 32 of actual problem sets themselves. In addition, these results show that, even in the presence of significant noise, stochastic approaches are still able to achieve high-quality results.

Acknowledgments

This paper is dedicated to Jean-Louis Lassez for all these long conservations about science, scientific communities, and the joy of research. Happy birthday, Jean-Louis, and thank you for being such an inspiration for so many of us! This research is partially supported by NSF ITR Award DMI-0121495.

References

1. R. Bent and P. Van Hentenryck. A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows. To appear in *Transportation Science*.

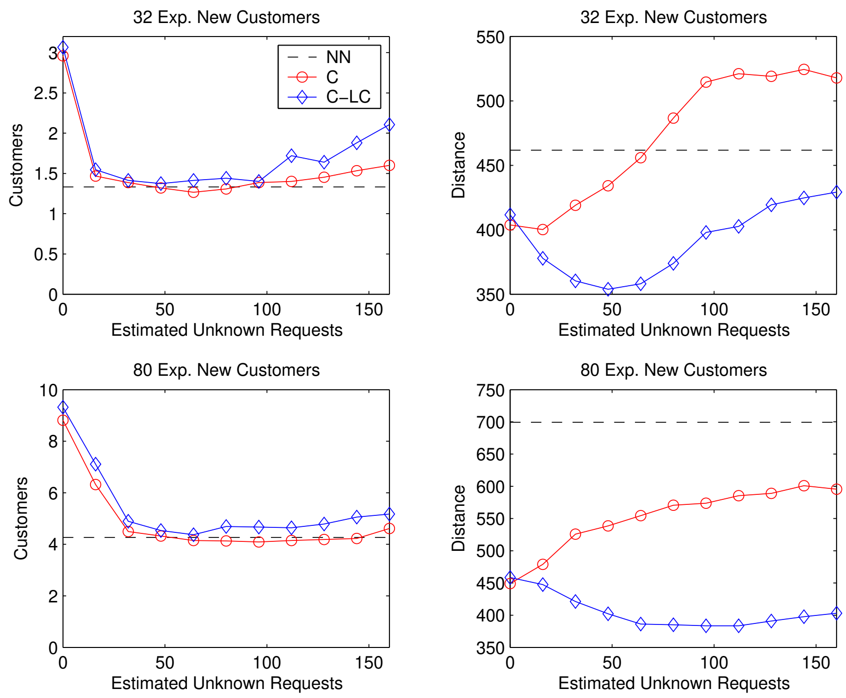

Fig. 7. Robustness Results

2. R. Bent and P. Van Hentenryck 2001. Scenario Based Planning for Partially Dynamic Vehicle Routing Problems with Stochastic Customers. *Operations Research*. (to appear).
3. R. Bent and P. Van Hentenryck. 2003. Dynamic Vehicle Routing with Stochastic Requests Technical Report CS-03-10, Brown University.
4. R. Bent and P. Van Hentenryck. 2004. The Value of Consensus in Online Stochastic Scheduling. In *ICAPS 2004*.
5. R. Bent and P. Van Hentenryck. 2004. Regrets Only! Online Stochastic Optimization under Time Constraints. In *AAAI 2004*.
6. J. Birge and F. Louveaux. 1997. Introduction to Stochastic Programming. Springer Verlag.
7. A. Cambell, and M. Savelsbergh. 2002. Decision Support for Consumer Direct Grocery Initiatives. *Report TLI-02-09, Georgia Institute of Technology*.
8. H. Chang, R. Givan, and E. Chong. 2000. On-line Scheduling Via Sampling. In *AIPS'2000*, 62–71.
9. L. Kaelbling, M. Littman, and A. Cassandra. 1998. Planning and Acting in Partially Observable Stochastic Domain. Artificial Intelligence, 101(1-2), 99–124.
10. P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Kluwer Academic Publishers, 1997.
11. A. Larsen, O. Madsen, and M. Solomon. Partially Dynamic Vehicle Routing-Models and Algorithms. *Journal of Operational Research Society*, 53:637–646, 2002.
12. P. Shaw. 1998. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *CP'98*, 417–431.