RUTCOR
RESEARCH
REPORT

Optimal Information Monitoring under a Politeness Constraint

Jonathan Eckstein[a] Avigdor Gal[b] Sarit Reiner[c]

RRR 16-2005, May, 2005

RUTCOR
Rutgers Center for
Operations Research
Rutgers University
640 Bartholomew Road
Piscataway, New Jersey
08854-8003
Telephone: 732-445-3804
Telefax: 732-445-5472
Email: rrr@rutcor.rutgers.edu
http://rutcor.rutgers.edu/∼rrr

[a]Department of Management Science and Information Systems and RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854 USA. E-mail: jeckstei@rutcor.rutgers.edu

[b]Faculty of Industrial Engineering and Management, Technion - Israel Institute of Technology, Technion City, Haifa 32000, Israel. E-mail: avigal@ie.technion.ac.il

[c]Faculty of Industrial Engineering and Management, Technion - Israel Institute of Technology, Technion City, Haifa 32000, Israel. E-mail: reiner@techunix.technion.ac.il

Rutcor Research Report
RRR 16-2005, May, 2005

Optimal Information Monitoring under a Politeness Constraint

Jonathan Eckstein Avigdor Gal Sarit Reiner

Abstract. We describe scheduling algorithms for monitoring an information source whose contents change at times modeled by a nonhomogeneous Poisson process. In a given time period of length T, we enforce a *politeness* constraint that we may only probe the source at most n times. This constraint, along with an optional constraint that no two probes may be spaced less than δ time units apart, is intended to prevent the monitor from being classified as a nuisance to be "locked out" of the information source.

To develop our algorithms, we use a portion of the cost model developed in our earlier work [8]. Our first algorithm assumes a discrete set of $N > n$ possible update times, and uses dynamic programming to identify a provably optimal subset of n of these times at which to probe the server. Our second algorithm is a simple direct search procedure for locally improving any continuous-time schedule with respect to the same cost model. In particular, this improvment procedure may be applied to the schedule obtained from our first algorithm. We demonstrate the effectiveness of the algorithms by comparing them with previously-proposed methods, using real-world data feeds and varying preference parameters.

1 Introduction

Consider a situation in which a client maintains a (possibly partial) "mirror" or copy of an information resource maintained by a server. We concentrate on "pull" environments in which it is the client's responsibility to monitor the server for information changes, as opposed to "push" environments in which the server is responsible for alerting all clients about alterations to the resource.

Applications calling for such periodic monitoring and transcription include certain kinds of data warehouses, pervasive systems (*e.g.*, Microsoft's Mobile Information Server[1] and Café Central[2]), stream data monitors, Web cache managers, and Web crawlers. The quality of data monitoring in such settings is measured by the amount of captured information, termed *completeness* [13], and the degree of delay in delivering updates, which has been variously termed *timeliness* [13], *obsolescence* [8], or *age* [5]. For example, when downloading email data from a server, one wishes to monitor the arrival of new email messages to avoid loss of data (due to quota restrictions) and to ensure the timely receipt of messages.

It has been argued, for example in [13], that monitoring of Web information sources must be pull-based rather than push-based, either due to lack of server cooperation (as in most Web-based applications) or the expense of maintaining large numbers of client profiles.[3] Furthermore, updates at the server will typically occur in some kind of random manner, so clients cannot know when an update has occurred without probing the server.

When using pull-based policies, one must either constrain the number of probes, or attach some kind of client cost to each probe. Otherwise, the client will probe continually, wasting network resources, and possibly being shut down as a presumed network attacker. For instance, Pandey *et al.* [13] assume a constraint of some maximum number of probes per unit time, using a discrete-time model. In our own prior work [8], we assigned a cost to each probe and proposed policies aimed at minimizing the combined cost of probing and obsolescence. However, these policies did not fully exploit the structure of [8]'s obsolescence cost model.

This paper is essentially a follow-up article to the numerical experiments of [8], proposing more rigorous, higher-performance scheduling algorithms for the same cost model. We introduce probe-scheduling algorithms that operate under the simple *politeness* constraint that over the entire planning period $[0, T]$, we may probe the server at most n times. This kind of constraint is practiced by Websites such as the British Telecom telephone directory,[4] and can be used in many applications, such as Web cache collaboration and peer-to-peer communication. In addition, we introduce an optional second constraint that no two probes can be spaced more closely than $\delta \geq 0$ time units apart. Under these constraints, probes should be scheduled to minimize obsolescence.

[1] http://www.microsoft.com/servers/miserver/

[2] http://www.comalex.com/central.htm

[3] Services such as Research In Motion can support up to 1 Million mobile users profiles, albeit limited in their scope.

[4] http://www.bt.com

Our cost model is essentially the "insertion" model of [8]. We review this model here for expository completeness, and then analyze its properties in further detail. The model takes into account two different time-varying factors: *update intensity* and *user importance*. By update intensity, we mean natural time-of-day and day-of-week variation in the frequency of changes to the information source. For example, a Website may tend to be updated primarily during working hours, or mainly on weekday evenings. We model this kind of variation via a nonhomogeneous Poisson process with periodic arrival intensity function $\lambda : [0, T] \to [0, \infty)$, where $[0, T]$ is the period over which we would like to construct our probing schedule. Algorithms that take advantage of update intensity patterns are likely to have better average performance than simple periodic polling algorithms, as proposed for example in [12].

"User importance" is meant to reflect that up-to-date information has greater value to the consumer at some times than at others. For example, someone may assign higher importance to receiving new e-mails during work hours than after hours. Similarly, timely monitoring of an online auction may become increasingly important towards the end of the bidding period. To model a diverse variety of such situations, we define an *importance function* $a : [0, T] \to [0, \infty)$; in this paper's computational experiments, for instance, we set

$$a(t) = \begin{cases} a_1, & \text{if } t \text{ is during work hours} \\ a_2, & \text{otherwise,} \end{cases} \tag{1}$$

where $a_1 > a_2$. However, our only restrictions on $a(\cdot)$ are that it be integrable and bounded above on $[0, T]$.

Section 2 of this paper reviews [8]'s cost model, which is parameterized by $\lambda(\cdot)$ and $a(\cdot)$, and formally introduces the politeness-constrained monitoring problem. We then develop two principal optimization algorithms for this model. For the first algorithm, developed in Section 3, we suppose that updates may only be performed at a discrete set of times $\mathcal{T}$ of cardinality $N > n$. Using dynamic programming, we calculate an optimal solution to this additionally-constrained version of the model in polynomial time. We also prove an $O(1/N)$ bound on how much the discretized problem's optimal cost can differ from the original problem's.

Our second algorithm, presented in Section 4, dispenses with the discrete-time constraint, but does not calculate a provable global optimum. Instead, it is a specialized, heuristic direct search procedure, using coordinate descent moves to iteratively improve a given solution. We also experimented with classical nonlinear programming methods for the same purpose, but the peculiar features of our cost function prevented them from being useful.

We claim that very good solutions can be obtained to the probe scheduling problem by applying our first algorithm on a sufficiently fine-grained time grid, and then subjecting the outcome to further improvement via our second algorithm. In support of this hypothesis, we present experiments using several real-world data feeds.

Section 6 briefly describes some related prior work in the information systems literature, and Section 7 offers conclusions and possibilities for future research.

We note that our general approach should be useful not only for the particular cost

model we develop, but for any pull-based monitoring system whose obsolescence cost model has a certain straightforward additive structure reflected in equation (6) below. Only one subroutine in our method, Algorithm 4 in Section 3, is strongly dependent on the particulars of our cost model. For a different application, one could simply replace this subroutine with a different one. In this case, however, the $O(1/n)$ error bound might fail to hold, or might require a modified analysis.

2 Update and cost models

Consider the time period $[0, T]$, and let $\{p_i\}_{i=1}^n$ represent the times the client probes the server. At probe i, the client data are synchronized with the state of the server at time p_i, the information becoming available to the client immediately. At the next probe, the client is informed of all the updates at the server during the interval $(p_i, p_{i+1}]$, and so forth. We define $p_0 = 0$, $p_n = T$, and require that $0 \leq p_1 \leq p_2 \leq \cdots \leq p_{n-1} \leq T$.

At any time, the client data may be *obsolescent*, that is, out of synchronization with the server. Our ultimate concern is scheduling the connection times $\{p_i\}$. To do so, however, it is helpful to model the update process at the server first.

2.1 Information source update model

Similarly to [5, 8, 11], we model information updates at the server via a Poisson process, meaning that successive update events do not influence one another. Such an independence assumption seems plausible in sources with widely distributed access, *e.g.*, incoming emails, postings to newsgroups, or posting of orders from independent customers. However, we do need to capture natural time-variability trends, such as more emails arriving during work hours and more bids being posted towards the end of an auction. To do so, we use a nonhomogeneous Poisson process (see for example [15, 16]) with instantaneous arrival rate $\lambda : \Re \to [0, \infty)$. The number of update events occurring in any interval $(s, f]$ is a Poisson random variable with expected value $\Lambda(s, f) = \int_s^f \lambda(t)\, dt$. We assume that $\lambda(\cdot)$ is a nonnegative integrable function bounded above on $[0, T]$. To simplify some of the algebraic manipulations below, we extend the definition of $\Lambda(s, f)$ to the case $s > f$ through the standard convention $\int_u^v g(t)\, dt = -\int_v^u g(t)\, dt$ when $u > v$, that is, $\Lambda(s, f) = -\Lambda(f, s)$ when $s > f$.

For convenience, we provide a summary of our notation in Table 1; it includes symbols to be defined in later sections.

2.2 Cost model

Let $C(s, t)$ denote the obsolescence cost attributable to updates occurring at the server during the time interval $(s, t]$. Then the cost incurred between probes $i - 1$ and i is $C(p_{i-1}, p_i)$, and

T	Length of planning horizon
p_i	Time of probe $i = 0, \ldots, n$
δ	Minimum spacing between probes
$a(t)$	Client importance level assigned to time $t \in [0, T]$
$A(v, f)$	Accumulation of $a(t)$ in $(v, f]$
$\lambda(t)$	Instantaneous intensity of updates at time t
$\Lambda(s, f)$	Expected number of update events in the interval $(s, f]$
$C(s, t)$	Obsolescence cost from time s to t
$\overline{C}(s, t)$	Expected value of $C(s, t)$
$C(p_0, \ldots, p_n)$	Total obsolescence cost of schedule
$\overline{C}(p_0, \ldots, p_n)$	Expected value of $C(p_0, \ldots, p_n)$
$\overline{C}(s, t, u)$	Expected cost from s to u, with an intervening probe at $t \in [s, u]$
$t(r)$	Time update event r occurred
$i(r)$	Index of the probe transcribing update event r to the client
$c(r)$	Cost attributable to update r
a_1/a_2	Preference ratio; see (1)
$\mathcal{T}$	Finite set of possible probing times
t_i	Element of $\mathcal{T}$
$D(m, i)$	Minimum expected obsolescence cost after an update at t_i, with at most m further updates
$j(m, i)$	Index of next time to probe after t_i to attain expected cost $D(m, i)$
$K[i, j]$	Cost cache matrix

Table 1: Summary of notation

the total obsolescence cost of the probe schedule $\{p_0, p_1, \ldots, p_n\}$ is

$$C(p_0, p_1, \ldots, p_n) = \sum_{i=1}^{n} C(p_{i-1}, p_i) \tag{2}$$

The obsolescence cost is clearly a function of the update times and the times these updates are observed by the client, as well as the user's subjective perception of obsolescence. Intuitively, the shorter the time between the update and its observation by the client, the better off the client should be. Furthermore, as modeled by our function $a(\cdot)$, there may be times at which it is more important than others for the client to learn of updates.

We will denote individual updates by an index r, and let $t(r)$ denote the time r occurred at the server. Let $i(r) = \min\{i \mid p_i \geq t(r)\}$, that is, the index of the probe transcribing r to the client. We model the obsolescence costs $c(r)$ attributable to update r as follows: from its occurrence at time $t(r)$ to the subsequent probe at time $p_{i(r)}$, r accumulates obsolescence at the (time-varying) rate $a(t)$. Thus,

$$c(r) = \int_{t(r)}^{p_{i(r)}} a(t)\, dt. \tag{3}$$

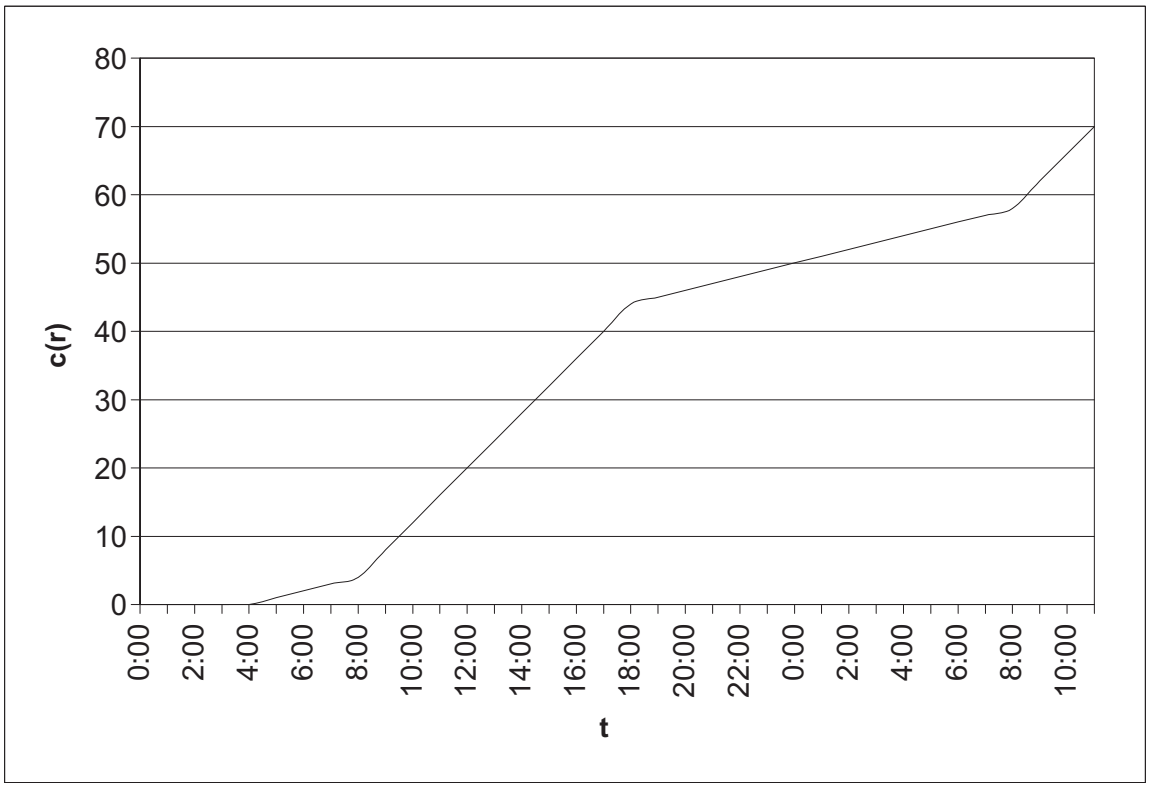


Figure 1: The obsolescence cost $c(r)$ (vertical axis) as a function of update time $p_{i(r)}$ (horizontal axis), using $a(\cdot)$ of the form (1).

Figure 1 illustrates the behavior of $c(r)$ in relation to the subsequent update time $p_{i(r)}$, where $a(t)$ takes the form (1). Update r occurs at 4 AM, and accumulates obsolescence linearly until work hours begin at 8AM. During work hours, $a(t)$ is larger, and obsolescence accumulates more rapidly until the close of business at 6 PM. It is worth noting that when $a(t)$ is a constant function, $c(r)$ takes a form resembling the age of a local element in [5].

More complex cost models involving nonlinear decay functions are also possible, but are not studied here. In the simple case of (1), we refer to a_1/a_2 as the *preference ratio*, rather than specifying a_1 and a_2 individually.

Having a model for $c(r)$, it is straightforward to write that the total obsolescence accumulating between probes $i-1$ and i is

$$C(p_{i-1}, p_i) = \sum_{r\,:\,t(r)\in(p_{i-1},p_i]} c(r). \tag{4}$$

The $C(p_{i-1}, p_i)$ and thus their sum $C(p_0, p_1, \ldots, p_n)$ are random variables. For any $s \le t$, we define $\overline{C}(s,t)$ to be the expected value of $C(s,t)$, and note that [8] proves that

$$\overline{C}(s,f) = \int_s^f \lambda(t) \left(\int_t^f a(\tau)\, d\tau \right) dt. \tag{5}$$

Defining $\overline{C}(p_0, p_1, \ldots, p_n)$ to be the expected value of $C(p_0, p_1, \ldots, p_n)$, we have from (2) that

$$\overline{C}(p_0, p_1, \ldots, p_n) = \sum_{i=1}^{n} \overline{C}(p_{i-1}, p_i) \tag{6}$$

An optimal probing schedule is then a choice of p_i that minimizes the expected obsolescence cost $\overline{C}(p_0, p_1, \ldots, p_n)$, that is, a solution to the optimization problem

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \overline{C}(p_{i-1}, p_i) \\ \text{S.T.} \quad & 0 = p_0 \leq p_1 \leq p_2 \leq \cdots \leq p_{n-1} \leq p_n = T, \end{aligned} \tag{7}$$

where $\overline{C}(p_{i-1}, p_i)$ is defined by (5). We call (7) the *polite probe scheduling problem.*

We can also imagine adding to (7) the constraint that no two probes can be closer than δ time units apart, where $\delta \geq 0$ is some given constant. We then obtain the model

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \overline{C}(p_{i-1}, p_i) \\ \text{S.T.} \quad & p_0 = 0 \\ & p_i - p_{i-1} \geq \delta \qquad\qquad i = 1, \ldots, n \\ & p_n = T, \end{aligned} \tag{8}$$

which is identical to (7) when $\delta = 0$.

3 Dynamic Programming Algorithm in Discrete Time

Unfortunately, the cost function of (7) and (8) has large numbers of local minima, and for simple, nonsmooth forms of $a(\cdot)$ and $\lambda(\cdot)$ is also nondifferentiable, having many points where derivatives with respect to the p_i do not exist. Figure 2 illustrates the case $n = 2$, plotting $\overline{C}(p_0, p_1, p_2)$ (on the vertical axis) as a function of $p_1 \in [0, T]$ (on the horizontal axis), where we fix $p_0 = 0$ and $p_2 = T$. Here, $T = 28$ days, $a(\cdot)$ is of the form (1) with $a_1/a_2 = 3$, and $\lambda(\cdot)$ is estimated from the DBWorld dataset we will describe in Section 5 (see Table 2). Standard continuous optimization techniques do not apply to problems of this form.

To avoid nondifferentiability, one can choose $\lambda(\cdot)$ to be a smooth function, and adopt a smooth form of $a(\cdot)$, as opposed to the discontinuous example given in (1). These adjustments result in $\overline{C}(s, f)$ having formal derivatives with respect to s and f, and give (7) a differentiable objective function. This approach will "smooth out" the sharp corners of the objective function $\overline{C}(p_0, p_1, \ldots, p_n)$ as illustrated in Figure 2, but only locally. The large numbers of local minima and the general "wiggly" shape of the cost function remain. As we have empirically verified, this structure makes it hard for standard nonlinear optimization methods to identify even local minima of the problem.

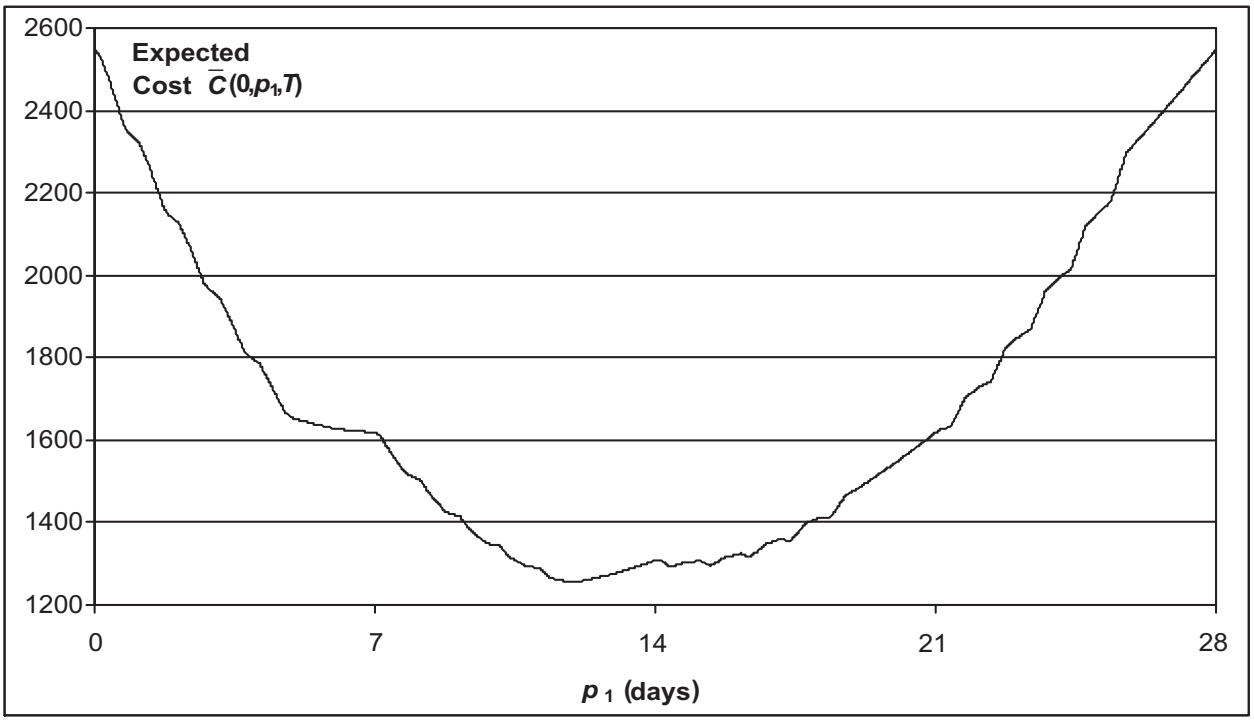


Figure 2: Sample graph of $\overline{C}(p_0, p_1, p_2)$ as a function of p_1.

To find an approximate solution to the problem, we suggest the following strategy. First, we identify a large but finite set $N \gg n$ of possible times $t_0 = 0 < t_1 < t_2 < \cdots < t_{N-1} < t_N = T$ at which we might want to place our n probes. Let $\mathcal{T} = \{t_0, t_1, \ldots, t_N\}$. If we constrain our scheduled probe times to lie within $\mathcal{T}$, we obtain the problem

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \overline{C}(p_{i-1}, p_i) \\ \text{S.T.} \quad & p_0 = 0 \\ & p_i - p_{i-1} \geq \delta \qquad\qquad i = 1, \ldots, n \\ & p_n = T \\ & p_1, p_2, \ldots, p_{n-1} \in \mathcal{T}. \end{aligned} \tag{9}$$

Since it has additional constraints, this problem clearly has a higher optimal objective value than (7). However, if N is large and the points of $\mathcal{T}$ "cover" $[0, T]$ sufficiently thickly, the difference should be slight; we provide a bound on the difference in optimal value between the two problems later in this section. The advantage of (9) is that it can be solved to global optimality by dynamic programming in polynomial time with respect to n and N.

Our dynamic programming approach to (9) proceeds as follows: for $m = 0, \ldots, n$ and $i = 0, \ldots, N-1$, define $D(m, i)$ to be the minimum feasible expected cost that can be accrued over the time interval $[t_i, T]$, given that a probe occured at t_i, one may perform at

most m additional probes, and probes may only occur at times in $\mathcal{T}$, with the last update falling at T.

We define $D(m, N) = 0$ for $m = 0, \ldots, M$, and letting a value of $+\infty$ denote that no feasible solution is possible, we have

$$D(1,i) = \begin{cases} \overline{C}(t_i, T) & t_i \leq T - \delta \\ +\infty & t_i > T - \delta, \end{cases} \qquad i = 0, \ldots, N-1. \tag{10}$$

Clearly, $D(n, 0)$ is the optimal value of problem (9). Next, take any $m \in \{0, \ldots, n\}$ and $i \in \{0, \ldots, N-1\}$. Considering that the first update after time t_i must occur at some time t_j, where $t_j \geq t_i + \delta$ and $i < j \leq N$, we obtain the recursion

$$D(m,i) = \min_{\substack{j \,:\, i<j\leq N \\ t_j \geq t_i + \delta}} \left\{ \overline{C}(t_i, t_j) + D(m-1, j) \right\}. \tag{11}$$

Define $j(m, i)$ to be some value of j attaining the minimum in (11). Note that if we know the values of $D(m-1, i)$ for all i, we can use (11) to calculate $D(m, i)$ for all i. Since the $D(0, i)$ are known, we can inductively use the recursion (11) to calculate all the $D(m, i)$ and $j(m, i)$, as follows:

Algorithm 1

For all $0 \leq i < j \leq N$, let $K[i, j] \leftarrow \overline{C}(t_i, t_j)$

for $i = 0, 1, \ldots, N$
  if $t_i \leq T - \delta$
    $D(1, i) \leftarrow \overline{C}(t_i, t_N)$
    $j(0, i) \leftarrow N$
  else
    $D(1, i) \leftarrow +\infty$
end
for $m = 2, 3, \ldots, n$
  for $i = 0, 1, \ldots, N$
    Choose $j(m, i)$ minimizing
      $K[i, j] + D(m-1, j)$ over $j = i+1, \ldots, N$ with $t_j \geq t_i + \delta$
    $D(m, i) \leftarrow K[i, j(m, i)] + D(m-1, j(m, i))$
  end
end.

$K[i, j]$ is a cache matrix saving values of $\overline{C}(t_i, t_j)$, so that $\overline{C}(\cdot, \cdot)$ does not have to be called repeatedly with the same arguments. Note that since we initialize $K[i, j]$ to $\overline{C}(t_i, t_j)$, the calculations of $D(m, i)$ and $j(m, i)$ agree with their definitions and (11).

Since the calculation of $j(m, i)$ takes $O(N)$ time, the complexity of the algorithm, except for the initialization of $K[\cdot, \cdot]$, is $O(nN^2)$.

At the end of the above calculation, $D(n,0)$ is known (it is not actually necessary to calculate $D(n,i)$ for $i > 0$). However, recall that our goal is to identify the optimal schedule itself.

To obtain the optimal schedule, note that it is optimal to perform the first update at time $t_{j(n,0)}$, so we set $p_1 = t_{j(n,0)}$. Setting $k \leftarrow j(n,0)$, it is optimal to perform the second update at time $t_{j(n-1,k)}$. We then set $k \leftarrow j(n-1,k)$, and proceed similarly to the third and subsequent updates. Formally, we construct the full optimal solution $p_0, p_1, \ldots, p_n$ in $O(n)$ time as follows:

Algorithm 2

$k \leftarrow 0$
$p_0 \leftarrow 0$
for $\ell = 1, \ldots, n$
    $k \leftarrow j(n+1-\ell, k)$
    $p_\ell \leftarrow t_k$
end.

Note that running Algorithm 1 for a given number of updates n has the side effect of also computing optimal schedules for all lesser numbers of updates $n' < n$. For example, the optimal cost for a schedule with $n-1$ updates is $D(n-1,0)$. Such schedules can be extracted by a slight modification to Algorithm 2.

Up to this point, the algorithm is applicable not only to our specific cost function, but to any scalar cost function $\overline{C}(p_0, p_1, \ldots, p_n)$ having the interval-wise additive structure expressed in (6). We now present some results that exploit the particular structure of the expected cost function (5).

First, to simplify some algebraic manipulations, we also formally define $\overline{C}(s,f)$ via (5) in the case $s > f$. Again using the standard negation rule for reversed integrals, we obtain for $s > f$ that

$$\begin{aligned}
\overline{C}(s,f) &= -\int_f^s \lambda(t) \left(\int_t^f a(\tau)\, d\tau \right) dt \\
&= -\int_f^s \lambda(t) \left(-\int_f^t a(\tau)\, d\tau \right) dt \\
&= \int_f^s \lambda(t) \left(\int_f^t a(\tau)\, d\tau \right) dt. \qquad (12)
\end{aligned}$$

Lemma 3 *For any times $s, v, f \in [0,T]$,*

$$\overline{C}(s,f) = \overline{C}(s,v) + \Lambda(s,v)A(v,f) + \overline{C}(v,f), \qquad (13)$$

where we let

$$A(v,f) = \int_v^f a(\tau)\, d\tau,$$

again using the standard negation rule $A(v,f) = -A(f,v)$ when $v > f$.

Proof.

$$\begin{aligned}
\overline{C}(s,f) &= \int_s^v \lambda(t) \int_t^f a(u)\, du\, dt + \int_v^f \lambda(t) \int_t^f a(u)\, du\, dt \\
&= \int_s^v \lambda(t) \left(\int_t^v a(u)\, du + \int_v^f a(u)\, du \right) dt + \overline{C}(v,f) \\
&= \int_s^v \lambda(t) \int_t^v a(u)\, du\, dt + \int_s^v \lambda(t) \int_v^f a(u)\, du\, dt + \overline{C}(v,f) \\
&= \overline{C}(s,v) + \Lambda(s,v)A(v,f) + \overline{C}(v,f).
\end{aligned}$$

□

Consider now the implementation of the first line of Algorithm 1, the construction of the cost cache matrix $K[i,j]$. We now exploit the particular properties of the cost formula (5). With this cost structure, the matrix $K[i,j]$ can be calculated especially efficiently, with only $O(N)$ evaluations of $\overline{C}(\cdot,\cdot)$. Applying Lemma 3, we have

$$\begin{aligned}
K[i,j] &= \overline{C}(t_i,t_j) \\
&= \overline{C}(t_i,t_{i+1}) + \Lambda(t_i,t_{i+1})A(t_{i+1},t_j) + \overline{C}(t_{i+1},t_j) \\
&= K[i,i+1] + \Lambda(t_i,t_{i+1})A(t_{i+1},t_j) + K[i+1,j]. \qquad (14)
\end{aligned}$$

We may thus populate the K matrix by first filling it with all zeroes, and then executing the following:

Algorithm 4

for $i = 0, 1, \ldots, N$
  $K[i,i+1] \leftarrow \overline{C}(t_i,t_{i+1})$
  $A[i] \leftarrow A(t_i,t_{i+1})$
  $L[i] \leftarrow \Lambda(t_i,t_{i+1})$
end
for $j = N, N-1, \ldots, 1$
  $A' \leftarrow 0$
  for $i = j-1, j-2, \ldots, 0$
    $A' \leftarrow A' + A[i+1]$
    $K[i,j] \leftarrow K[i,i+1] + L[i]A' + K[i+1,j]$
  end
end.

The first **for** loop caches the values of $\overline{C}(t_i,t_{i+1})$, $A(t_i,t_{i+1})$, and $\Lambda(t_i,t_{i+1})$ for all i. In the inner of the following two **for** loops, we update A' so that it contains $A(t_{i+1},t_j)$ at each iteration. Thus, the final assignment is identical to (14).

Suppose that E is some upper bound on the time to compute $\Lambda(t_i,t_{i+1})$, $A(t_i,t_{i+1})$, and $\overline{C}(t_i,t_{i+1})$ for any $i = 0, \ldots, N-1$. Then the calculation of K above executes in $O(NE+N^2)$ time, and the entire dynamic programming algorithm requires $O(NE+N^2) + O(nN^2) = O(NE+nN^2)$ time. To summarize our complexity analysis,

Proposition 5 *Algorithms 1, 2, and 4 together calculate an optimal solution to problem* (9) *in* $O(NE + nN^2)$*, time, where* E *is an upper bound on the time to compute* $\Lambda(t_i, t_{i+1})$*,* $A(t_i, t_{i+1})$*, and* $\overline{C}(t_i, t_{i+1})$*.*

If we choose the functions $\lambda(\cdot), a(\cdot)$ to be piecewise-polynomial with some fixed degree (in the examples and experiments in this paper, we choose the special case of piecewise-*constant* functions), then E has a bound proportional to the number of breakpoints of $\lambda(\cdot), a(\cdot)$ contained in any $[t_{i-1}, t_i]$. If we choose the $\{t_i\}$ and the breakpoints of $\lambda(\cdot), a(\cdot)$ in a non-pathological way — for example, evenly spaced in $[0, T]$ — then we will have $NE = O(N)$ and the overall complexity of Algorithm 1 simplifies to $O(nN^2)$. This complexity should allow for fairly large values of N.

We next consider how accurately the discretized optimal schedule computed by Algorithms 1 and 2 approximates the true optimal cost of the continuous problems (7) and (8). To this end, we require some bounding and sensitivity results for $\overline{C}(\cdot, \cdot)$.

Lemma 6 *For any times* $s, f \in [0, T]$*, we have* $0 \leq \overline{C}(s, f) \leq \frac{\bar{a}\bar{\lambda}}{2}(f - s)^2$*, where* $\bar{a}, \bar{\lambda} > 0$ *are upper bounds over* $[0, T]$ *on the functions* $a(\cdot)$ *and* $\lambda(\cdot)$*, respectively.*

Proof. Consider first the case $s \leq f$. Noting that $\lambda(t) > 0$ and $a(\tau) \in [0, \bar{a}]$ in (5), we first obtain

$$0 \leq \overline{C}(s, f) \leq \int_s^f \lambda(t) \left(\int_t^f \bar{a} \, d\tau \right) dt = \int_s^f \lambda(t) \cdot \bar{a} \cdot (f - t) \, dt$$

Since $\lambda(t) \in [0, \bar{\lambda}]$ above, we further obtain

$$0 \leq \overline{C}(s, f) \leq \int_s^f \bar{\lambda}\bar{a} \cdot (f - t) \, dt = \frac{\bar{a}\bar{\lambda}}{2}(f - s)^2.$$

For the case $s > f$, we apply the same logic to (12). □

Lemma 7 *Suppose* $\epsilon > 0$ *and* $s, f, s', f' \in [0, T]$ *are such that* $|s - s'|, |f - f'| \leq \epsilon$*. Then*

$$\left| \overline{C}(s, f) - \overline{C}(s', f') \right| \leq 2\bar{a}\bar{\lambda} \left(\epsilon^2 + |f - s| \, \epsilon \right). \tag{15}$$

Proof. Applying Lemma 3 twice,

$$\begin{aligned} \overline{C}(s', f') &= \overline{C}(s', s) + \Lambda(s', s)A(s, f') + \overline{C}(s, f') \\ &= \overline{C}(s', s) + \Lambda(s', s)A(s, f') + \overline{C}(s, f) + \Lambda(s, f)A(f, f') + \overline{C}(f, f'), \end{aligned}$$

which rearranges to

$$\overline{C}(s', f') - \overline{C}(s, f) = \overline{C}(s', s) + \Lambda(s', s)A(s, f') + \Lambda(s, f)A(f, f') + \overline{C}(f, f').$$

Taking absolute values and applying the triangle inequality,

$$\left| \overline{C}(s, f) - \overline{C}(s', f') \right| \leq \overline{C}(s', s) + |\Lambda(s', s)| \, |A(s, f')| + |\Lambda(s, f)| \, |A(f, f')| + \overline{C}(f, f').$$

From this inequality, Lemma 6, the bounds on $a(\cdot)$ and $\lambda(\cdot)$, and $|s-s'|, |f-f'| \leq \epsilon$, we may then deduce

$$\begin{aligned} \left|\overline{C}(s,f) - \overline{C}(s',f')\right| &\leq \frac{\bar{a}\bar{\lambda}}{2}\epsilon^2 + \bar{\lambda}\epsilon \cdot \bar{a}\,|f'-s| + \bar{\lambda}\,|f-s| \cdot \bar{a}\epsilon + \frac{\bar{a}\bar{\lambda}}{2}\epsilon^2 \\ &\leq \bar{a}\bar{\lambda}\epsilon^2 + \bar{a}\bar{\lambda}\epsilon\left(|f-s|+\epsilon\right) + \bar{a}\bar{\lambda}\,|f-s|\,\epsilon \\ &= 2\bar{a}\bar{\lambda}\left(\epsilon^2 + |f-s|\,\epsilon\right). \end{aligned}$$

□

Lemma 7 bounds the changes in $\overline{C}(s,f)$ that can result from perturbations to s and f. This result makes it possible to bound the cost of appending the discretization constraint $p_1, \ldots, p_{n-1} \in \mathcal{T}$ to the continuous problems (7) or (8):

Proposition 8 *Let C^* be the optimal value of (8) and $\ddot{C}$ be the optimal value of the discretized problem (9), as computed by Algorithm 1. Suppose that either of the following holds:*

(i) $\delta = 0$, in which case we define $\gamma = \max_{j=1,\ldots,N}\{t_j - t_{j-1}\}$. In this case, (8) reduces to (7).

(ii) $\delta > 0$, and the $\{t_j\}$ are evenly spaced γ time units apart, where δ is a multiple of γ. Specifically, for some integer $k \geq 1$ and all $j = 1, \ldots, N$, $t_j - t_{j-1} = T/N = \gamma = \delta/k$.

Then $\ddot{C} \leq C^* + \bar{a}\bar{\lambda}\left((n/2)\gamma^2 + T\gamma\right)$.

Proof. We first note that Lemma 7 implies that $\overline{C}(\cdot,\cdot)$ is continuous on $[0,T]^2$, and the feasible region of (8) is closed and bounded. Therefore, the minimum value of (8) is attained by at least one feasible point $p^* = (p_0^*, p_1^*, \ldots, p_n^*) \in \Re^{n+1}$. Consider now the vector $\hat{p} = (\hat{p}_0, \hat{p}_1, \ldots, \hat{p}_n)$ obtained by rounding each p_i^* to the closest member of $\mathcal{T}$, rounding downwards in case of a tie. In both cases (i) and (ii), the maximum distance between elements of $\mathcal{T}$ is γ, so we have $|\hat{p}_i - p_i^*| \leq \gamma/2$ for all i. Applying Lemma 7, we have for all i that

$$\begin{aligned} \overline{C}(\hat{p}_{i-1}, \hat{p}_i) &\leq \overline{C}(p_{i-1}^*, p_i^*) + 2\bar{a}\bar{\lambda}\left((\gamma/2)^2 + (\gamma/2)(p_i^* - p_{i-1}^*)\right) \\ &= \overline{C}(p_{i-1}^*, p_i^*) + \bar{a}\bar{\lambda}\left(\gamma^2/2 + \gamma(p_i^* - p_{i-1}^*)\right) \end{aligned}$$

Adding the above inequality for $i = 1, \ldots, n$ produces

$$\begin{aligned} \overline{C}(\hat{p}_0, \hat{p}_1, \ldots, \hat{p}_n) &\leq \overline{C}(p_0^*, p_1^*, \ldots, p_n^*) + \bar{a}\bar{\lambda}\left(\sum_{i=1}^n \gamma^2/2 + \sum_{i=1}^n \gamma(p_i^* - p_{i-1}^*)\right) \\ &= C^* + \bar{a}\bar{\lambda}((n/2)\gamma^2 + T\gamma). \end{aligned}$$

Now, if $\hat{p}$ is feasible for (9), then $\ddot{C}$'s optimality for (9) implies $\ddot{C} \leq \overline{C}(\hat{p}_0, \hat{p}_1, \ldots, \hat{p}_n) \leq C^* + \bar{a}\bar{\lambda}((n/2)\gamma^2 + T\gamma)$. Therefore, it suffices to show that $\hat{p}$ is feasible for (9). By construction, $\hat{p}_i \in \mathcal{T}$ for all i, and since $t_0 = 0$ and $t_N = T$, we have $\hat{p}_0 = 0$ and $\hat{p}_n = t_N = T$. Thus, it is sufficient to show $\hat{p}_i - \hat{p}_{i-1} \geq \delta$ for $i = 1, \ldots, n$.

In case (i), we have $\delta = 0$, so we simply observe that since $p_{i-1}^* \leq p_i^*$ and the rounding operation is a nondecreasing map, we have $\hat{p}_{i-1} \leq \hat{p}_i$ for all i. Thus, it remains only to consider case (ii). We treat this remaining case by taking any $i \in \{1, \ldots, n\}$ and considering three subcases:

- First, suppose $\hat{p}_i \leq p_i^*$. Let $j \in \{1, \ldots, N\}$ be such that $\hat{p}_i = t_j$. Then

$$\begin{aligned} & p_i^* \leq \hat{p}_i + \gamma/2 \\ \Rightarrow \quad & p_{i-1}^* \leq \hat{p}_i + \gamma/2 - \delta = t_{j-k} + \gamma/2 \\ \Rightarrow \quad & \hat{p}_{i-1} \leq t_{j-k} = \hat{p}_i - \delta. \end{aligned}$$

- If we do not have $\hat{p}_i \leq p_i^*$, next suppose that $\hat{p}_{i-1} \geq p_{i-1}^*$. In this case, let $j \in \{0, \ldots, N\}$ be such that $t_j = \hat{p}_{i-1}$. Then

$$\begin{aligned} & p_{i-1}^* \geq \hat{p}_{i-1} - \gamma/2 \\ \Rightarrow \quad & p_i^* \geq \hat{p}_{i-1} - \gamma/2 + \delta = t_{j+k} - \gamma/2 \\ \Rightarrow \quad & \hat{p}_i \geq t_{j+k} = \hat{p}_{i-1} + \delta. \end{aligned}$$

- If neither of the above cases apply, then we must have $\hat{p}_i > p_i^*$ and $\hat{p}_{i-1} < p_{i-1}^*$. Then we simply deduce

$$\hat{p}_{i-1} < p_{i-1}^* \leq p_i^* - \delta < \hat{p}_i - \delta.$$

□

Applying this result in the case of evenly-spaced $\{t_j\}$ and thus $\gamma = T/N$, we immediately obtain the following corollary. Note that the restriction placed on N and δ is only operative when $\delta > 0$; when $\delta = 0$, it always holds.

Corollary 9 *Suppose $\mathcal{T}$ consists of evenly spaced points, that is, $t_j = j(T/N)$ for $j = 0, \ldots, N$, and we consider only values of N such that δ is a nonnegative integer multiple of T/N. Then $\ddot{C} - C^* = \mathrm{O}(1/N)$, that is, (9)'s optimal value appromimates (8)'s by an additive $\mathrm{O}(1/N)$ factor.*

4 Improving a Probe Schedule in Continuous Time

Algorithm 1 provides a probe schedule $\{p_i\}_{i=1}^n$ which is optimal subject to the constraint $p_0, p_1, \ldots, p_n \in \mathcal{T} = \{t_0, t_1, \ldots, t_N\}$. For large N and reasonably distributed $\{t_i\}$, we have just proven that the resulting solution should be nearly optimal for the original probe scheduling problem (8).

We next present a method for taking any probe schedule $\{p_i\}_{i=1}^n$ and improving its expected cost $\overline{C}(p_0, \ldots, p_n)$. The method is a simple direct search method, and does not guarantee either global or true local optimality in the classic differential sense. However, it strictly improves the expected cost of the solution, and is reliable even for the ill-behaved cost functions generated by our model. Our main purpose is to apply this improvement procedure to the output of Algorithm 1, but it can be applied from any starting point $\{p_i\}_{i=1}^n$ meeting the constraints of (8).

Suppose three consecutive probes occur at times s, t, u, where $s + \delta \leq t \leq u - \delta$. First, define $\overline{C}(s, t, u) = \overline{C}(s, t) + \overline{C}(t, u)$. Next, take any $\mu \in (0, 1]$, we define $\Delta_\mu^-(s, t, u)$ to be

the improvement in expected cost that would result from moving t a fraction μ of the way towards $s+\delta$, that is, replacing t with $\mu(s+\delta)+(1-\mu)t$. We define $\Delta^+_\mu(s,t,u)$ similarly, but moving t a fraction μ of the way towards $u-\delta$. Formally, we set

$$\Delta^-_\mu(s,t,u) = \overline{C}(s,t,u) - \overline{C}(s,\mu(s+\delta)+(1-\mu)t,u)$$
$$\Delta^+_\mu(s,t,u) = \overline{C}(s,t,u) - \overline{C}(s,(1-\mu)t+\mu(u-\delta),u).$$

If $\Delta^-_\mu(s,t,u) > 0$, then the expected cost incurred in $[s,u]$ will be improved by replacing $t \leftarrow \mu(s+\delta)+(1-\mu)t$; similarly, if $\Delta^+_\mu(s,t,u) > 0$, taking $t \leftarrow (1-\mu)t+\mu(u-\delta)$ will lower the expected cost on $[s,u]$. We call these operations a *left μ-move* and a *right μ-move* on (s,t,u), respectively.

We propose improving the cost of the schedule $p_0,\ldots,p_n$ by looking at all possible left or right μ-moves involving consecutive probes p_{i-1},p_i,p_{i+1}, where $i=1,\ldots,n-1$. We start by setting μ to some starting value μ_0. If $\delta>0$, we would typically choose $\mu_0=1$, whereas if $\delta=0$, we would typically select a smaller value like $\mu_0=1/2$, since placing two probes at exactly the same time can only increase a schedule's cost. We then iterate in a greedy manner by choosing the left or right μ-move that yields the greatest decrease in the cost, that is, corresponds to the largest value of $\Delta^\sigma_\mu(p_{i-1},p_i,p_{i+1})$ for $\sigma=+,-$ and $i=1,\ldots,n-1$. We repeatedly choose this most improving μ-move until there is no such move that improves the cost significantly, that is, $\Delta^\sigma_\mu(p_{i-1},p_i,p_{i+1}) < \iota$ for all $\sigma=+,-$ and $i=1,\ldots,n-1$, where $\iota>0$ is some small constant. When this occurs, we reduce μ by taking $\mu \leftarrow \rho\mu$, where $\rho\in(0,1)$ is another tuning parameter. We repeat in this manner until μ drops below some stopping threshold μ_*, and then halt.

Algorithm 10 *Given an initial schedule $p_0,\ldots,p_n$, and parameters $\mu_0\in(0,1]$, $\mu_*\in(0,\mu]$, $\rho\in(0,1)$, and $\iota>0$, modify $p_1,\ldots,p_{n-1}$ as follows:*

$\mu \leftarrow \mu_0$
while $(\mu \ge \mu_*)$
  while $(\max_{\substack{\sigma=+,-\\ i=1,\ldots,n-1}} \Delta^\sigma_\mu(p_{i-1},p_i,p_{i+1}) \ge \iota)$
    Choose $\sigma\in\{+,-\}$, $i\in\{1,\ldots,n-1\}$
      minimizing $\Delta^\sigma_\mu(p_{i-1},p_i,p_{i+1})$
    if $(\sigma=-)$
      $p_i \leftarrow \mu(p_{i-1}+\delta)+(1-\mu)p_i$
    else if $(\sigma=+)$
      $p_i \leftarrow (1-\mu)p_i+\mu(p_{i+1}-\delta)$
  end
  $\mu \leftarrow \rho\mu$
end.

Suppose now that we implement Algorithm 10 by keeping the values $\{\Delta^\sigma_\mu(p_{i-1},p_i,p_{i+1})\}$, where $\sigma=\{+,-\}$ and $i=1,\ldots,n-1$, in a heap of size $2(n-1)$. If we keep appropriate

data structures, the maximum element can then be identified in O(1) time. Furthermore, each μ-move only alters the value of p_i, which means the only elements in the heap that can change value are

$$\begin{array}{ll} \Delta_\mu^-(p_{i-2}, p_{i-1}, p_i) & \Delta_\mu^+(p_{i-2}, p_{i-1}, p_i) \\ \Delta_\mu^-(p_{i-1}, p_i, p_{i+1}) & \Delta_\mu^+(p_{i-1}, p_i, p_{i+1}) \\ \Delta_\mu^-(p_i, p_{i+1}, p_{i+2}) & \Delta_\mu^+(p_i, p_{i+1}, p_{i+2}), \end{array}$$

two of which will not exist in the boundary cases $i = 1$ and $i = n - 1$. Thus, at most six elements of the heap will be out of position after p_i is changed, and they may be repositioned correctly in $O(\log n)$ time. Thus, the inner loop of Algorithm 10 can be executed in $O(\log n)$ time. At the outset, and whenever μ is changed, the entire heap needs to be recalculated, taking $O(n \log n)$ time.

While the number of iterations of Algorithm 10's outer loop can be easily predicted, the number of iterations of the inner loop cannot. However, since $\overline{C}(p_0, \ldots, p_n)$ is always nonnegative and each iteration reduces $\overline{C}(p_0, \ldots, p_n)$ by at least $\iota > 0$, it follows that the algorithm must terminate in a finite number of steps. The empirical tests below suggest that Algorithm 10 has acceptable iteration counts for realistic input data.

It would be desirable to prove that if Algorithm 10 were to be run indefinitely by setting $\iota = \mu_* = 0$, it would asymptotically approach a classically defined local minimum of problem (8). Unfortunately, $\overline{C}(\cdot, \cdot)$ and hence the objective function of (8) are in general nondifferentiable, raising the possibility of various kinds of "jamming" — that is, inability to make progress along coordinate directions, even though the current point is not a local minimum. Even if we choose $\lambda(\cdot)$ and $a(\cdot)$ to be smoothly differentiable functions, making the objective function of (8) differentiable, the constraints $p_i - p_{i-1} \leq \delta$ can still induce a form of jamming. Such jamming could be overcome by applying techniques of constrained pattern search, as found for example in [10]. In particular, one could adapt Algorithms 8.1 and 8.2 of [10] to produce schedule improvement methods with desirable asymptotic properties[5] in the case that $\lambda(\cdot)$ and $a(\cdot)$ are smoothly differentiable. However, such methods would be far more complicated than Algorithm 10, because they would in general have to examine more search directions than Algorithm 10's simple axis-parallel directions. Specifically, the algorithm would have to identify "blocks" of consecutive probes that are within some distance $\epsilon > 0$ of being binding for the packing/sequencing constraints $p_i - p_{i-1} \leq \delta$. Such blocks would consist of indices $k, k+1, \ldots, k+\ell$ for which

$$\begin{array}{rcl} p_{k+1} & \leq & p_k + \delta + \epsilon \\ p_{k+2} & \leq & p_{k+1} + \delta + \epsilon \\ & \vdots & \\ p_{k+\ell} & \leq & p_{k+\ell-1} + \delta + \epsilon \end{array}$$

[5] One such property is $\chi(p) \to 0$, where $\chi(p)$ is an error bound function measuring the violation of the KKT first-order necessary conditions for a local minimum.

For each such block, the method would have to check 2ℓ possible search directions, corresponding to moving the first $m \leq \ell$ probes downward by the same offset or the last $m \leq \ell$ probes upward by the same offset, that is, for some $\theta > 0$ and $m \in \{1, \ldots, k\}$,

$$p_i \leftarrow p_i - \theta \qquad \forall\, i = k, k+1, \ldots, k+m$$

or

$$p_i \leftarrow p_i + \theta \qquad \forall\, i = k+\ell-m, k+\ell-m+1, \ldots, k+\ell.$$

Single coordinate moves, as practiced by Algorithm 10, are the $\ell = 1$ special case of this pattern.

We have not pursued this line of inquiry further at the moment, since simple choices of $\lambda(\cdot)$ and $a(\cdot)$ may not be continuously differentiable, and the global optimality properties of the discretized schedule obtained by Algorithm 1 are likely to be of far greater importance in practice than finding an exact *local* minimum of the expected cost function. Thus, we have investigated only the fairly simple local improvement technique embodied in Algorithm 10.

It is also possible that, instead of applying Algorithm 10's simple greedy method, one could apply any number of popular metaheuristics, such as tabu search or simulated annealing, to the basic neighborhood structure defined by left and right μ-moves. Again, in view of the near-optimality of our dynamic programming algorithm, we have not investigated such approaches.

5 Numerical Experiments

We now describe a series of experiments in which we apply Algorithm 1 and Algorithm 10 to real-world data traces, comparing their performance with previously-proposed methods.

5.1 Test data and problem parameters

To evaluate the algorithms empirically, we used two real-world data traces. The first is a 833-data-point trace of postings to the DBWorld bulletin board between November 9, 2000 and June 13, 2001. The top part of Figure 3 presents a scatter plot with a sample of the DBWorld trace data. The horizontal axis represents dates ranging from September 11, 2000 to March 29, 2001, while the vertical axis indicates the time of day that an update event occurred. The bottom data trace in Figure 3 is for MicroNet, a UC Berkeley university forum. The data for the MicroNet trace were collected in 2000 and 2001 (illustrated in the bottom part of Figure 3), as well as the first nine months of 2003. For MicroNet, we have only recorded thread initiators, *i.e.*, those messages that start a new thread of discussion. Other messages (recognized by an initial "Re:") were discarded.

We divided each trace into two portions, a *training set* and a *testing set*. For DBWorld, the training set contains the first 580 data points and the testing set contains the remaining 253 data points. For MicroNet, we used the 174 data points collected in the second half of 2000 as the training set. The testing set contains 749 data points collected from January 2

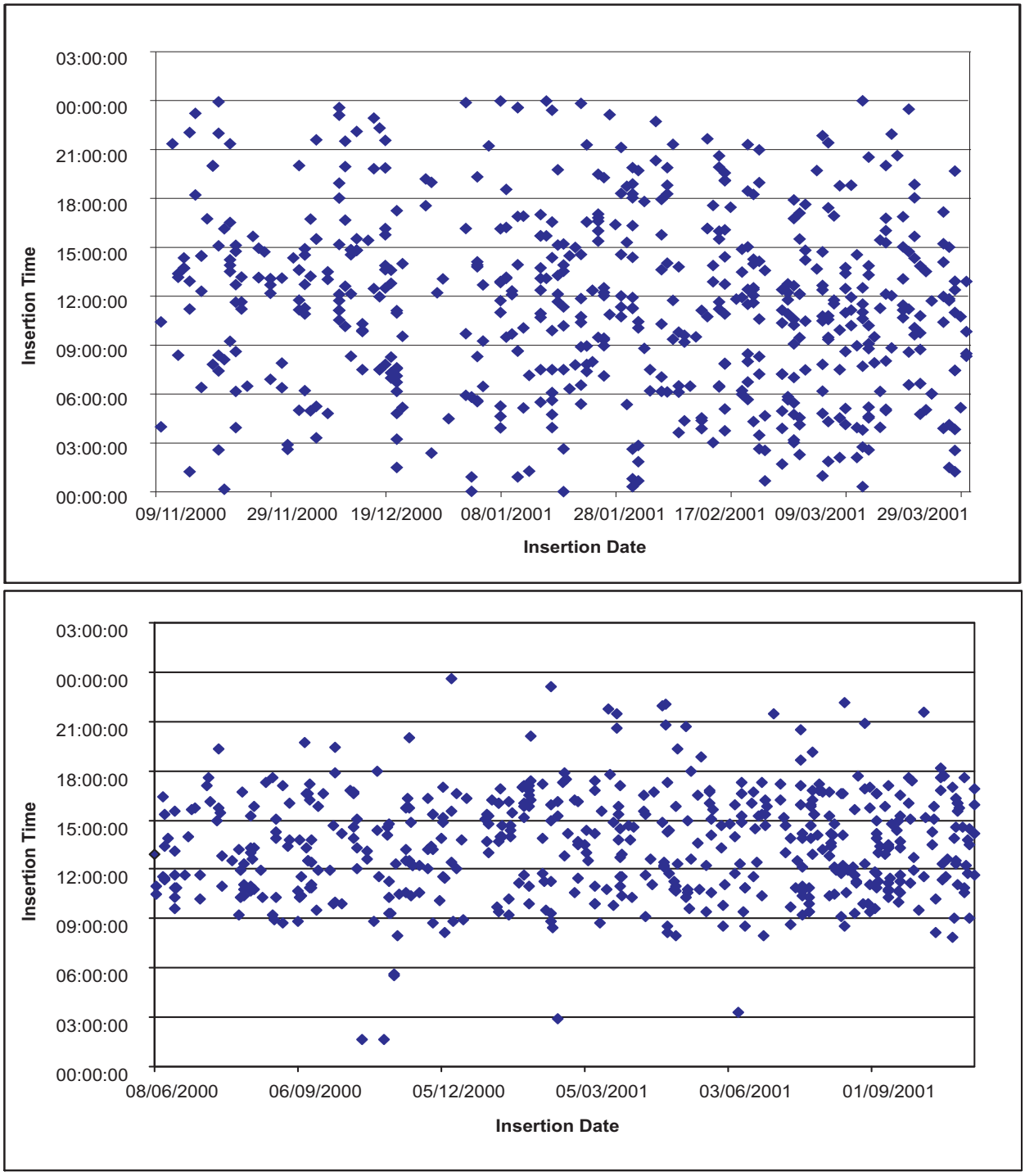


Figure 3: The DBWorld (top) and MicroNet (bottom) datasets.

	Weekdays	Saturday	Sunday
[0:00, 3:00)	2.40	1.50	1.15
[3:00, 6:00)	5.96		
[6:00, 9:00)	6.04		
[9:00, 18:00)	7.50		
[18:00, 21:00)	3.03		
[21:00, 24:00)	2.41		

Table 2: $\lambda(t)$ pattern for the DBWorld trace.

	Weekdays	Weekend
[0:00, 9:00)	0.25	0.08
[9:00, 19:00)	2.60	
[19:00, 24:00)	0.14	

Table 3: $\lambda(t)$ pattern for the MicroNet trace.

through September 5, 2003. For both traces, we set both the probe scheduling period and testing set length to four weeks.

To obtain an update intensity function $\lambda(\cdot)$, we modeled each dataset via a repetitive piecewise constant Poisson process, similarly to [8]. We thus took $\lambda(\cdot)$ to be periodic piecewise constant function, repeating itself over a fixed time period, which we chose to be one week for both data traces. We divided each week into fixed intervals, within which $\lambda(t)$ is constant, and then applied the standard maximum likelihood estimation formula for the arrival rate within each interval. Tables 2 and 3 show the $\lambda(t)$ levels fitted to the DBWorld and MicroNet training sets, respectively.

We modeled $a(\cdot)$ using (1), with work hours defined to be [9:00, 18:00) on weekdays for the DBWorld trace and [9:00, 19:00) on weekdays for the MicroNet trace. In our experiments, we used four different preference ratios, specifically $a_1/a_2 = 1, 2, 3, 4$. The larger the value of a_1/a_2 ratio, the more emphasis is given to work hours. In all our experiments, we fixed $\delta = 0$.

5.2 Algorithm metrics and parameters

We evaluated the performance of the algorithms with respect to the following criteria:

- The expected cost $\overline{C}(p_0, \ldots, p_n)$ over the planning period
- The actual obsolescence cost $C(p_0, \ldots, p_n)$ over the *testing set*
- The number of iterations of Algorithm 10.

Note that the actual cost $C(p_0, \ldots, p_n)$ is subject to random fluctuations arising from the training set sample, but has the advantage of measuring the combined utility of our model and scheduling procedure on a real data stream. On the other hand, $\overline{C}(p_0, \ldots, p_n)$ measures only the desirability of the schedule $p_0, \ldots, p_n$ under the assumption that the cost model is accurate.

In our experiments with Algorithm 1, we varied the number of probes n, holding N constant at 672. We set the possible probe times t_i to be evenly spaced, one hour apart.

The experiments compare the relative performance of four different policies: *uniform*, *threshold*, *first arrival* (*FA*), and the schedule generated by Algorithms 1, 2, and 4, which we call *Discrete Optimal Expected* (*DOE* for short). The *uniform* policy simply spaces n probes evenly throughout $[0, T]$. Such uniform allocation schedules were proposed for Web crawling in [5]. In the *threshold* policy, proposed in [8], the user provides a cost parameter Π. Once the expected cost exceeds Π, a probe takes place. Finally, the *first arrival* policy, proposed in [11, 8], probes whenever the probability of having any new data exceeds some threshold π, regardless of the current importance level or the total number of data updates that might be pending transcription. Note that this policy uses only the update portion of the model, and not the full cost model and its importance function $a(\cdot)$.

We began by generating schedules with the threshold and first arrival policies, for various values of Π and π. For each schedule, we recorded n, the number of actual probes. Then, we ran the uniform and DOE policies with the same n, and compared the results.

In all the experiments with Algorithm 10, we took the initial shifting parameter to be $\mu_0 = 1/2$, the final shifting factor to be $\mu_* = 1/512$, and the reduction parameter to be $\rho = 1/2$.

We now describe two sets of experiments. The first set compares the policies without using Algorithm 10 for local improvement. In the second set of experiments, we analyze the performance of Algorithm 10 with its starting schedule set by each of the four policies.

5.3 Experiments with Algorithm 1

For both the DBWorld and MicroNet datasets, Figure 4 presents a comparison of the *expected* cost $\overline{C}(p_0, \ldots, p_n)$ obtained through each of the four policies, based on the model constructed from the training data, and with a preference ratio of 3. To avoid cluttering the figures, we present only the lower portion of the expected cost axis; our analysis of the results holds for the whole spectrum.

The expected cost for all polices falls as n increases, which accords with intuition, although DOE is the only policy for which it is theoretically guaranteed (if we hold $\mathcal{T}$ fixed). Also, as one would expect from its derivation, the DOE policy yields the lowest expected cost for any given n. As n grows, the differences between the different policies shrink. While the two traces are qualitatively similar, DOE performs better for the MicroNet data.

Figure 5 also shows expected costs, but for a preference ratio of 1. As can be readily seen, the results are similar, and DOE still outperforms all other policies. We observed this phenomenon for all preference ratios. From this point on, we therefore focus on just a single

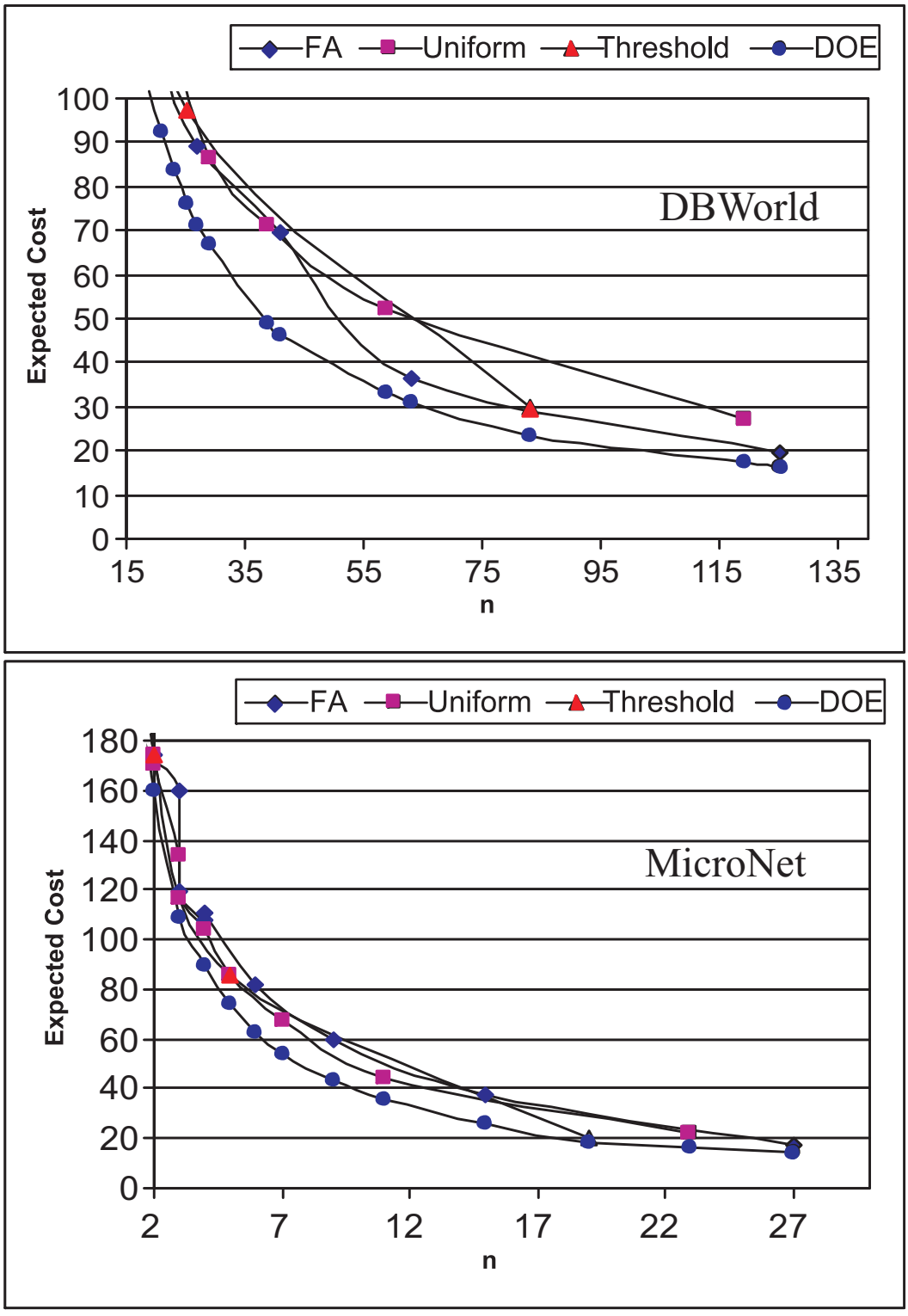


Figure 4: Comparison of the expected costs of the four policies, for a preference ratio of 3.

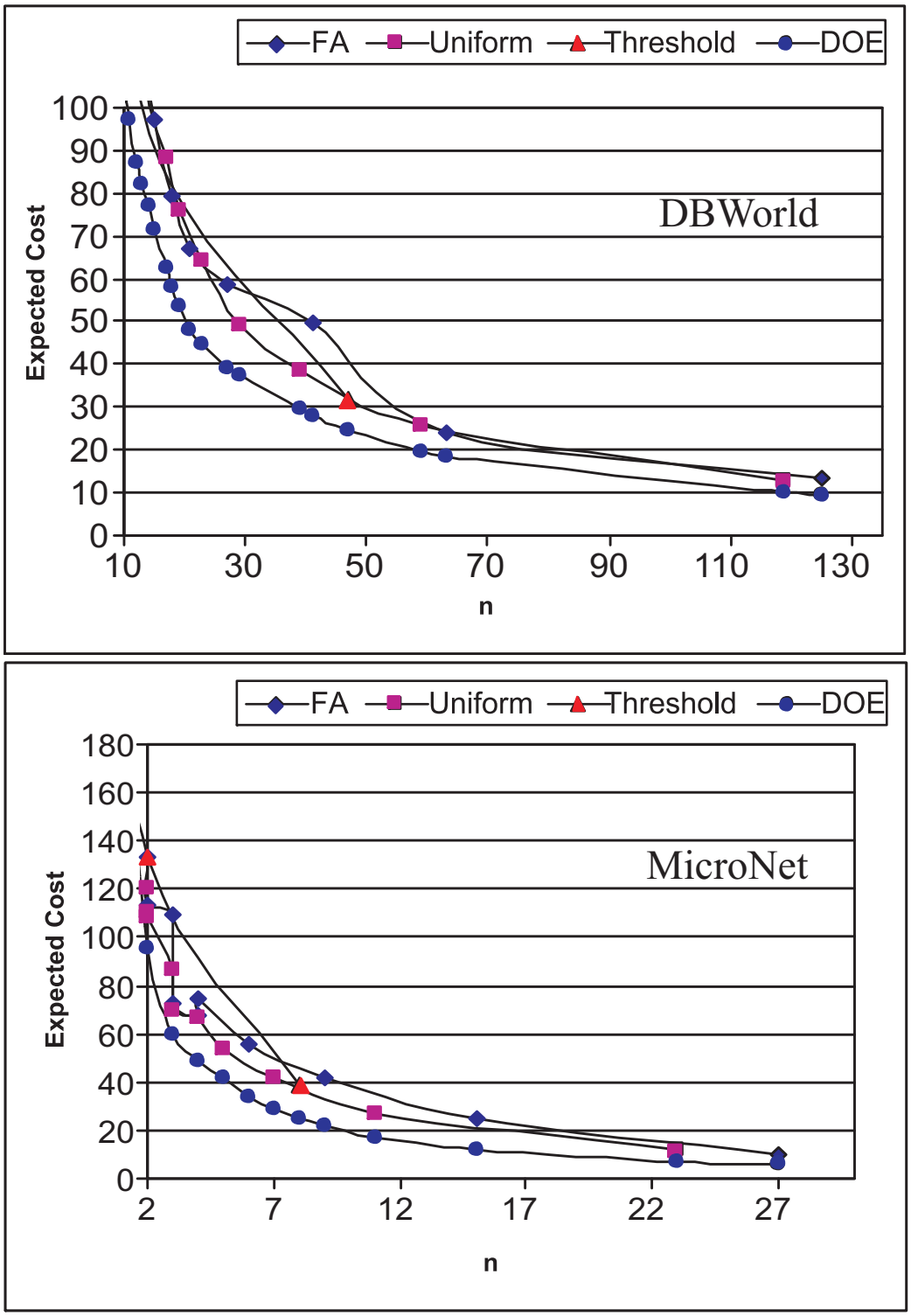


Figure 5: Comparison of the expected costs of the four policies for a preference ratio of 1.

value of the preference ratio, for which we have chosen $a_1/a_2 = 3$.

Figure 6 presents a comparison of the *actual* cost realized by each of the four policies, evaluated on the testing set. These data are affected both by any inaccuracies in the update model embodied by $\lambda(\cdot)$, and random variations in the testing set sample. For DBWorld, DOE still outperforms all other policies. For MicroNet, while DOE outperforms other policies for $n \geq 7$, it is sometimes inferior to other policies when $n < 7$. Note that such small values of n correspond to very infrequent polling — fewer than once every four days. For such low values of n, the random variations in the data stream have particularly high impact on the actual cost.

Figure 7 presents the difference between the actual and the expected cost of the DOE schedule, normalized by the actual cost. For DBWorld, random variations generate large differences for low values of n, but as n grows, the discrepancies fall to the order of $\pm 10\%$. For MicroNet, the actual costs are uniformly higher than expected, most likely indicating that MicroNet usage increased in the two years that elapsed between the training and testing sets. Evidently, however, the relative weekly update intensity pattern must have remained similar, since DOE still produces the best probe schedules, except for very small values of n.

5.4 Experiments with Algorithm 10

We next turn our attention to our improvement method, Algorithm 10, considering how the starting probe schedule affects the cost of the final schedule produced by the improver. We applied the improver to each of the four policies of the last section, and compared the resulting costs.

Figure 8 shows the expected costs of improved schedules seeded by each of the four policies. The improvement method seems to be quite effective, in that the differences between expected costs are far less pronounced than those prior to improvement, as shown in Figure 4. Still the different initial policies do lead to different improved costs, reflecting that the improver cannot guarantee a global optimum. Second, close inspection shows that the DOE policy still performs the best, probably as the result of producing better initial schedules.

Figure 9 shows the actual costs of the improved schedules on the testing sets. For DBWorld, the results are very similar to Figure 8's, with DOE being the best policy by a narrow margin. For MicroNet, DOE is markedly better than the other policies, except for one value of n where it is slightly outperformed by the threshold policy. We are not sure how to explain this phenomenon. For MicroNet, the training and testing sets are more qualitatively different than for DBWorld, and somehow the improved DOE policies were generally more robust to these differences than the other improved policies, despite the rather small differences in the expected costs. Further experimentation on additional data feeds would be required to determine whether the phenomenon is significant or common.

We next discuss the degree of improvement achieved by Algorithm 10. Figure 10 shows the expected cost improvement obtained for each policy's initial schedule, as a fraction of the initial expected cost. DOE shows less improvement than any of the other policies, most likely reflecting that it generates better schedules to start with. DOE also improves more for

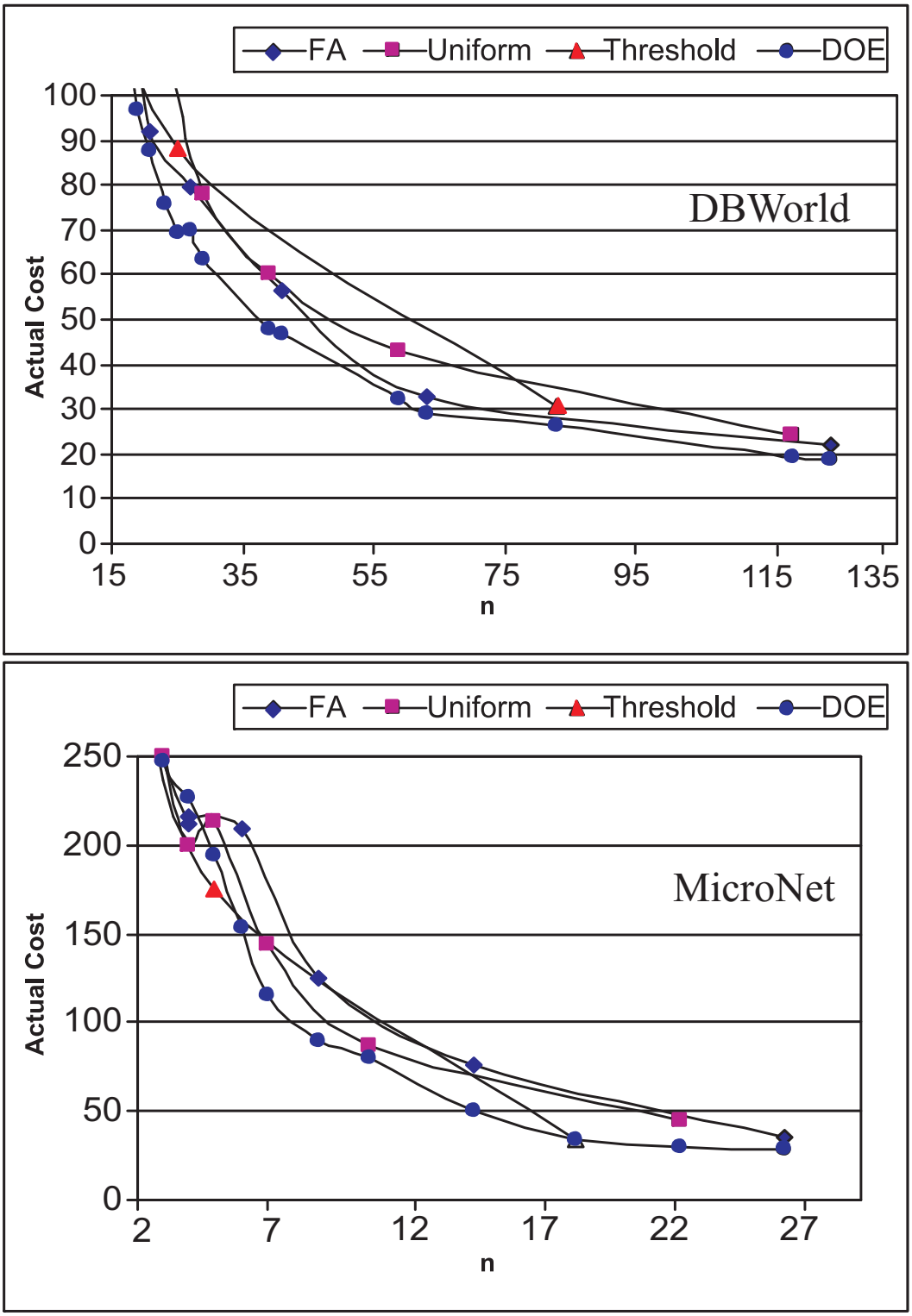


Figure 6: Comparison of the actual costs of the four policies.

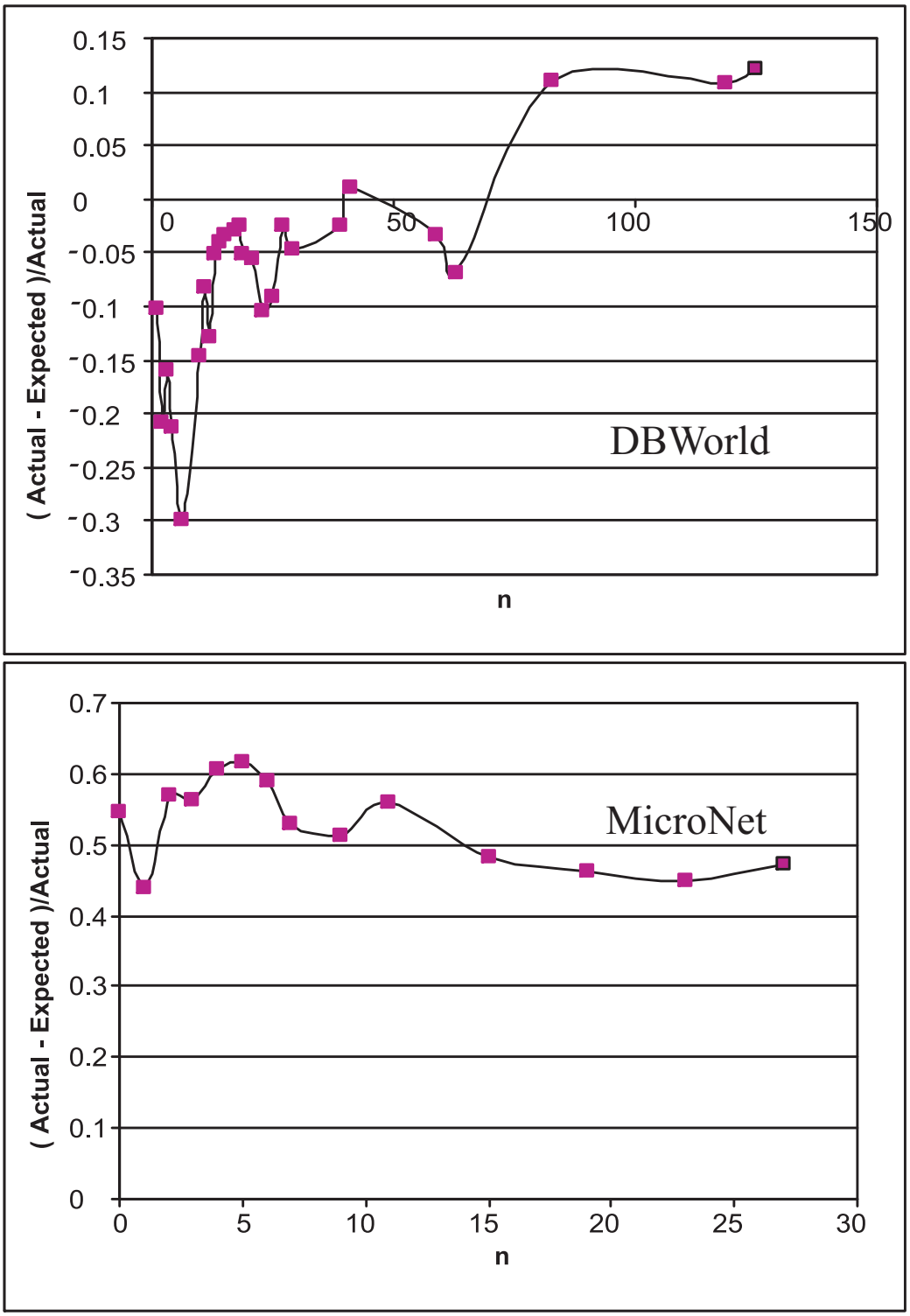


Figure 7: Comparison of actual and expected cost of the DOE policy.

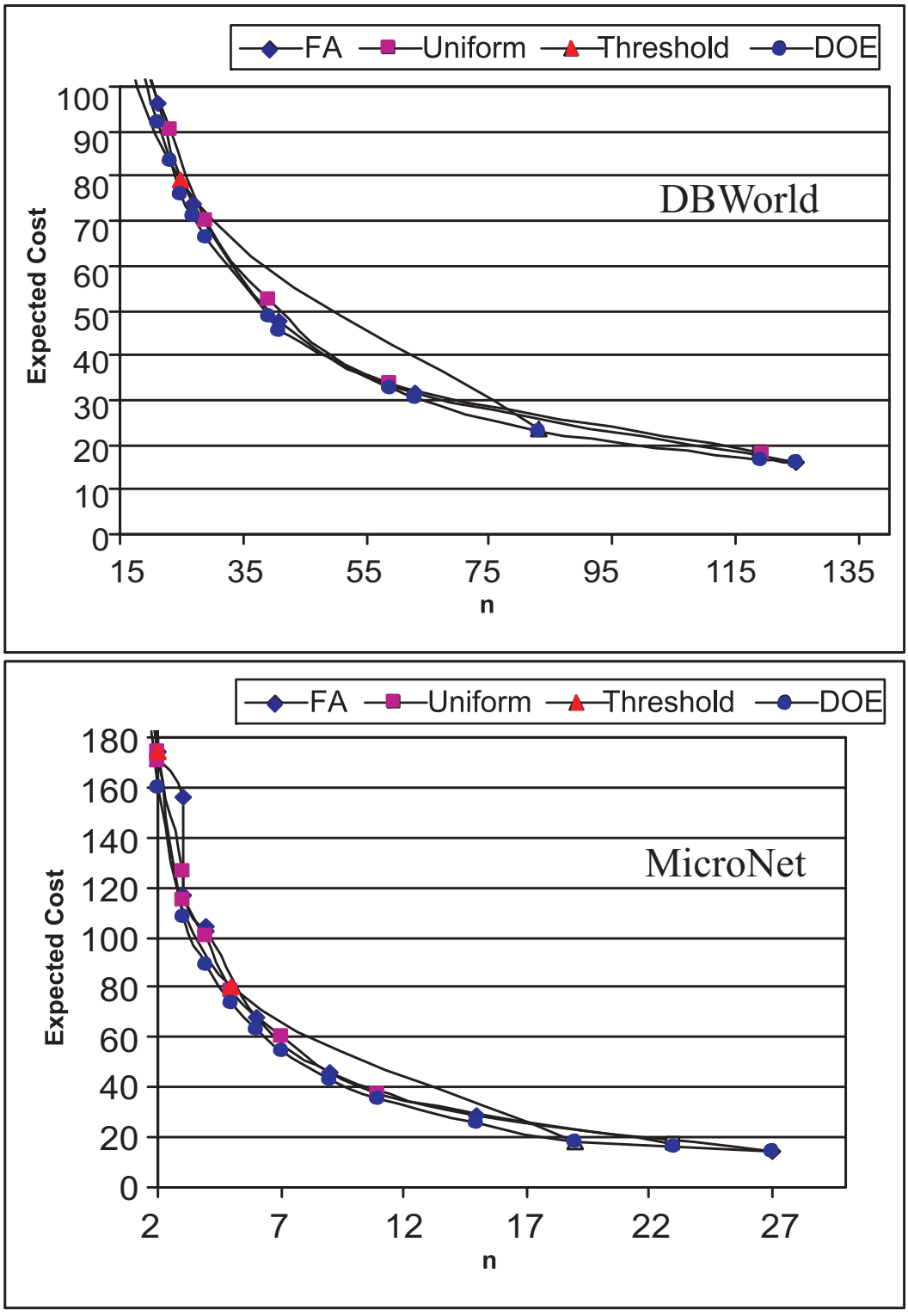


Figure 8: Comparison of the expected cost of the four policies, after improvement.

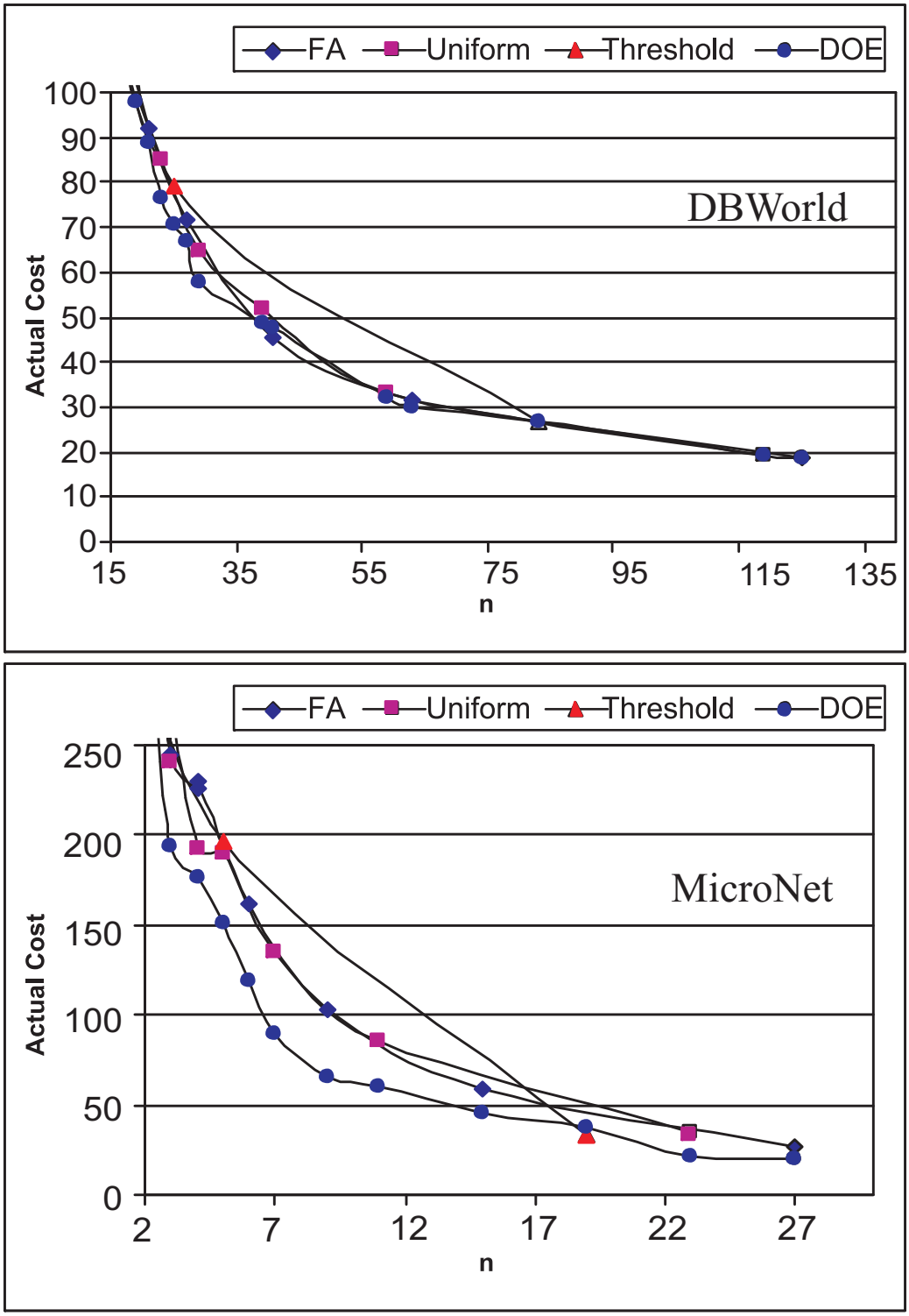


Figure 9: Comparison of the actual cost of the four policies, after improvement.

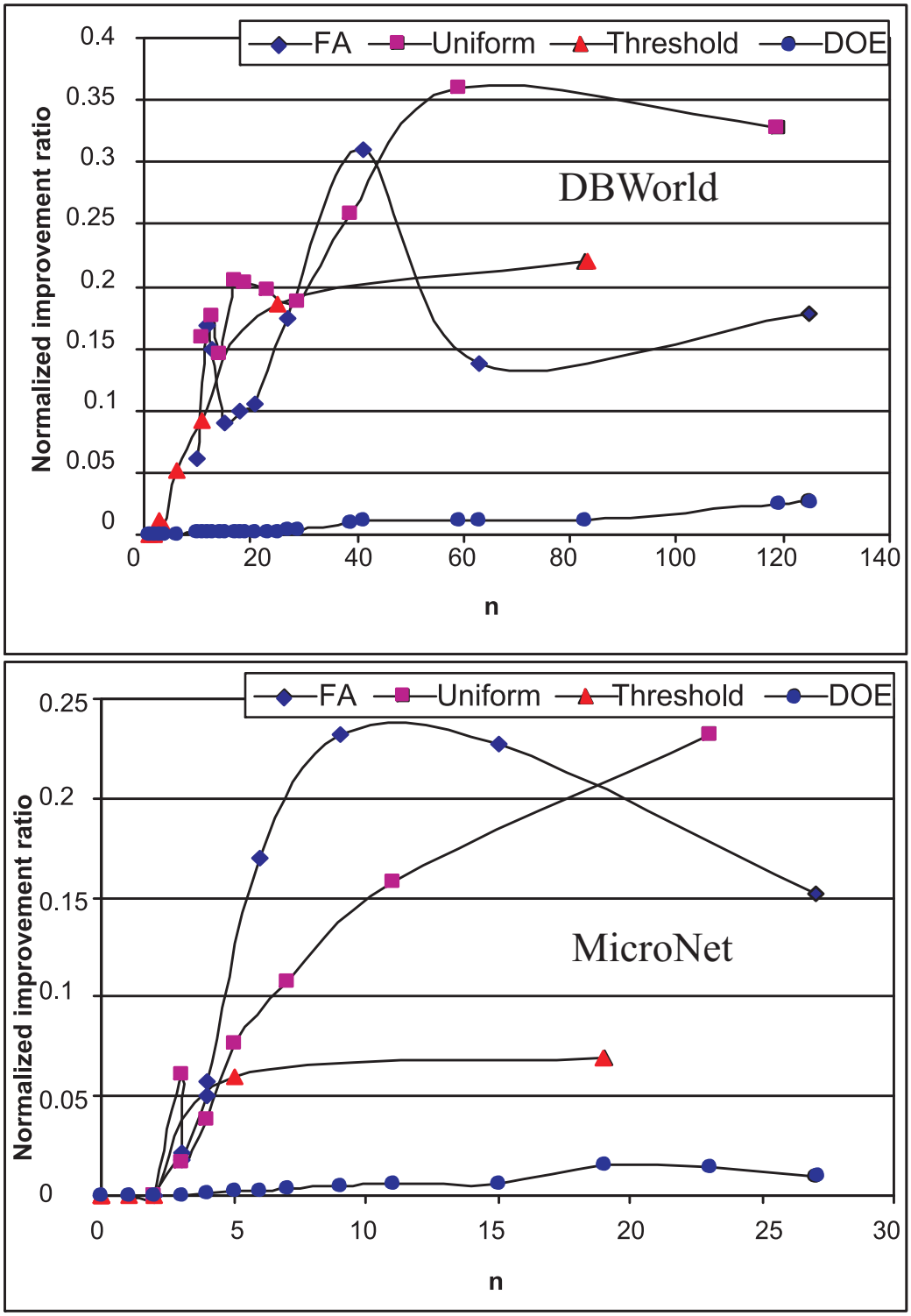


Figure 10: Comparison of the improvement in expected cost obtained by Algorithm 10.

bigger n values, probably due to the greater degrees of freedom available to the improver. It is somewhat surprising that the other algorithms show less monotone behavior, which may reflect some combination of differences in the quality of the initial schedule and the improver getting "stuck" near inferior local minima.

For our final experiment, we measured the number of μ-move iterations performed by Algorithm 10 until it terminated with $\mu < \mu_*$. For the DBWorld dataset, Figure 11 provides a comparative illustration on a logarithmic scale (top) and linear scale (bottom), to clearly show relative performance for both small and large n. The results for the MicroNet dataset are similar. For any given n, seeding the improver with DOE gives the fewest iterations, and, based on previous experiments, produces a lower expected cost than the other policies. All policies, including DOE, have their number of iterations increasing roughly exponentially as n grows, but DOE shows the smoothest trend.

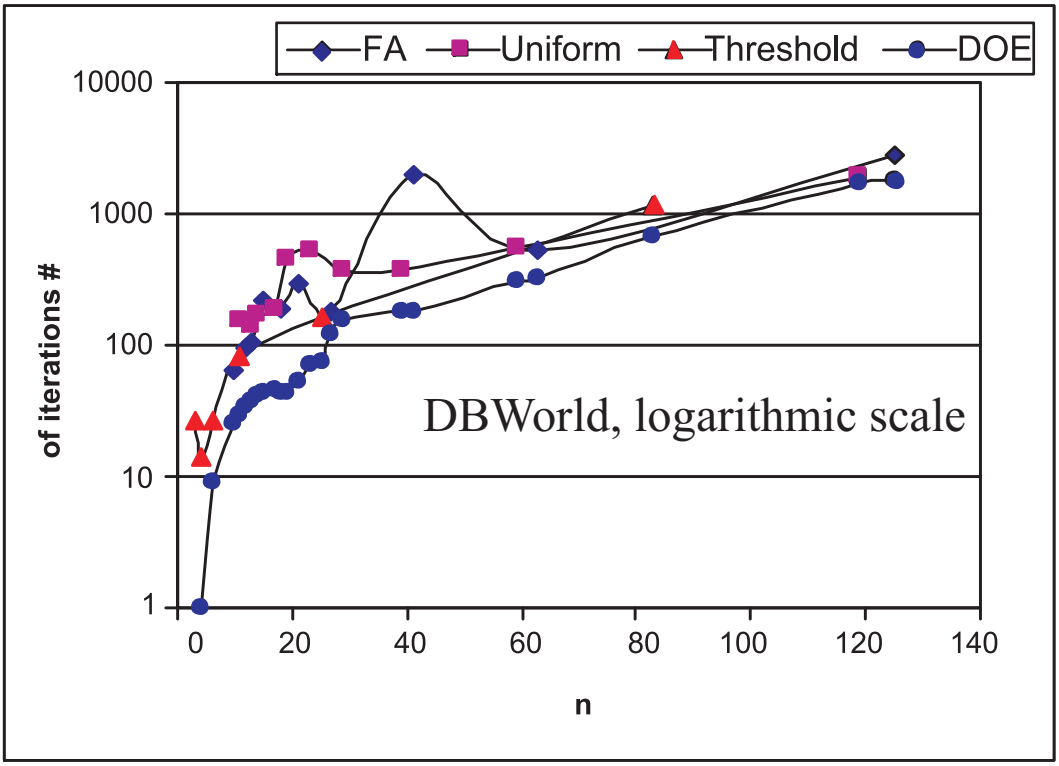
6 Related Work in the Information Systems Literature

Our research here may be classified as pull-based, in that the server is being probed periodically to refresh the client's local replica. Both push-based methods — *e.g.*, [2, 17] — and pull-based methods — *e.g.*, TTL [9] with its extensions [6, 1] — are common in applications such as Web caching and publish/subscribe, along with profile-driven cache management [4] and cache synchronization [5, 3]. Combined push-pull methods also exist [7].

Methods for optimizing Web information monitoring have recently been presented in [14, 13]. In particular, [14] presents a resource allocation problem aimed at minimizing the weighted importance of changes that are not reported to the user. The optimization problem there, as well as in [13], is subject to a system constraint of at most C probes per unit time, over all resources, which is similar but not identical to our politeness constraint. The work in [13] extends that of [14], both in terms of the cost function, and by providing an online algorithm (as opposed to the offline algorithm of [14]) which is a 2-approximate of an optimal schedule.

The algorithms presented in this paper tackle an optimization problem in the same context. However, our constraints of n probes over the period $[0, T]$ and separation by at least $\delta \geq 0$ are slightly different from the constraint in [13]. Our approach uses a stochastic update model, which is not presented explicitly in [14, 13]. Also, our formulation is parameterized by the user importance function $a(\cdot)$, generalizing the weighting method in [14, 13], and permitting a higher level of customization in the schedule. All the algorithms proposed here fall into the offline category — once one has estimated $\lambda(\cdot)$ from the training set, they treat the arrival models as fixed, and do not attempt to update it or gather additional data. Nevertheless, as our empirical analysis shows, the algorithms yields good results even with the use of real-world traces that may not fully conform to the update model assumptions. We also note that if a Poisson update process is really an accurate model, an online algorithm should not outperform an offline one, since the arrival process is memoryless.

Several works, including [5, 8, 11], have suggested using a Poisson model to model Web resource updates. A preliminary attempt to describe the time dependency of updates in the



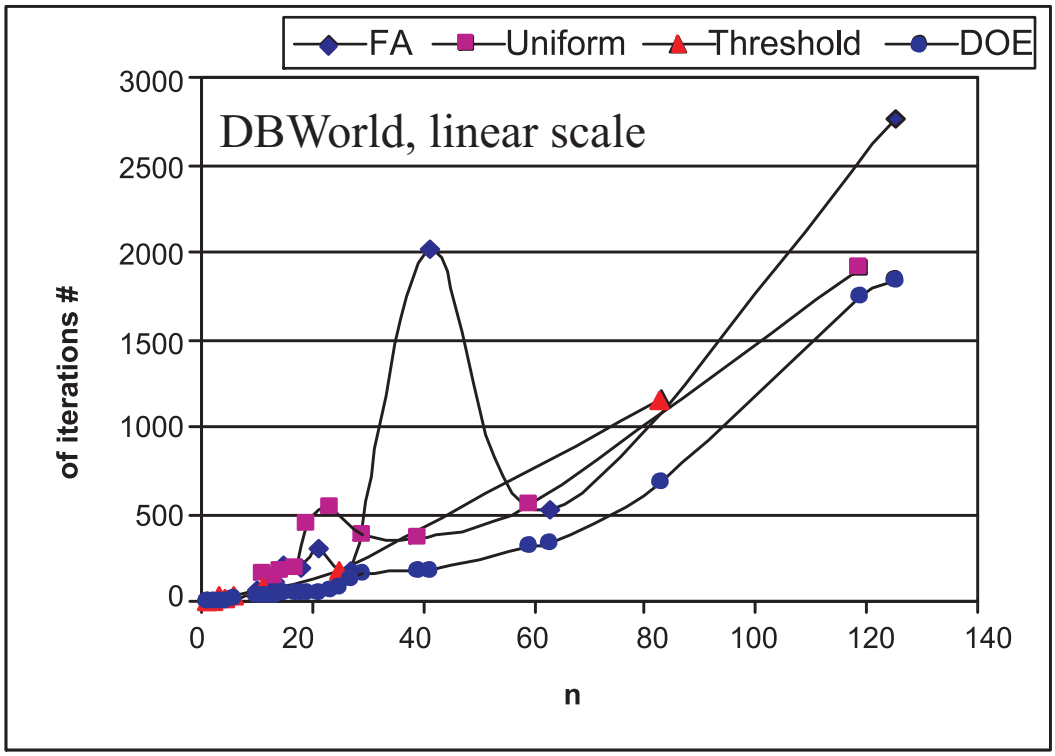


Figure 11: Comparison of the number of μ-move improver iterations.

context of Web management was given in [5], which suggests a simple homogeneous Poisson process. In addition, the work in [5] supposes that probes are performed at evenly-spaced time intervals. In the context of the cost model we have used, such uniform policies are suboptimal.

One way to capture varying update intensity, proposed in [11], is to use a shifting time window. In [11], the arrival rate λ is updated in an online manner based on the number of updates observed in the last K time units, K being a tunable parameter. This approach has the advantage of being able to capture update intensity variations not predictable from historical patterns, but has the disadvantage of being error-prone during predictable "shoulder" periods when λ either sharply increases or decreases. The probe scheduling technique suggested in [11] is identical to the first arrival policy suggested in [8] and tested here. Again, within the context of this paper's cost model, our empirical analysis indicates that first arrival policies are suboptimal.

Finally, in relation to [8], we use an equivalent cost model, but Algorithms 1 and 10 exploit it more fully than the threshold and first arrival policies proposed there. Thus, we obtain better policies with respect to the same update and cost models.

7 Conclusion

We have presented what we believe to be a reasonable model for measuring obsolescence cost when monitoring an information source, using an importance function $a(\cdot)$, and based on the model proposed for relational databases in [8]. Then, in the main novelty of this paper, we developed algorithms that combine this model with information about time variations in update intensity — via the function $\lambda(\cdot)$ — to construct probing schedules. As one would expect, these algorithms produce better schedules, measured in both an expected and actual sense, than previously-proposed scheduling heuristics which do not make full use of the obsolescence cost and/or update model.

Although an exact solution of the optimization problem (7) we have formulated is very difficult, our dynamic programming algorithm exactly optimizes an approximate, discretized form (9) of the problem, with an additive error bound of $O(1/N)$. Our second algorithm can then fine-tune the resulting schedule. Our computational experiments show that good schedules do appear to result from this procedure. Furthermore, it appears that our improver algorithm can be very beneficial when applied to other initial schedules. In this case, however, there is no $O(1/N)$ guarantee, and the improver may sometimes take significantly more iterations than when applied to the near-optimal schedules generated by our dynamic programming method.

The optimization problems and algorithms presented in this paper concern only the monitoring of a single resource. If the main constraints are politeness to the servers, the same techniques could be used to monitor *multiple* resources by simply constructing an independent probe schedule for each server. However, if there are significant constraints on the client's network bandwidth or ability to process updates, such an approach might not be viable for large numbers of servers. Then, one might have to formulate a more complicated

optimization problem describing a combined probing schedule for all resources. This problem might have to take into account client processing and network capacity, network proximity and performance, varying user importance for data from various servers, update intensity variations at each server, and perhaps even behavior correlations between servers. This much more complicated problem is a possible topic for future research.

References

[1] L. Bright and L. Raschid. Using latency-recency profiles for data delivery on the Web. In *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 550–561, Hong Kong, China, August 2002.

[2] P. Cao and C. Liu. Maintaining strong cache consistency in the World Wide Web. *IEEE Transactions on Computers*, 47(4):445–457, 1998.

[3] D. Carney, S. Lee, and S. Zdonik. Scalable application-aware data freshening. *Proceedings of the IEEE CS International Conference on Data Engineering*, pages 481–492, March 2003.

[4] M. Cherniack, E. Galvez, M. Franklin, and S. Zdonik. Profile-driven cache management. In *Proceedings of the IEEE CS International Conference on Data Engineering*, pages 645–656, Bangalore, India, March 2003.

[5] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, pages 117–128, Dallas, Texas, May 2000.

[6] E. Cohen and H. Kaplan. Refreshment policies for Web content caches. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM)*, pages 1398–1406, Anchorage, Alaska, April 2001.

[7] P. Deolasee, A. Katkar, P. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Disseminating dynamic Web data. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 265–274, Hong Kong, China, May 2001.

[8] A. Gal and J. Eckstein. Managing periodically updated data in relational databases: A stochastic modeling approach. *Journal of the ACM*, 48(6):1141–1183, 2001.

[9] J. Gwertzman and M. Seltzer. World Wide Web cache consistency. In *Proceedings of the USENIX Annual Technical Conference*, pages 141–152, San Diego, January 1996.

[10] T.G. Kolda, R.M. Lewis, and V. Torczon. Optimization by direct search: new perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[11] J.-J. Lee, K.-Y. Whang, B. S. Lee, and J.-W. Chang. An update-risk based approach to TTL estimation in web caching. In *Proceedings of the Conference on Web Information Systems Engineering (WISE)*, pages 21–29, Singapore, December 2002.

[12] J.F. Naughton, D.J. DeWitt, D. Maier, A. Aboulnaga, J. Chen, L. Galanis, J. Kang, R. Krishnamurthy, Q. Luo, N. Prakash, R. Ramamurthy, J. Shanmugasundaram, F. Tian, K. Tufte, S. Viglas, Y. Wang, C. Zhang, B. Jackson, A. Gupta, and R. Chen. The Niagara Internet query system. *IEEE Data Engineering Bulletin*, 24:27–33, 2001.

[13] S. Pandey, K. Dhamdhere, and C. Olston. WIC: A general-purpose algorithm for monitoring web information sources. In *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 360–371, Toronto, ON, Canada, September 2004.

[14] S. Pandey, K. Ramamritham, and S. Chakrabarti. Monitoring the dynamic web to respond to continuous queries. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 659–668, Budapest, Hungary, May 2003.

[15] S. Ross. *Introduction to Probability Models.* Academic Press, 1980.

[16] H.M. Taylor and S. Karlin. *An Introduction to Stochastic Modeling.* Academic Press, 1994.

[17] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering server-driven consistency for large scale dynamic web services. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 45–57, Hong Kong, China, May 2001.