

# Online Stochastic Optimization Under Time Constraints

Russell Bent, Pascal Van Hentenryck, and Eli Upfal

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 proposes a generic framework for online stochastic optimization and several of its instantiations. In particular, it studies the expectation algorithm E that evaluates all choices against all samples at each decision step and introduces the consensus C and regret R algorithms that only solve each sample once per step. Both theoretical and experimental results are presented on the algorithms. The theoretical results indicate that, under reasonable and practical assumptions, the expected quality loss of algorithm E is  $o(1)$ , while algorithm R provides a  $\rho(1 + o(1))$ -approximation when its underlying regret function is a  $\rho$ -approximation. The experimental results are presented on three problems of fundamentally different nature: packet scheduling and multiple vehicle routing (MVR) with and without time windows. They clearly show the benefits of stochastic information and the practical value of the consensus and regret algorithms under severe time constraints.

## 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 sequence 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 paper 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 or predictive models are available. The framework assumes that the distribution of future requests is independent of current 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 dispatching, packet scheduling, and online deliveries.

This paper summarizes our recent progress in this area and presents new theoretical and experimental results. All results are presented in a unified framework, abstracting the contributions spread across multiple papers and crystallizing the intuition beyond the algorithmic design decisions. Its 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 E. When time constraints are present,

the critical issues faced by the online algorithms is how to use their time wisely, since only a few samples can be optimized within the time constraints. The paper then introduces two novel algorithms, i.e. consensus C and regret R, to approximate algorithm E with significantly fewer offline optimizations [4, 5]. In addition, the paper shows that the generic online algorithm can be elegantly generalized to accommodate many features that are critical in practical applications: precomputation (to make immediate decisions), least-commitment (to avoid suboptimal decisions), service guarantees (to serve all accepted requests), and aggregate decisions (to serve several requests simultaneously) [6].

The various instantiations of the generic online algorithm are evaluated theoretically and experimentally. The theoretical results show that, under reasonable and practical assumptions, the expected quality loss of algorithm E (when compared to the offline optimal solution) is  $o(1)$  with a total of  $n|R|\Omega(\log(n|R|))$  offline optimizations, where  $n$  is the number of decision steps and  $R$  are the possible choices at each iteration. Moreover, algorithm R is shown to be an expected  $\rho(1 + o(1))$ -approximation algorithm (whenever its underlying regret function is a  $\rho$ -approximation) and requires a total of  $n\Omega(\log(n|R|))$  offline optimizations. The experimental results evaluate the algorithms on three fundamentally different applications: packet scheduling and multiple vehicle routing with and without time windows. These three applications represent different 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, 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: their offline problems are NP-hard and feature many of the modeling complexities of practical applications. The experimental results clearly demonstrate the benefits of stochastic information and the practical value of the regret algorithm.

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 the expectation algorithm for loose time constraints and Section 5 shows how this algorithm can be approximated by consensus and regret under strict time constraints. Section 6 presents the theoretical analysis for the expectation and regret algorithms and Section 7 compares the algorithms experimentally on packet scheduling under various time constraints. Section 8 generalizes the online algorithm to incorporate precomputation, service guarantees, least-commitment and pointwise consensus/regret. Sections 9 and 10 present experimental results of the generalized algorithm on complex multiple vehicle routing applications. Sections 11 and 12 present related work and conclude the paper.

## 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  $\gamma$  maximizing  $W(\gamma) = \sum_{t \in H} w(\gamma(t))$ . 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

```

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

```

**Fig. 1.** The Generic Online Algorithm

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.

*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 [7] and partially observable Markov decision processes [12].

### 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  $t$  (i.e.,  $\gamma(t) \leftarrow r$ ). To implement function `CHOOSEREQUEST`, the algorithms have at their disposal two black-boxes:

1. a function `OPTIMALSOLUTION( $R, t, \Delta$ )` that, given a set  $R$  of requests, a time  $t$ , and a number  $\Delta$ , 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( $R, t$ )
1  $A \leftarrow \text{READY}(R, t)$ ;
2 return  $\text{argmax}(r \in A) w(r)$ ;

```

*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( $R, t$ )
1  $\gamma \leftarrow \text{OPTIMALSOLUTION}(R, t)$ ;
2 return  $\gamma(t)$ ;

```

## 4 Online Stochastic Optimization without Time Constraints

This section presents a stochastic algorithm that optimizes expectation. The algorithm is 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  $A \leftarrow \text{READY}(R, t)$ ;
2 for  $r \in A$ 
3   do  $f(r) \leftarrow 0$ ;
4 for  $i \leftarrow 1 \dots \mathcal{O}/|A|$ 
5   do  $S \leftarrow R \cup \text{GETSAMPLE}([t + 1, t + \Delta])$ ;
6     for  $r \in A$ 
7       do  $f(r) \leftarrow f(r) + (w(r) + W(\text{OPTIMALSOLUTION}(S \setminus \{r\}, t + 1)))$ ;
8 return  $\text{argmax}(r \in A) f(r)$ ;

```

Line 1 computes the requests which can be served at time  $t$  and removes dominated requests from consideration (which is important for performance reasons). Lines 2-3 initialize the evaluation function  $f(r)$  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.

## 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, algorithm E distributes 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 algorithm does not yield much information. This is precisely why online vehicle routing algorithms [2] cannot use algorithm E, 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. Before going into the algorithm, it is important to introduce the concept of local loss between serving an optimal and a given request at a specific step.

**Definition 1 (Local Loss).** Let  $R$  be the set of requests at time  $t$  and  $r \in R$ . The local loss of  $r$  wrt  $R$  and  $t$ , denoted by  $\text{LOCALLOSS}(r, R, t)$ , is defined as

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

*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 sample once and thus to examine  $\mathcal{O}$  samples 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  $\gamma$  is credited  $W(\gamma)$  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  $S \leftarrow R \cup \text{GETSAMPLE}([t + 1, t + \Delta])$ ;
5       $\gamma \leftarrow \text{OPTIMALSOLUTION}(S, t)$ ;
6       $f(\gamma(t)) \leftarrow f(\gamma(t)) + W(\gamma)$ ;
7  return  $\text{argmax}(r \in R) f(r)$ ;

```

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  $\gamma(t)$  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. 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.<sup>1</sup>

*Regret (R):* The key insight in Algorithm R is the recognition that, in many applications, it is possible to estimate the local loss of a request  $r$  at time  $t$  quickly. In other words, once the optimal solution  $\gamma$  of a sample is computed, it is easy to compute the local loss of all the requests, thus approximating E with one optimization. This intuition can be formalized using the concept of *regret*.

<sup>1</sup> 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.

**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  $\gamma = \text{OPTIMALSOLUTION}(R, t)$ , over-approximates the local loss of  $r$  wrt  $R$  and  $t$ , i.e.,

$$\text{REGRET}(r, R, t, \gamma) \geq \text{LOCALLOSS}(r, R, t).$$

Moreover, there exists two functions  $f_o$  and  $f_r$  such that

- $\text{OPTIMALSOLUTION}(R, t)$  runs in time  $O(f_o(R, \Delta))$ ;
- $\text{REGRET}(r, R, t, \gamma)$  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  $\gamma(t)$  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 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   $A \leftarrow \text{READY}(R, t)$ ;
2  for  $r \in A$ 
3      do  $f(r) \leftarrow 0$ ;
4  for  $i \leftarrow 1 \dots \mathcal{O}$ 
5      do  $S \leftarrow R \cup \text{GETSAMPLE}([t + 1, t + \Delta])$ ;
6           $\gamma \leftarrow \text{OPTIMALSOLUTION}(S, t)$ ;
7           $f(\gamma(t)) \leftarrow f(\gamma(t)) + W(\gamma)$ ;
8          for  $r \in A \setminus \{\gamma(t)\}$ 
9              do  $f(r) \leftarrow f(r) + (W(\gamma) - \text{REGRET}(\gamma, r, R, t))$ ;
10 return  $\text{argmax}(r \in A) f(r)$ ;

```

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.,  $W(\gamma) - \text{REGRET}(\gamma, r, R, t)$ . 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 Theoretical Analysis

This section analyzes the solution quality and the runtime performance of the algorithms. Both of these properties depend on the number of samples used by the function CHOOSEREQUEST in each iteration. A large number of samples results in high-quality solutions and long execution, while a small sample size may lead to solutions that are far from optimal. The main results in this section relate the sample size and the solution quality: They show that, under natural assumptions, a relatively

small number of samples per iteration suffices for achieving high solution quality in the expected sense. *The analysis is generic: it does not depend on any particular property of the input distribution. One can get significantly stronger results for well-behaved classes of input distributions.* The section first focuses on the expectation algorithm E, from which the results of the regret algorithm R can be derived.

*Expected Loss* Consider a run of the algorithm with an input sequence  $I = i_1, \dots, i_n$  drawn from a distribution  $F$ . Let  $S = s_1, \dots, s_n$  denote the sequence of steps taken by the algorithm and let  $\Omega = \omega_1, \dots, \omega_n$  be the optimal offline solution for  $I$ . We are interested in bounding the expected loss

$$\mathbf{E}_{I,L}[W(\Omega) - W(S)]$$

between the profits of the optimal offline solution and the online algorithm, where the expectation is taken over both the input distribution and the random choices  $L$  of the algorithm.

**Definition 3 (Expected Loss).** *Let  $I$  be an input sequence,  $\Omega$  be an optimal solution for  $I$ , and  $S$  be the output of an online algorithm  $\mathcal{A}$  on  $I$ . The expected loss of algorithm  $\mathcal{A}$  is defined as  $\mathbf{E}_{I,L}[W(\Omega) - W(S)]$ , where the expectation is taken over both the input distribution and the random choices  $L$  of the algorithm.*

*Notations* For simplicity, in the following, we omit  $I$  and  $L$  when it is clear from the context. In addition,  $I_t$  denotes the inputs revealed during the first  $t$  steps of the run and  $S_t$  denotes the sequence of steps taken by the algorithm in the first  $t$  steps. Given a sequence  $S_i$  for the first  $i$  steps, the optimal sequence of steps following these first  $i$  steps is denoted by

$$\Omega(S_i) = \omega_{i+1}(S_i), \dots, \omega_n(S_i)$$

In particular, the sum

$$W(\Omega(S_{i-1})) = \sum_{t=i}^n w(\omega_t(S_{i-1}))$$

gives the optimal profit of the  $n - i + 1$  steps given the first  $i - 1$  steps  $S_{i-1}$ . Similarly,

$$W(\Omega(S_{i-1} : r)) = \sum_{t=i}^n w(\omega_t(S_{i-1} : r))$$

is the optimal profit of the  $n - i$  steps given the sequence  $S_{i-1} : r$ , i.e., the concatenation of sequence  $S_{i-1}$  with request  $r$ . For simplicity, we will also use  $\omega_i$  as an abbreviation of  $\omega_i(S_{i-1})$ .

*Expected Local Loss* We now define the expected local loss, i.e., the expected loss entailed by choosing a request  $r$  (instead of an optimal request) in step  $i$  of the algorithm.

**Definition 4 (Expected Local Loss).** *The expected local loss of a request  $r$  at step  $i$ , denoted by  $\Delta_i(r)$ , is defined as*

$$\Delta_i(r) = (w(\omega_i) + \mathbf{E}[W(\Omega(S_{i-1} : \omega_i))]) - (w(r) + \mathbf{E}[W(\Omega(S_{i-1} : r))]).$$

Observe that the expected local loss at step  $i$  is computed with respect to the optimal steps following  $S_i$ . Nevertheless, we show that the expected loss is the sum of the expected local losses.

**Lemma 1.** *Let  $I$  be an input sequence,  $\Omega$  be an optimal solution for  $I$ , and  $S$  be the output of an online algorithm  $\mathcal{A}$  on  $I$ . Then,*

$$\mathbf{E}[W(\Omega) - W(S)] = \sum_{i=1}^n \mathbf{E}[\Delta_i(s_i)].$$

*Proof.* Observe first that

$$\mathbf{E}[W(\Omega(S_{i-1}))] = \mathbf{E}[w(\omega_i)] + \mathbf{E}[W(\Omega(S_{i-1} : \omega_i))]$$

where the expectation is taken over the input revealed in step  $i$ . As a consequence, by definition of expected local losses,

$$\mathbf{E}[\Delta_i(s_i)] = \mathbf{E}[W(\Omega(S_{i-1}))] - (\mathbf{E}[w(s_i)] + \mathbf{E}[W(\Omega(S_i))]).$$

and

$$\sum_{i=1}^n \mathbf{E}[\Delta_i(s_i)] = \mathbf{E}[W(\Omega)] - \sum_{i=1}^n \mathbf{E}[w(s_i)] = \mathbf{E}[W(\Omega) - W(S)]. \quad \square$$

*Bounding Expected Local Losses* We now bound the expected local losses for algorithm E. In particular, we show that

$$\mathbf{E}[\Delta_i(s_i)] \leq \sum_{r \in R} \Delta_i(r) e^{-m(\Delta_i(r))^2 / 2\sigma_i^2}$$

where  $m$  is the number of samples taken at each step and  $\sigma_i$  is a bound on the standard deviation of the samples.

**Lemma 2.** *Let  $I$  be an input sequence of length  $n$ ,  $\Omega$  be an optimal solution for  $I$ , and  $S$  be the output of the online algorithm E on  $I$ . The expected local losses in step  $i$  of the algorithm satisfy*

$$\mathbf{E}[\Delta_i(s_i)] \leq \sum_{r \in R} \Delta_i(r) e^{-m(\Delta_i(r))^2 / 2\sigma_i^2}$$

when  $n \rightarrow \infty$ , where  $m$  denotes the number of samples taken at each step and  $\sigma_i$  is a bound on the standard deviation of the samples.

*Proof.* To make the decision at step  $i$ , the algorithm computes, for all  $r \in R$ , an estimate

$$Y(S_{i-1} : r) = \tilde{\mathbf{E}}[W(\Omega(S_{i-1} : r))]$$

of the expectation  $\mathbf{E}[W(\Omega(S_{i-1} : r))]$ . Since the algorithm chose action  $s_i$ , it follows that

$$w(s_i) + Y(S_{i-1} : s_i) \geq w(\omega_i) + Y(S_{i-1} : \omega_i).$$

or, equivalently,

$$w(\omega_i) - w(s_i) \leq Y(S_{i-1} : s_i) - Y(S_{i-1}, \omega_i).$$

By definition of expected local losses, it follows that

$$\Delta_i(s_i) \leq Y(S_{i-1} : s_i) - Y(S_{i-1} : \omega_i) - (\mathbf{E}[W(\Omega(S_{i-1} : s_i))] - \mathbf{E}[W(\Omega(S_{i-1} : \omega_i))]), \quad (1)$$



giving us a necessary condition for request  $s_i$  to be served at step  $i$ . To estimate  $\mathbf{E}[\Delta_i(s_i)]$ , we write

$$\mathbf{E}[\Delta_i(s_i)] = \sum_{r \in R} \Delta_i(r) \Pr(s_i = r),$$

and denote

$$Z_{i,r} = Y(S_{i-1} : s_i) - Y(S_{i-1} : \omega_i) - (\mathbf{E}[W(\Omega(S_{i-1} : s_i))] - \mathbf{E}[W(\Omega(S_{i-1} : \omega_i))]).$$

Since (1) is a necessary condition for an optimal request, it follows that

$$\Pr(r = s_i) \leq \Pr(\Delta_i(r) \leq Z_{i,r}).$$

Now  $Y(S_{i-1} : s_i) - Y(S_{i-1} : \omega_i)$  is the average of  $m$  independent, identically distributed, random variables, each with mean

$$\mathbf{E}[W(\Omega(S_{i-1} : s_i))] - \mathbf{E}[W(\Omega(S_{i-1} : \omega_i))]$$

where  $m$  is the number of samples. Because we have no knowledge about the input distribution, by the central limit theorem,<sup>2</sup> we can argue that

$$\sqrt{m}Z_{i,r}/\sigma_i \sim N(0, 1)$$

where  $\sigma_i$  is a bound on the standard deviation of the sample. Applying a Chernoff bound for the standard normal random variable (see [18][p. 416]), it follows that

$$\Pr(\Delta_i(r) \leq Z_{i,r}) \leq e^{-m(\Delta_i(r))^2/2\sigma_i^2}.$$

and

$$\begin{aligned} \mathbf{E}[\Delta_i(s_i)] &= \sum_{r \in R} \Delta_i(r) \Pr(s_i = r) \\ &\leq \sum_{r \in R} \Delta_i(r) \Pr(\Delta_i(r) \leq Z_{i,r}) \\ &\leq \sum_{r \in R} \Delta_i(r) e^{-m(\Delta_i(r))^2/2\sigma_i^2}. \quad \square \end{aligned}$$

*Bounding Expected Losses* We are now in position to present the main result of this section.

**Theorem 1.** *Let  $I$  be an input sequence of length  $n$ ,  $\Omega$  be an optimal solution for  $I$ , and  $S$  be the output of the online algorithm  $\mathbf{E}$  on  $I$ . The expected loss of Algorithm  $\mathbf{E}$  is bounded by*

$$\mathbf{E}[W(\Omega) - W(S)] \leq \sum_{i=1}^n \sum_{r \in R} \Delta_i(r) e^{-m(\Delta_i(r))^2/2\sigma_i^2}$$

when  $n \rightarrow \infty$ , where  $m$  denotes the number of samples taken at each step and  $\sigma_i$  is a bound on the standard deviation of the samples.

<sup>2</sup> Reference [14] presents an alternative approach not using the central limit theorem. Both approaches lead to the same result.

*Proof.* Direct consequence of Lemmas 1 and 2.  $\square$

This result has some interesting consequences. In particular, assuming that  $\sigma_i$  is  $O(1)$ , the expected loss of algorithm E is  $o(1)$  when the number  $m$  of samples taken at each step is  $\Omega(\log(n|R|))$  inducing  $\Omega(|R| \log(n|R|))$  offline optimizations per step.

**Corollary 1.** *Assume that the standard deviations on the samples are  $O(1)$ . Then, algorithm E, using  $\Omega(\log(n|R|))$  samples per iteration, has an expected loss of  $o(1)$  and performs  $\Omega(|R| \log(n|R|))$  offline optimizations per step.*

Consider now the algorithm R. Denote by  $\text{OPTIMALSOLUTION}(r, R, t)$  the optimal solution for  $R$  and  $t$  when  $r$  is scheduled at time  $t$  (assuming it can) and by  $\text{REGRETSOLUTION}(r, R, t)$  the algorithm approximating  $\text{OPTIMALSOLUTION}(r, R, t)$  using regrets. Assume that algorithm  $\text{REGRETSOLUTION}$  is a  $\rho$ -approximation, i.e.,

$$\rho \times \text{REGRETSOLUTION}(r, R, t) \geq \text{OPTIMALSOLUTION}(r, R, t)$$

for all  $r, R$ , and time  $t$  such that  $r$  can be scheduled at time  $t$ . Under this assumption, algorithm R returns an expected  $\rho(1 + o(1))$ -approximation of the optimal solution using  $\Omega(\log(n|R|))$  offline optimizations per step. Indeed, Theorem 1 also holds when the solutions  $\Omega$  and  $S$  are replaced by their approximations  $\tilde{\Omega}$  and  $\tilde{S}$  that use algorithm  $\text{REGRETSOLUTION}$ . As a consequence,  $\mathbf{E}[W(\tilde{\Omega}) - W(\tilde{S})]$  is  $o(1)$  for  $\Omega(\log(n|R|))$  samples. Since  $\rho W(\tilde{\Omega}) \geq W(\Omega)$ , it follows that

$$\rho(1 + o(1)) W(\tilde{S}) \geq W(\Omega).$$

Similarly, if algorithm  $\text{REGRETSOLUTION}$  is a bounded approximation with bound  $\rho$  of  $\text{OPTIMALSOLUTION}$ , i.e.,

$$\text{REGRETSOLUTION}(r, R, t) + \rho \geq \text{OPTIMALSOLUTION}(r, R, t)$$

then, algorithm R returns a solution whose expected loss is bounded by  $\rho + o(1)$ .

**Corollary 2.** *Assume that algorithm  $\text{REGRETSOLUTION}$  is a  $\rho$ -approximation and that the standard deviations on the samples are  $O(1)$ . Then, algorithm R, using  $\Omega(\log(n|R|))$  samples per iteration, is a  $(\rho(1 + o(1)))$ -approximation of the optimal solution and it performs  $\Omega(\log(n|R|))$  offline optimizations per step. Moreover, if algorithm  $\text{REGRETSOLUTION}$  is an bounded approximation with bound  $\rho$ , algorithm R, using  $\Omega(\log(n|R|))$  samples per iteration, has an expected loss of  $\rho + o(1)$  and it performs  $\Omega(n \log(n|R|))$  offline optimizations per step.*

This result is very important in practice, since it means that algorithm R approximates algorithm E while reducing the number of offline optimizations by a factor  $|R|$ . In general, it is not possible to obtain a similar result for consensus. However, we will come back to this issue in the context of vehicle routing applications, which have special structures.

## 7 Packet Scheduling

This section reports experimental results on the online packet scheduling problem studied in [9]. 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).

### 7.1 The Offline Problem

We are given a set  $J$  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  $H = [\underline{H}, \overline{H}]$  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  $\gamma : H \rightarrow J$  which assigns a job to each time in the schedule horizon. A schedule  $\gamma$  is feasible if

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

The weight of a schedule  $\gamma$ , denoted by  $w(\gamma)$ , is given by  $w(\gamma) = \sum_{t \in H} w(\gamma(t))$ . The goal is to find a feasible schedule  $\gamma$  maximizing  $w(\gamma)$ . This offline problem can be solved in quadratic time  $O(|J||H|)$  [9].

### 7.2 The Online Problem

The experimental evaluation is based on the problems of [9, 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.

### 7.3 The Regret Function

We now specify the regret function which consists of swapping a constant number of packets in the optimal schedule and is based on a simple case analysis. Consider a job  $r \in \text{READY}(R, t)$ .

If job  $r$  is not scheduled (i.e.,  $r \notin \gamma$ ), the key idea is to try rescheduling  $\gamma(t)$  instead of the job of smallest weight in  $\gamma$ . The regret becomes

$$\min(s \in [t, a(\gamma(t)) + d]) w(\gamma(s)) - w(r),$$

since the replaced job is removed from  $\gamma$  and  $r$  is added to the schedule. In the worst case, the replaced job is  $\gamma(t)$  and the regret is  $w(\gamma(t)) - w(r)$ .

If job  $r$  is scheduled at time  $t_r$ , the regret function first tries to swap  $r$  and  $\gamma(t)$  in which case the regret is 0. If this is not possible, the function tries rescheduling  $\gamma(t)$  instead of the job of smallest weight in  $\gamma$ . If  $\gamma(t)$  cannot be rescheduled, the regret function simply selects the best possible unscheduled job which may be scheduled at  $t_r$  and the regret now becomes

$$w(\gamma(t)) - \max(u \in U_r) w(u)$$

where

$$U_r = \{j \in \text{READY}(R, t_r) \mid j \notin \gamma\},$$

since job  $\gamma(t)$  is lost in the schedule. If  $\gamma(t)$  is rescheduled at time  $s$ , then the regret concludes by selecting the best possible unscheduled job which may be scheduled at  $t_r$  and the regret now becomes

$$w(\gamma(s)) - \max(u \in U_{r,s}) w(u)$$

where

$$U_{r,s} = \{j \in \text{READY}(R, t_r) \mid j \notin \gamma \vee j = \gamma(s)\}.$$

This regret function takes  $O(\max(d, |C|))$  time, which is sublinear in  $|J|$  and  $|H|$  and essentially negligible for this application. We now prove that it provides a 2-approximation.

**Theorem 2.** *The regret function for packet scheduling is a 2-approximation.*

*Proof.* Let  $R$  be the set of requests at time  $t$  and let  $r \in R$  be a request that can be scheduled at time  $t$ . Let  $\gamma^*$  be an optimal solution, i.e.,  $\gamma^* = \text{OPTIMALSOLUTION}(R, t)$ , let  $\gamma_r$  be an optimal solution when  $r$  is scheduled at time  $t$ , i.e.,  $\gamma_r = \text{OPTIMALSOLUTION}(r, R, t)$ , and let  $\tilde{\gamma}_r$  be the solution obtained by the regret function. We show that

$$\frac{w(\gamma_r)}{w(\tilde{\gamma}_r)} \leq 2.$$

Most of the proof consists of showing that, for each lost packet  $l$ , there is another packet in  $\gamma^*$  whose weight is at least  $w(l)$  giving us a 2-approximation since  $w(\gamma_r) \leq w(\gamma^*)$ .

First observe that the result holds when  $w(x) \leq w(r)$  since, in the worst case, the regret function only loses packet  $x$ . So we restrict attention to  $w(x) \geq w(r)$ . If  $x \in \tilde{\gamma}_r$ , i.e., if the regret function swaps  $x$  with another packet  $y$  (case 1), the result also holds since  $w(y) \leq w(x)$ . If  $x \notin \tilde{\gamma}_r$  and  $x$  can be scheduled after time  $t$ , it means that there exists a packet  $y$  at each of these times satisfying  $w(y) \geq w(x)$  and the result holds. It thus remains to consider the case where  $x$  can only be scheduled at time  $t$  and is thus lost in  $\gamma_r$ . If  $r \notin \gamma^*$ , the regret function is optimal, since otherwise  $r$  would be in the optimal schedule after time  $t$ . Otherwise, it is necessary to reason about a collection of packets. Indeed,  $w(\gamma^*) = w(x) + w(r) + w(S)$ , where  $S = \{p \in \gamma^* \mid p \neq x \ \& \ p \neq r\}$ . We also know that  $w(\tilde{\gamma}_r) \geq w(r) + w(S)$  since, in the worst case, the regret function loses packet  $x$ . Finally,  $w(\gamma_r) = w(r) + w(Z)$ , where  $Z$  are the packets scheduled after time  $t$ . Since  $\gamma^*$  is optimal, it follows that  $w(Z) \leq w(r) + w(S)$  and the result follows.  $\square$

## 7.4 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 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-optimizing.

The results also indicate that consensus outperforms E whenever few optimizations are available (e.g.,  $\leq 15$ ). The improvement is particularly significant when there are very few available optimizations. Consensus is dominated by E when the number of available optimizations increases, although

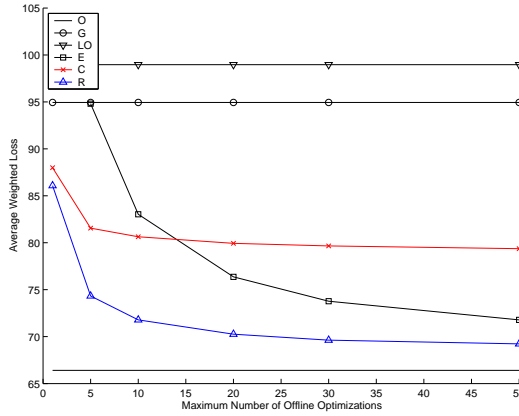


Fig. 2. The Regret Algorithm on Packet Scheduling

it still produces significant improvements over the oblivious algorithms. This is of course pertinent, since E is not practical for many problems with time constraints. 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 even when there are a reasonably large number of them, (weak time constraints).

Finally, it is interesting to observe that algorithm R with 10 offline optimizations produces the same solution quality as algorithm E with 50 iterations. Since the number of ready requests at each time  $t$  is about 5 in average, the experimental results nicely agree with the theoretical analysis.

## 8 The Online Stochastic Algorithm Revisited

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

### 8.1 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 be generalized to provide these functionalities. The key idea is to maintain a set of sample 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  $\Gamma$  is initialized in Line 2. The request is selected in line 5 by function CHOOSEREQUEST which now receives  $\Gamma$  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 abstracted away. The decision code is what is

```

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

GENERATESOLUTION( $R, t$ )
1  $\Gamma \leftarrow \emptyset$ ;
2 repeat
3    $S \leftarrow R \cup \text{GETSAMPLE}([t + 1, t + \Delta])$ ;
4    $\gamma \leftarrow \text{OPTIMALSOLUTION}(S, t)$ ;
5    $\Gamma \leftarrow \Gamma \cup \{\gamma\}$ ;
6 until until time  $t + 1$ 
7 return  $\Gamma$ ;

CHOOSEREQUEST-C( $R, t, \Gamma$ )
1 for  $r \in R$ 
2 do  $f(r) \leftarrow 0$ ;
3 for  $\gamma \in \Gamma$ 
4 do  $f(r) \leftarrow f(r) + w(\gamma)$ ;
5 return  $\text{argmax}(r \in R) f(r)$ ;

```

**Fig. 3.** The Generic Online Algorithm with Precomputation

left in the instantiations of function CHOOSEREQUEST. The figure also gives the implementation of CHOOSEREQUEST for algorithm C to illustrate the instantiations.

## 8.2 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 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  $\Gamma$  and function REMOVEINFEASIBLESOLUTIONS removes those solutions which cannot accommodate the new requests.

## 8.3 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

```

ONLINEOPTIMIZATION( $H, R$ )
1  $w \leftarrow 0$ ;
2  $\Gamma \leftarrow \text{GENERATESOLUTIONS}(R, 0)$ ;
3 for  $t \in H$ 
4 do  $N \leftarrow \text{ACCEPTREQUESTS}(R, t, \Gamma)$ ;
5    $\Gamma \leftarrow \text{REMOVEINFEASIBLESOLUTIONS}(R, t, N, \Gamma)$ ;
6    $R \leftarrow \text{AVAILABLEREQUESTS}(R, t) \cup N$ ;
7    $r \leftarrow \text{CHOOSEREQUEST}(R, t, \Gamma)$ ;
8    $\text{SERVEREQUEST}(r, t)$ ;
9    $w \leftarrow w + w(r)$ ;
10   $R \leftarrow R \setminus \{r\}$ ;
11   $\Gamma \leftarrow \{\gamma \in \Gamma \mid \gamma(t) = r\}$ ;
12   $\Gamma \leftarrow \Gamma \cup \text{GENERATESOLUTIONS}(R, t)$ ;

```

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

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  $\text{CHOOSEREQUEST}(R \cup \{\perp\}, t, \Gamma)$  in line 5 of the algorithm.

#### 8.4 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  $\gamma$  at time  $t$  returns a tuple  $\gamma(t) = (r_1, \dots, r_n) = (\gamma_1(t), \dots, \gamma_n(t))$ . It suffices to replace  $r$  in the online algorithm by a tuple  $(r_1, \dots, r_n)$  to obtain a generic algorithm over tuples of decisions. However, it is important to reconsider how to choose requests in this new context. A straightforward generalization of consensus would give

```

CHOOSEREQUEST-C( $R, t$ )
1 for  $e \in R^n$ 
2   do  $f(e) \leftarrow 0$ ;
3 for  $i \leftarrow 1 \dots \mathcal{O}$ 
4   do  $S \leftarrow R \cup \text{GETSAMPLE}([t + 1, t + \Delta])$ ;
5      $\gamma \leftarrow \text{OPTIMALSOLUTION}(S, t)$ ;
6      $f(\gamma(t)) \leftarrow f(\gamma(t)) + W(\gamma)$ ;
7 return  $\text{argmax}(e \in R^n) f(e)$ ;

```

Unfortunately, this generalized implementation of consensus is not particularly effective, especially when there are many requests and few samples. 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 across all samples and by selecting the best coupling available among the solutions. This *pointwise consensus* can be formalized as follows:

```

CHOOSEREQUEST-PC( $R, t$ )

```

```

1 for  $r \in R, i \in 1..n$ 
2   do  $f_i(r) \leftarrow 0$ ;
3 for  $i \leftarrow 1 \dots \mathcal{O}$ 
4   do  $S \leftarrow R \cup \text{GETSAMPLE}([t + 1, t + \Delta])$ ;
5      $\gamma \leftarrow \text{OPTIMALSOLUTION}(S, t)$ ;
6     for  $i \in 1..n$ 
7       do  $f_i(\gamma_i(t)) \leftarrow f_i(\gamma_i(t)) + W(\gamma)$ ;
8    $\gamma^* = \text{argmax}(\gamma \in F) \sum_{i=1}^n f_i(\gamma_i(t))$ ;
9 return  $\gamma^*(t)$ ;

```

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

## 9 Vehicle Routing

This section describes the applications of the online generic algorithm with precomputation, service guarantees, pointwise consensus, 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.

### 9.1 The Problem

The application is based on the model proposed in [16] where customers are distributed in a  $20\text{km} \times 20\text{km}$  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 [16]. We use  $n$  to denote the expected number of customers and  $H$  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  $\{0\%, 5\%, \dots, 100\%\}$ . For a DOD  $x$ , there are  $n(1 - x)$  known customers. The remaining customers are generated using an exponential distribution with parameter  $\lambda = \frac{nx}{H}$  for their inter-arrival times. It follows from the corresponding Poisson distribution (with parameter  $\lambda H$ ) 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 and 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.

### 9.2 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) [19, 1]. These initial



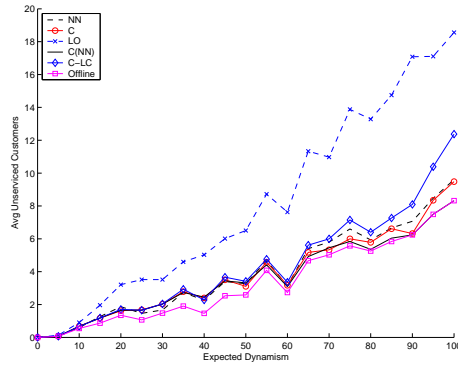


Fig. 5. Results on the Number of Served Customers

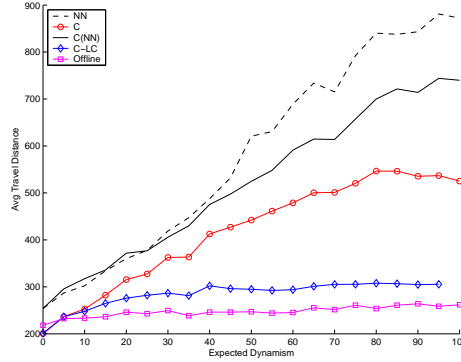


Fig. 6. Results on Travel Distance

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 experimentally 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 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. Note that the experimental results do not discuss the regret algorithm, since the quality of pointwise consensus alone is largely sufficient for these applications.

### 9.3 Experimental Results

The online generic algorithm is compared with the Nearest Neighbor (NN) heuristic proposed in [16] 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 more sophisticated optimization algorithms on fewer samples than a weaker method on more samples.

### 9.4 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.

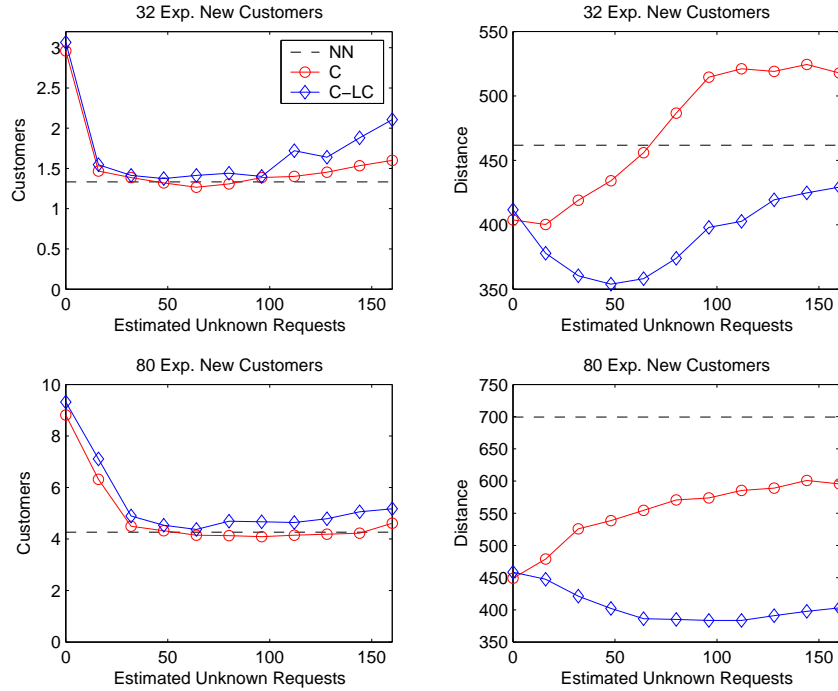


Fig. 7. Robustness Results

### 9.5 Visualizing the Algorithms

This section presents a visualization of the algorithms over time. The goal is to explain the experimental results intuitively in terms of the actual decisions taken by the algorithms and to provide insights on the differences in behavior and solution quality.

The visualizations only consider one run of the algorithm (with 50% DOD), although other runs typically exhibit similar behaviors. They report three snapshots for each algorithm, which depict the routes visited after 1 hour, 4 hours, and 8 hours respectively. Each snapshot shows the four vehicles, one in each quadrant. The customers that are known and accepted at the time of the snapshot are shown in yellow and those who are rejected by the algorithm are shown in red. All accepted and rejected customers are shown in all quadrants, since it is not clear which vehicles will actually serve them. Note that the right side of each snapshot will provide some interesting information. It depicts the expected number of customers, the degree of dynamism, the number of plans available at this stage, the number of unserved and rejected customers, and the travel costs. The available plans show the projected travel cost as well. Finally, the arcs in yellow show the current solution used by the algorithms to make decisions which, of course, that solution evolves over time.

*Algorithm NN* Figures 8, 9, and 10 visualize algorithm NN. After 1 hour, algorithm NN has travelled 133.8km and expects to travel 232km. It still has 42 unserved customers and has not rejected any request. After 4 hours, algorithm NN has travelled 382.2km and has visited all the known customers.

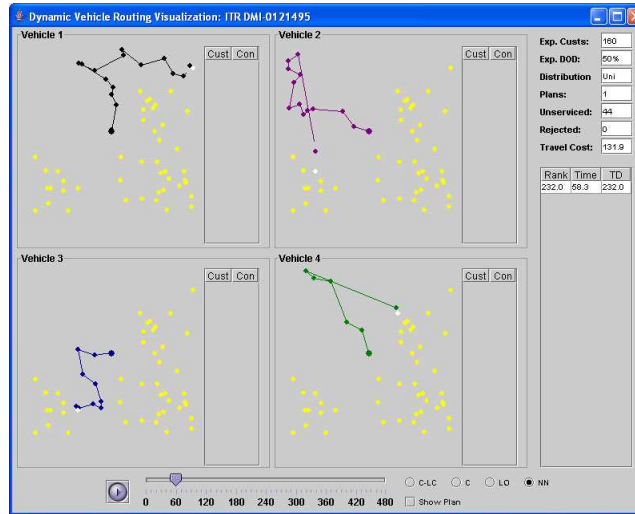


Fig. 8. Algorithm NN after 1 Hour

After 8 hours, algorithm NN has rejected 3 customers and travelled 580.7km. An interesting observation about algorithm NN is that it becomes essentially a first-come/first-serve algorithm after 4 hours, since all known customers have been served at that point. We will come back to this observation later on when we discuss consensus. Finally, note the travel patterns of the vehicles. Each of which visits a significant region of the space and features many crossings.

*Algorithm LO* Figures 11, 12, and 13 visualize algorithm LO, whose behaviour is particularly interesting. After 1 hour, algorithm LO has travelled 63.5km and expects to travel 182km in its best solution.<sup>3</sup> It still has 67 unserved customers and has not rejected any request. In other words, it has travelled less than algorithm NN and anticipates a smaller total travel time. Moreover, it has only deployed two vehicles at this stage, since using fewer vehicles typically mean smaller travel times in these problems. After 4 hours, algorithm LO has travelled 214.1km, deployed three vehicles, and still has 26 customers to serve for an anticipated travel distance of 255.5km. After 8 hours, algorithm LO has rejected 6 customers, travelled 423.1km, and finally deployed its last vehicle to serve only one customer. As the experimental results showed, algorithm LO is not competitive with NN as far as customer service is concerned. The main reason is now apparent: algorithm LO over-optimizes travel distance and leaves little room to accommodate new requests at the end of the routing. In particular, it deploys its vehicles too late, believing that it can serve the existing customers with fewer of them. As a consequence, they are not well positioned to accommodate new requests.

*Algorithm C* Figures 8, 9, and 10 visualize algorithm C. After 1 hour, algorithm C has travelled 121.9km and expects to travel 273.3km. It still has 47 unserved customers and has not rejected any request. After 4 hours, algorithm C has travelled 348.2km and has visited all the known customers

<sup>3</sup> Algorithm LO generates as many solutions as possible in the allowed time. It generalizes the seminal work in [11].

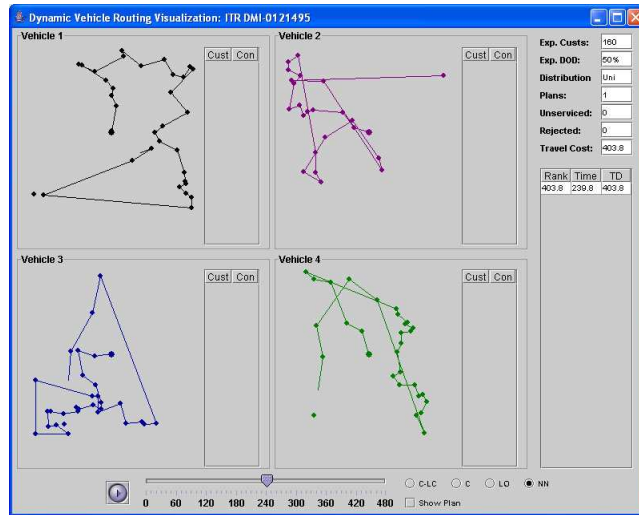


Fig. 9. Algorithm NN after 4 Hours

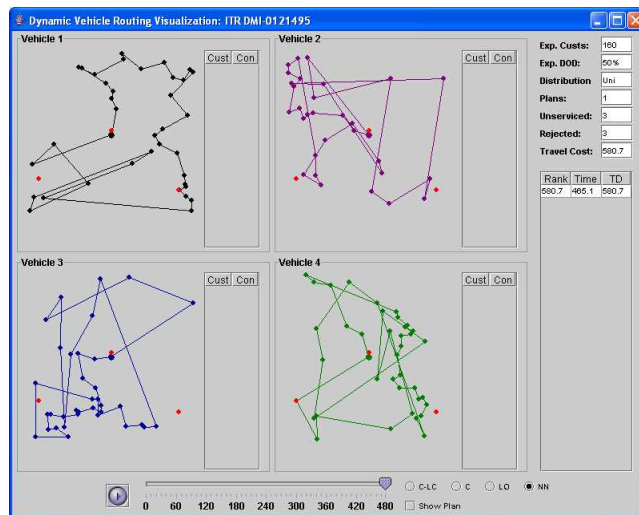


Fig. 10. Algorithm NN after 8 Hours

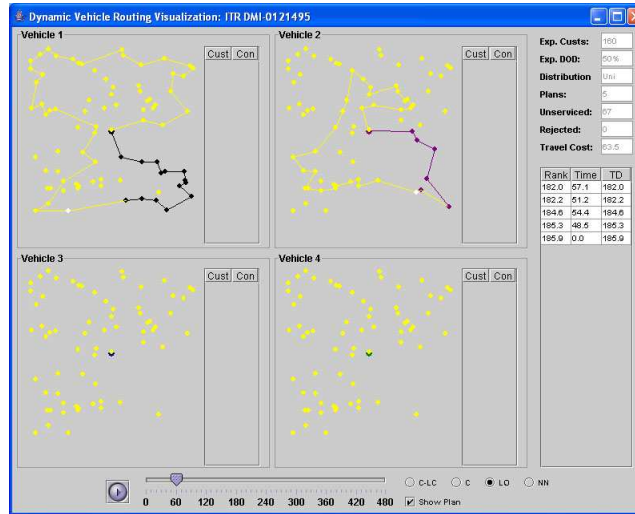


Fig. 11. The LO Algorithm after 1 Hour

but one. After 8 hours, algorithm C has rejected 3 customers and travelled 518.2km. Interestingly, algorithm C also becomes a greedy algorithm after 4 hours, since all known customers have been served at that point. This indicates that algorithm C has been too eager to serve the existing requests and is not able to amortize some of their travel with future requests. This behavior clearly illustrates the need to generalize traditional offline algorithm to accommodate some of the structures present in online algorithms. Observe also how the “nice” travel patterns after 4 hours deteriorate in the second half of the day, travelling to customers that are relatively remote and exhibiting some significant crossings.

*Algorithm C-LC* Figures 17, 18, and 19 visualize algorithm C-LC. After 1 hour, algorithm C-LC has only travelled 21.6km and expects to travel around 237km. It still has 83 unserved customers and has not rejected any request. Observe how the vehicles are slowly deployed and do not rush to serve customers since they expect to have plenty of time to serve them. After 4 hours, algorithm C-LC has travelled 150.4km, has 62 unserved customers, and anticipate a travel distance around 283km. Recall that algorithm C has served all known customers at this point (but one). After 8 hours, algorithm C-LC has rejected 3 customers and travelled only 363.9km. Observe the nice patterns of the vehicles and the relatively small number of crossings.

## 10 Vehicle Routing with Time Windows

We now evaluate various algorithms on online multiple vehicle routing with time windows. This problem was studied initially in [2] to show the value of stochastic information in vehicle routing. It is particularly interesting because the feasibility constraints are much stronger than in the previous application. The challenge then is not on reducing travel distances, but rather to serve as many customers as possible.

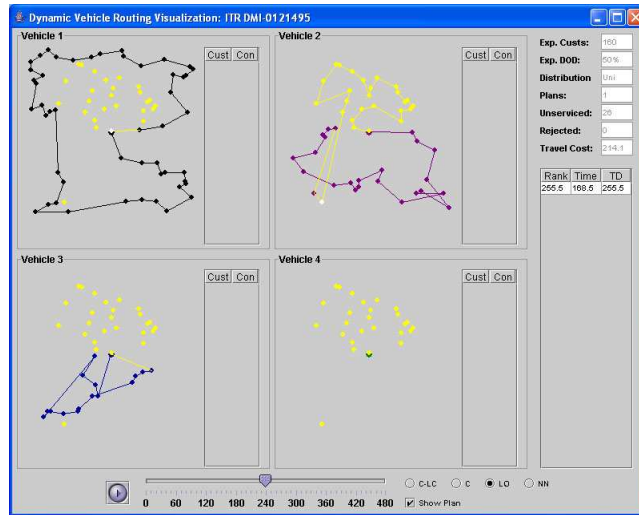


Fig. 12. The LO Algorithm after 4 Hours

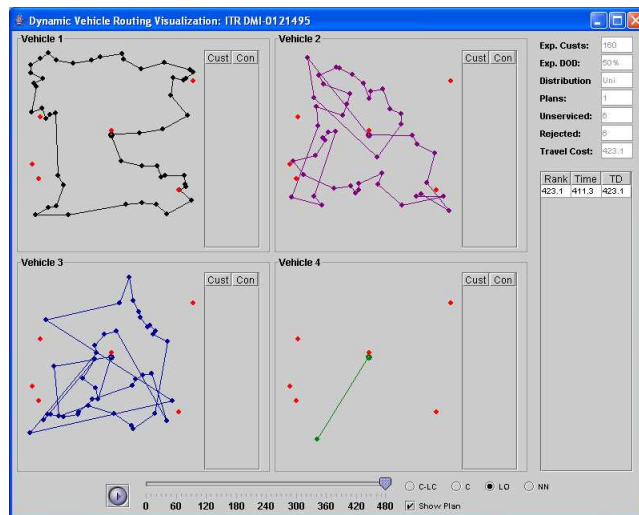


Fig. 13. The LO Algorithm after 8 Hours

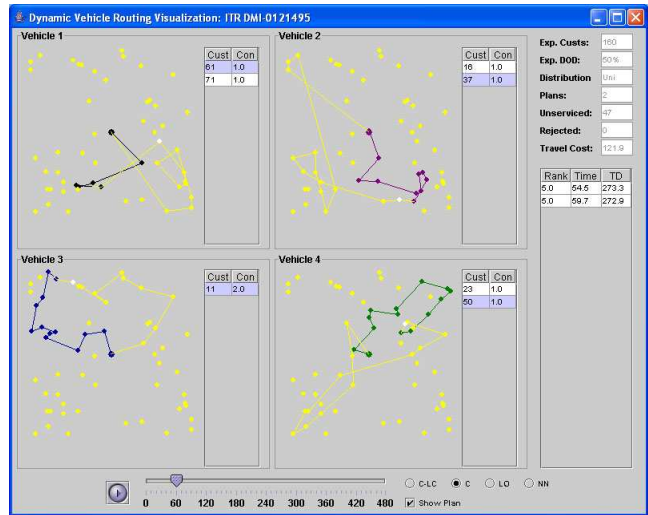


Fig. 14. The C Algorithm after 1 Hour

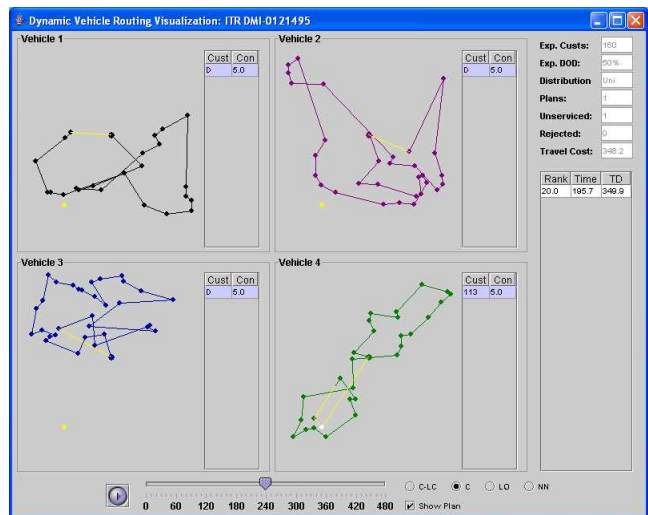


Fig. 15. The C Algorithm after 4 Hours



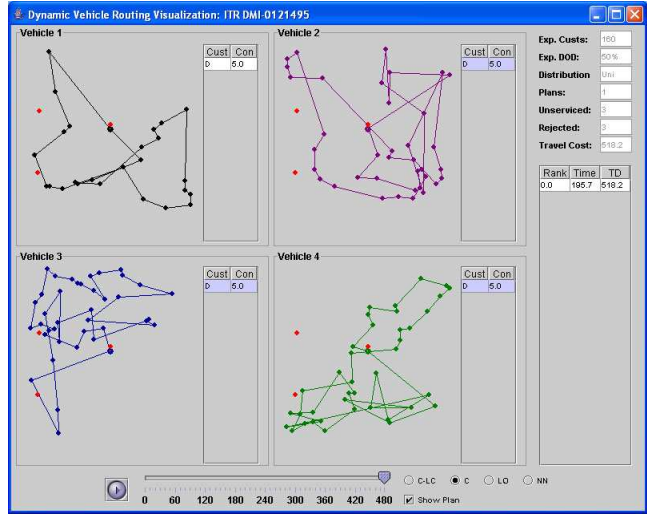


Fig. 16. The C Algorithm after 8 Hours

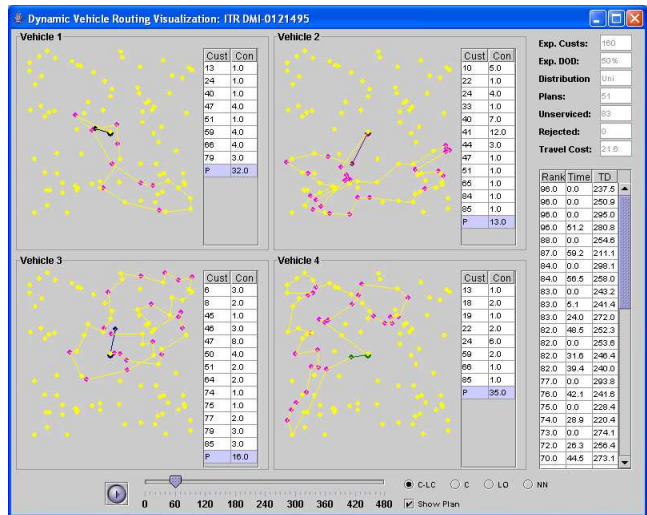


Fig. 17. The C-LC Algorithm after 1 Hour

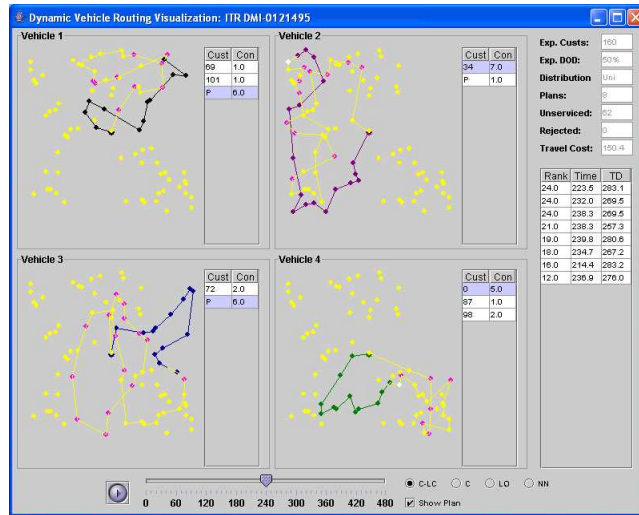


Fig. 18. The C-LC Algorithm after 4 Hours

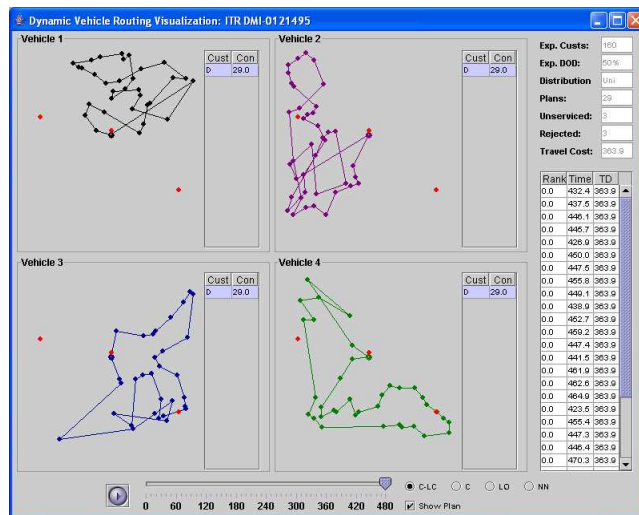


Fig. 19. The C-LC Algorithm after 8 Hours

### 10.1 Problem Formulation

These vehicle routing problems are specified formally in [2] where all the details can be found. Each problem contains a depot, a number of customer regions and a number of customer service 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 or, equivalently, minimizing the number of rejected customers. In the online version, 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.

Note that the VRPTW is a hard NP-complete problem whose instances are extremely difficult to solve optimally. Only 2 to 10 offline optimizations can be solved in between two events and the number of events is large (e.g., 50 different requests). Hence, algorithm E is not practical at all, since it would not even be able to evaluate each request on a simple sample.

### 10.2 Experimental Setting

The experimental results are based on the class-4 problems from [2], where all details can be found. They are derived from the Solomon benchmarks which are very challenging and involve 100 customers. The 15 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 online algorithms was determined by solving the offline problems and adding two vehicles.

### 10.3 The Regret Function

The regret function is simple and fast. Consider the decision of choosing which customer to serve next on vehicle  $v$  and let  $s$  be the first customer on the route of vehicle  $v$ . To evaluate the regret of another customer  $r$  on a vehicle  $v$ , the key idea is to determine if there is a feasible swap of  $r$  and  $s$  on  $v$ , in which case the regret is zero. Otherwise, if such a swap violates the time window constraints, the regret is 1. *The main benefit of this regret function is to recognize that some choices of customers are essentially equivalent.*

The regret function is a 2-approximation, since it loses at most one additional customer. Moreover, when the objective function is viewed as minimizing the number of rejected customers, the regret function provided a bounded approximation with bound 1. Interestingly, on this application, consensus is also a bounded approximation, since it implicitly assumes the systematical rejection of one additional customer. Note that this is also the case for the first component of the optimization function in the application described in the previous section.

Problem	DOD	Vehicles	LO	C	R
20-20-60-rc101-1	46.3%	16	3.3	3	3.48
20-20-60-rc101-2	45.8%	15	5.84	4.32	4.84
20-20-60-rc101-3	50.0%	16	3.02	3.24	3.46
20-20-60-rc101-4	45.6%	17	6.96	5.08	5.32
20-20-60-rc101-5	47.4%	16	6.2	6.08	5.72
20-20-60-rc102-1	59.0%	15	2.12	1.1	1.94
20-20-60-rc102-2	57.5%	15	7.06	3.66	3.7
20-20-60-rc102-3	56.0%	15	6.52	4.12	3.6
20-20-60-rc102-4	52.0%	14	2.76	2.58	3.12
20-20-60-rc102-5	57.6%	15	5.08	2.88	2.9
20-20-60-rc104-1	76.1%	13	22.4	13.38	9.68
20-20-60-rc104-2	75.6%	14	25.58	13.86	12.16
20-20-60-rc104-3	76.1%	13	19.3	10.64	8.98
20-20-60-rc104-4	72.2%	12	21.16	14.32	9.42
20-20-60-rc104-5	74.4%	11	17.18	13.38	10.2

**Table 1.** Regret on Online Vehicle Routing with Time Windows

## 10.4 Experimental Results

Table 1 depicts the results on the 15 instances of the Solomon benchmarks. Each instance is solved 50 times because of the nondeterministic nature of the sampling and LNS algorithms. The second column gives the degree of dynamism and the third column gives the number of vehicles. The last columns specify the number of missed customer by algorithms LO, C, and R. First observe that the regret algorithm produces significant benefits over LO, especially on the problems where the degree of dynamism is high (about 70%) inducing stricter time constraints. On these highly dynamic problems, R may reduce the number of missed customers by 225% and always produces a reduction of at least 69%. regret algorithm does not bring any benefit over consensus for the first two classes of problems with lower degrees of dynamism. However, it produces some dramatic improvements on the highly dynamic instances. On these problems, the regret algorithm reduces the number of missed customers by up to 52% and always produces reductions above 18%. This is a very interesting result, since consensus is particularly effective on these problems and also provides a bounded approximation. However, by recognizing “equivalent” choices, the regret algorithm further improves the approximation and produces significant benefits for the most time-constrained instances.

## 11 Related Work

Online algorithms (e.g., [10]) have been addressed for numerous years but research has traditionally focused on techniques oblivious to the future and on competitive ratios [13]. It is only recently that researchers have begun to study how information about future uncertainty may improve the performance of algorithms. This includes scheduling problems [9], vehicle routing problems [2, 8] and elevator dispatching [17] to name a few. Research on these problems has varied widely, but the unifying theme is that probabilistic information about the future significantly increases quality. The expectation method was the primary method used in [9], They also pointed out why POMDPs are too general for this

class of problems. The consensus approach was motivated by online stochastic vehicle routing [2] and applied to online packet scheduling in [4]. The regret approach was derived from our desire to obtain theoretical results on the solution quality [5].

## 12 Conclusion

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 proposes a generic framework for online stochastic optimization and several of its instantiations, including algorithm E, C, and R. The theoretical results indicate that, under reasonable and practical assumptions, the expected quality loss of algorithm E is  $o(1)$  for a total of  $n|R|\Omega(\log(n|R|))$  offline optimizations, while algorithm R is a  $\rho(1 + o(1))$ -approximation whenever its underlying regret function is a  $\rho$ -approximation and requires a total of  $n\Omega(\log(n|R|))$  offline optimizations. The experimental results, on packet scheduling and multiple vehicle routing without time windows, confirm the theoretical results. They clearly show the benefits of stochastic information and the practical value of algorithms C and R under severe time constraints.

## Acknowledgments

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. *Transportation Science*, 38(4), 515-530, 2004.
2. R. Bent and P. Van Hentenryck. 2001. Scenario Based Planning for Partially Dynamic Vehicle Routing Problems with Stochastic Customers. *Operations Research*, 52(6), 977-987, 2004.
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. R. Bent and P. Van Hentenryck. 2004. Online Stochastic and Robust Optimization. In *Proceedings of the Ninth Asian Computing Science Conference (ASIAN'04)*, Chiang Mai University, Thailand.
7. J. Birge and F. Louveaux. 1997. Introduction to Stochastic Programming. Springer Verlag.
8. A. Cambell, and M. Savelsbergh. 2002. Decision Support for Consumer Direct Grocery Initiatives. *Report TLI-02-09, Georgia Institute of Technology*.
9. H. Chang, R. Givan, and E. Chong. 2000. On-line Scheduling Via Sampling. In *AIPS'2000*, 62-71.
10. Fiat, A., and Woeginger, G. *Online Algorithms: The State of the Art*. Springer Verlag, 1998.
11. M. Gendreau and F. Guertin and J. Y. Potvin and E. Taillard. Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching. *Transportation Science*, 33(4), 381-390, 1999.
12. L. Kaelbling, M. Littman, and A. Cassandra. Planning and Acting in Partially Observable Stochastic Domain. *Artificial Intelligence*, 101(1-2), 99-124, 1998.

13. Karlin, A.; Manasse, M.; Rudolph, L.; and Sleator, D. 1988. Competitive Snoopy Caching. *Algorithmica* 3:79–119.
14. A.J. Kleywegt, A. Shapiro, and T. Homer-De-Mello. The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM j. on Optimization*, 12:479–502, 2001.
15. P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Kluwer Academic Publishers, 1997.
16. A. Larsen, O. Madsen, and M. Solomon. Partially Dynamic Vehicle Routing-Models and Algorithms. *Journal of Operational Research Society*, 53:637–646, 2002.
17. Nikovski, D., and Branch, M. 2003. Marginalizing Out Future Passengers in Group Elevator Control. In *UAI'03*.
18. S. Ross. *A First Course in Probability*. Fifth Edition. Prentice Hall, New Jersey, 1997.
19. P. Shaw. 1998. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *CP'98*, 417–431.