OPERATIONS RESEARCH
Vol. 61, No. 1, January–February 2013, pp. 32–44
ISSN 0030-364X (print) | ISSN 1526-5463 (online)

http://dx.doi.org/10.1287/opre.1120.1148

When Is the Right Time to Refresh Knowledge Discovered from Data?

Xiao Fang, Olivia R. Liu Sheng
Department of Operations and Information Systems, David Eccles School of Business, University of Utah, Salt Lake City, Utah 84112
{xiao.fang@business.utah.edu, olivia.sheng@business.utah.edu}

Paulo Goes
Department of Management Information Systems, Eller College of Management, University of Arizona, Tucson, Arizona 85721,
pgoes@eller.arizona.edu

Knowledge discovery in databases (KDD) techniques have been extensively employed to extract knowledge from massive data stores to support decision making in a wide range of critical applications. Maintaining the currency of discovered knowledge over evolving data sources is a fundamental challenge faced by all KDD applications. This paper addresses the challenge from the perspective of deciding the right times to refresh knowledge. We define the knowledge-refreshing problem and model it as a Markov decision process. Based on the identified properties of the Markov decision process model, we establish that the optimal knowledge-refreshing policy is monotonically increasing in the system state within every appropriate partition of the state space. We further show that the problem of searching for the optimal knowledge-refreshing policy can be reduced to the problem of finding the optimal thresholds and propose a method for computing the optimal knowledge-refreshing policy. The effectiveness and the robustness of the computed optimal knowledge-refreshing policy are examined through extensive empirical studies addressing a real-world knowledge-refreshing problem. Our method can be applied to refresh knowledge for KDD applications that employ major data-mining models.

Subject classifications: data mining; knowledge discovery in databases; knowledge refreshing; Markov decision process.
Area of review: Games, Information, and Networks.
History: Received February 2011; revisions received December 2011, July 2012; accepted December 2012. Published online in *Articles in Advance* February 8, 2013.

1. Introduction

Knowledge discovery in databases (KDD) provides organizations with essential techniques to extract knowledge from massive data stores to support organizational decision making (Fayyad et al. 1996). Recent years have seen an explosion of applications that take the KDD paradigm to new levels of intensity and reach (e.g., Cooper and Giuffrida 2000, Sarkar and Sriram 2001, Albert et al. 2004, Rebbapragada et al. 2010). Massive amounts of data in different formats are now available to organizations both internally and externally. Every computer, smart phone, or network device is able to function as a data sensor to capture consumer behavior data, and business transaction data, social network data, environmental and societal data, which can be used to generate important knowledge related to marketing, operation, disease treatment and prevention, public health, security, and many other applications. KDD is the core component in the generation of the knowledge that supports advanced decision making in all of these important applications.

Most of the previous KDD research focuses on issues such as efficiency improvement of the KDD process (e.g., designing more computationally efficient data-mining algorithms) or innovative KDD applications, assuming that data are static. However, for real-world KDD applications, new data are continuously accumulated and data are dynamic in reality. Newly added data could bring in new knowledge and invalidate part or even all of the earlier discovered knowledge. As a result, knowledge discovered using KDD becomes obsolete over time. To support effective decision making, knowledge discovered using KDD needs to be kept current with its dynamic data source (Cooper and Giuffrida 2000). Consider the following emerging and increasingly important KDD applications in such areas as business intelligence, public health, and security.

- Knowledge about customer purchase patterns, which is discovered by running KDD on a database of customer purchase transactions, is essential in supporting various marketing decisions such as product assortment, store layout design, cross selling, and product promotion (Agrawal and Srikant 1994, Han and Kamber 2006). As new purchase transactions are continuously added to the database, some earlier discovered purchase patterns become invalid and some new purchase patterns emerge. Conceivably, making decisions based on obsolete knowledge is ineffective. For example, products promoted on the basis of invalid patterns may not be purchased, whereas neglecting emerging patterns may result in the loss of promotion opportunities.

• Surveillance systems are increasingly being deployed in defense operations, public health applications, cyber security, and several other domains of public administration. They entail the rapid collection of massive amounts of relevant data, the extraction of the knowledge about the underlying conditions, and the presence of anomalies or special conditions that may trigger events of interest. For example, disease surveillance systems collect syndromic data about infectious disease, infer knowledge from collected data, and generate early alarm of disease outbreaks (Chen and Zeng 2009). As more data are collected by such systems, keeping knowledge current is absolutely critical. Working with obsolete knowledge can be disastrous, for example, in the detection of how infectious diseases are spreading (Chen and Zeng 2009) or the detection of terrorism and cyberterrorism activities (Chen 2006).

Indeed, maintaining the currency of knowledge over evolving data sources is a fundamental challenge faced by all KDD applications, and it is ranked as one of the top three knowledge management issues by 2,073 knowledge management practitioners (King et al. 2002). A trivial solution is to run KDD whenever there is a change in data. However, such solution is neither practical, due to the high cost of running KDD (Manku and Motwani 2002), nor necessary, because it often results in no new knowledge discovered (Ganti et al. 1999). On the other hand, running KDD too seldom could lead to significant obsoleteness for the knowledge in hand. Therefore, it is critical to determine when to run KDD so as to optimize the trade-off between the obsoleteness of knowledge and the cost of running KDD. We address the fundamental challenge from this perspective and study the problem of deciding the right times to run KDD to refresh knowledge, namely, the knowledge-refreshing problem.

The rest of the paper is organized as follows. We start in §2 by reviewing related work and discuss research gaps tackled by this study. We define the knowledge-refreshing problem and model it as a Markov decision process in §3. The optimal knowledge-refreshing policy is analyzed, and a method for computing the optimal policy is proposed in §4. The method can be applied to refresh knowledge for KDD applications that employ major data-mining models such as association rule mining and classification. We empirically examine the effectiveness and the robustness of the proposed optimal knowledge-refreshing policy using a real-world knowledge-refreshing problem in §5. The paper is concluded with future directions of the study in §6. An electronic companion to this paper is available as part of the online version at http://dx.doi.org/10.1287/opre.1120.1148.

2. Literature Review

One stream of related research consists of prior work on incremental data mining and data stream mining. Incremental data mining studies how to efficiently maintain data mining results over evolving data sources. A number of incremental data-mining algorithms have been proposed for major data-mining models: incremental classification algorithms (e.g., Schlimmer and Fisher 1986, Utgoff 1989), incremental association rule mining algorithms (e.g., Cheung et al. 1996, Thomas et al. 1997), and incremental clustering algorithms (e.g., Can 1993). Incremental data-mining algorithms maintain data-mining results by revising previously mined results rather than mining from scratch. Current data-intensive applications are based on huge-size data sources with rapid and continuous loading of large volumes of data, termed as data streams (Aggarwal 2007). It has been shown that incremental data-mining algorithms are not capable of processing data streams (Domingos and Hulten 2000). To address this challenge, algorithms for data stream mining have been proposed recently (Aggarwal 2007). Data-stream mining algorithms usually employ data reduction techniques, such as sampling, to maintain approximate data-mining results for data streams. Representative data-stream mining algorithms include the Hoeffding tree algorithm (Hulten et al. 2001) for classification, the lossy counting algorithm (Manku and Motwani 2002) for association rule mining, and the algorithm by Guha et al. (2003) for clustering. Our research differs from previous studies on incremental data mining and data stream mining in the following ways. First, this research studies when to run KDD to refresh knowledge, whereas prior studies investigate how to run KDD. In addition, this research studies the knowledge-refreshing problem from a managerial perspective—balancing the trade-off between the obsoleteness of knowledge and the cost of running KDD—whereas past studies aim at improving the computational efficiency of running KDD.

Our research is also related to prior studies that have developed optimal policies or heuristic approaches for database checkpoint (Chandy et al. 1975), database reorganization (Park et al. 1990), view materialization (Srivastava and Rotem 1988, Segev and Fang 1991, Adelberg et al. 1995), and data warehouse synchronization (Dey et al. 2006). These studies primarily research optimal fixed-interval policies, under which, for example, database checkpoints are set after every fixed interval (e.g., periodically) (Chandy et al. 1975). An optimal fixed-interval policy is optimal among all fixed-interval policies but generally not optimal among all policies that also allow a chosen action to occur at nonfixed intervals. Our proposed knowledge-refreshing policy is optimal among all knowledge-refreshing policies. In addition, the proposed model, analytical properties, as well as computation method, are developed in the context of knowledge refreshing and thus differ from those in the prior studies. Moreover, we empirically show that our proposed knowledge-refreshing policy substantially outperforms optimal fixed-interval policies by applying them to a real-world knowledge-refreshing problem. In our earlier work (Fang and Rachamadugu 2009), we propose a

deterministic model for the knowledge-refreshing problem. The key components of the knowledge-refreshing problem, such as data or request arrival, are stochastic in reality. To better understand and solve real-world knowledge-refreshing problems, the stochastic nature of the problem needs to be considered. Accordingly, we develop a stochastic model for the knowledge-refreshing problem, analyze properties characterizing the problem and its solution, and propose a solution method in this study. Recent work on diagnostic system maintenance has studied "the problem of determining the optimal amount of effort that should be exerted to maintain the system" (Bensoussan et al. 2009, p. 294), which is different from our research, which studies when to run KDD to refresh knowledge. In addition, Bensoussan et al. (2009) study expert systems and focus on a specific knowledge-based task—binary classification, whereas we study KDD, which supports a much broader range of tasks, including classification, clustering, and association. Consequently, optimal policies proposed by Bensoussan et al. (2009), which are effective for diagnostic system maintenance, are either inapplicable or ineffective for knowledge refreshing. Two types of maintenance models are considered in Bensoussan et al. (2009): continuous and discrete models. Optimal polices based on the continuous maintenance models, which are suitable for problems with low maintenance cost (Bensoussan et al. 2009), are inapplicable to the knowledge-refreshing problem because it is impractical to run KDD continuously to refresh knowledge due to the high cost of running KDD (Manku and Motwani 2002). The optimal periodical policy derived from the discrete maintenance model is essentially a fixed-interval policy, which is ineffective to the knowledge-refreshing problem as discussed above.

3. Knowledge Refreshing: Problem and Model

3.1. The Knowledge-Refreshing Problem

We first introduce knowledge loss, a key concept for the knowledge-refreshing problem. Knowledge loss refers to the phenomenon that knowledge discovered by a previous run of KDD becomes obsolete gradually as new data are continuously added after the KDD run. Due to incoming new data, (a) part or even all of the earlier discovered knowledge becomes invalid; and (b) new knowledge emerges, which was not captured by the previous run of KDD. To quantify knowledge loss based on these two factors, we represent knowledge discovered by KDD as a set. Set representation of knowledge is appropriate for knowledge discovered by popular and powerful KDD techniques that are based on major data-mining models. For example, knowledge discovered by association rule mining is either a finite set of large itemsets or a finite set of association rules (Agrawal and Srikant 1994); whereas knowledge discovered using major classification models consists of a finite set of classification rules (Quinlan 1993) or can be converted into a finite set of classification rules (Baesens et al. 2003). We thus measure knowledge loss l_t at time t as

$$l_t = 1 - \frac{|K_t \cap \hat{K}_t|}{|K_t \cup \hat{K}_t|}, \tag{1}$$

where K_t denotes the knowledge one has at time t, which was discovered by the latest KDD run before time t, $\hat{K}_t$ denotes the knowledge discovered if KDD *were* run at time t, and $|\bullet|$ denotes the cardinality of a set. $K_t \cap \hat{K}_t$ represents the valid knowledge one has at time t. $K_t \cup \hat{K}_t$ can be rewritten as $(K_t \backslash \hat{K}_t) \cup (K_t \cap \hat{K}_t) \cup (\hat{K}_t \backslash K_t)$, where $K_t \backslash \hat{K}_t$ represents the invalid knowledge one has at time t and $\hat{K}_t \backslash K_t$ represents the emerging new knowledge not captured by the latest KDD run.[1] l_t is within the range of $[0, 1]$. $l_t = 0$ if and only if $K_t = \hat{K}_t$, which indicates that all of the earlier discovered knowledge is still valid and no new knowledge emerges (i.e., no loss). On the other hand, $l_t = 1$ if and only if $K_t \cap \hat{K}_t = \phi$, which means that the earlier discovered knowledge is fully outdated (i.e., total loss). The higher the value of l_t, the more the knowledge loss is.

The following example illustrates Equation (1). Knowledge about products frequently purchased together is critical in supporting decisions on product promotion and recommendation (Linden et al. 2003). Let $K_t = \{\{a, b\}, \{c, d\}\}$ be the knowledge one has at time t, which reveals that products a and b are frequently purchased together, and also products c and d. Let $\hat{K}_t = \{\{a, b\}, \{e, f\}, \{g, h\}\}$, if KDD *were* run at time t. Applying (1), $l_t = 3/4$. In this example, some earlier discovered knowledge (i.e., $\{c, d\}$) becomes invalid at time t and emerging new knowledge (i.e., $\{e, f\}$ and $\{g, h\}$) is not captured by an earlier KDD run. Supporting decisions at time t using K_t incurs cost due to these invalid old knowledge and emerging new knowledge. For example, product promotion decisions based on the invalid old knowledge are ineffective and promoted products may not be purchased; on the other hand, promotion opportunities informed by the emerging new knowledge are lost.

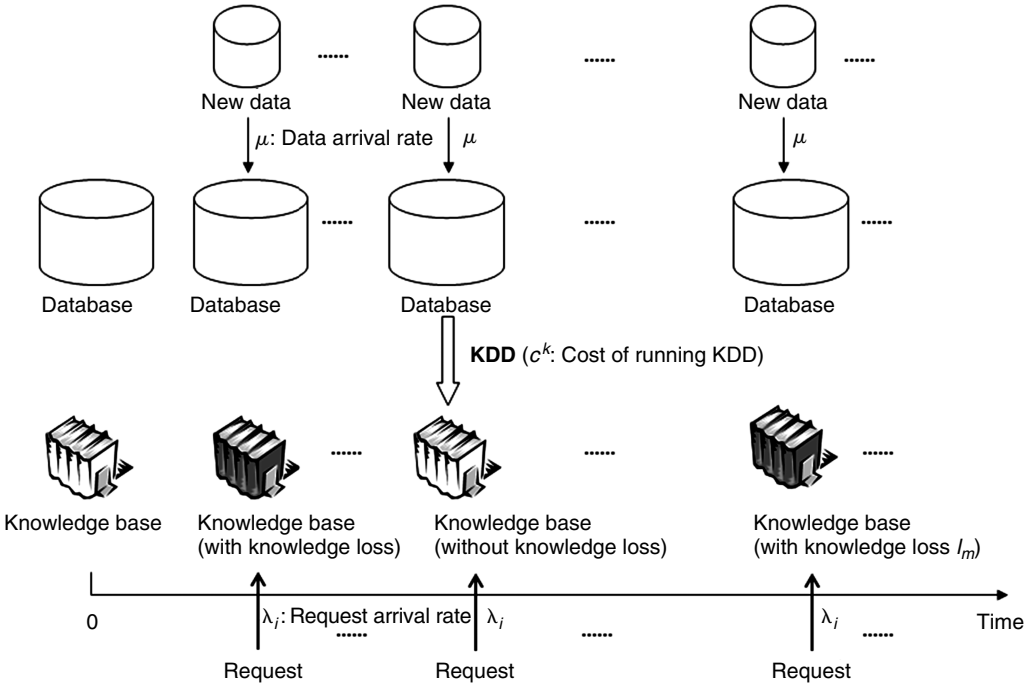
Let d_t denote the amount of new data accumulated by time t, where new data refer to data arrived after the latest KDD run. The knowledge loss phenomenon that earlier discovered knowledge becomes obsolete gradually as new data are added is modeled as

$$l_t = F(d_t), \tag{2}$$

where $F(\cdot)$ is an increasing function. For its general applicability, we analyze the optimal knowledge-refreshing policy using the general form $F(\cdot)$. We also show how to estimate a specific function form of $F(\cdot)$ for a real-world knowledge-refreshing problem in §5.2.

Figure 1 depicts the knowledge-refreshing problem. At the start of a time horizon, a knowledge base contains up-to-date knowledge discovered from a database.

Figure 1. The knowledge-refreshing problem.



The database evolves over time as new data are populated into it. Running KDD keeps the knowledge base current with the evolving database but incurs the cost of running KDD. Various decision support applications send requests to retrieve knowledge from the knowledge base. Without running KDD, a request could experience knowledge loss for the knowledge retrieved. The knowledge refreshing problem is to determine whether to run KDD upon the arrival of a request so that the total cost, including costs of running KDD and costs incurred by knowledge loss, over a time horizon is minimized.

3.2. The Model

The knowledge-refreshing problem is modeled as a Markov decision process. We assume that the arrival of new data follows a Poisson process with intensity μ. We model n different types of requests to a knowledge base, $\{r_i\}$, $i = 1, 2, \ldots, n$, $n \geqslant 1$. Requests are differentiated according to different types of decisions supported by them. Different types of decisions may occur at different frequencies and vary in importance. Consequently, different types of requests may have different arrival rates, and the same amount of knowledge loss may incur different costs for different types of requests. Each type of request is assumed to follow a Poisson process with intensity λ_i independently. Request arrivals are assumed to be independent of data arrivals. The impact of the Poisson arrival assumptions is analyzed empirically in §5.4. For the convenience of readers, notation used in this paper is summarized in Table 1.

The decision points are the moments when a request to a knowledge base arrives. Let M denote the number

Table 1. Notation.

μ = data arrival rate
r_i = type of request to a knowledge base, $i = 1, 2, \ldots, n$
λ_i = arrival rate of request type r_i
A = action space
a_m = action taken at time m, $a_m \in A$
d_m = amount of new data accumulated by time m

q_m = type of request at time m
s_m = system state at time m, $s_m = (q_m, d_m)$

S = state space, $s_m \in S$
Z^+ = set of all nonnegative integers

$y_{m-1,\,m}$ = amount of new data accumulated between time $m-1$ and time m

$P^{a_m}_{s_m s_{m+1}}$ = transition probability, see (4) and (5)
$c(s_m, a_m)$ = system cost at time m
l_m = knowledge loss at time m
c^k = cost of running KDD
ϕ = shape parameter; see (7)
c_{r_i} = cost of knowledge loss incurred if a type r_i request is served by a fully outdated knowledge base
δ_m = decision rule at time m, $a_m = \delta_m(s_m)$
$J(s_m)$ = minimum expected total system cost from time m, with state s_m, to the end of a time horizon; see (9)
S_i = subspace, $i = 1, 2, \ldots, n$; see Definition 3
$k_{m,\,i}$ = threshold at time m, $m = 1, 2, \ldots, M$, for request type r_i, $i = 1, 2, \ldots, n$; see Proposition 1
π^* = optimal knowledge refreshing policy

of decision points over a time horizon. The system time is labeled as $0, 1, 2, \ldots, M$, with 0 being the start of a time horizon, and m, $m = 1, 2, \ldots, M$, denoting the system time of decision point m or the system time when the mth request arrives. The action space is defined to be a binary set $A = \{0, 1\}$, where 0 denotes not running KDD and 1 denotes running KDD. Let $a_m \in A$ be the action chosen at time m. The system state at time m is represented by a vector $s_m = (q_m, d_m)$, where q_m denotes the type of request arrived at time m and d_m denotes the amount of new data accumulated by time m (e.g., measured in number of records/transactions). We have $q_m \in \{r_i\}$, $i = 1, 2, \ldots, n$ and $d_m \in \mathrm{Z}^+$, where Z^+ denotes the set of all nonnegative integers. The system state space S is the Cartesian product of $\{r_i\}$, $i = 1, 2, \ldots, n$ and Z^+, which is infinite.

The transition of d_m is controlled by system actions. If KDD is not run at time $m-1$, new data accumulated by time $m-1$ are carried forward to time m. On the other hand, if KDD is executed at time $m-1$, new data accumulated by time $m-1$ are processed and not treated as new data at time m. Therefore, the transition of d_m is expressed as

$$d_m = \begin{cases} d_{m-1} + y_{m-1,\,m} & \text{if } a_{m-1} = 0, \\ y_{m-1,\,m} & \text{if } a_{m-1} = 1, \end{cases} \tag{3}$$

where $1 \leqslant m \leqslant M$, d_{m-1} denotes the amount of new data accumulated by time $m-1$, and $y_{m-1,\,m}$ denotes the amount of new data accumulated between time $m-1$ and time m, $y_{m-1,\,m} \in \mathrm{Z}^+$. We set d_0 to 0, i.e., no new data at the start of a time horizon, because knowledge base is current with database at the start of a time horizon.

The system transition probability $P^{a_m}_{s_m s_{m+1}}$ from system state $s_m = (q_m, d_m)$ at time m to system state $s_{m+1} = (q_{m+1}, d_{m+1})$ at time $m+1$ under action $a_m = 0$ is given by

$$P\{q_{m+1} = r_i, d_{m+1} = k_2 \mid q_m = r_h, d_m = k_1, a_m = 0\} = \begin{cases} \dfrac{\mu^{k_2 - k_1}\lambda_i}{\left(\mu + \sum_{j=1}^{n}\lambda_j\right)^{k_2 - k_1 + 1}} & \text{if } k_2 \geqslant k_1, \\ 0 & \text{if } k_2 < k_1, \end{cases} \tag{4}$$

where i, $h = 1, 2, \ldots, n$ and k_1, $k_2 \in Z^+$.

The system transition probability $P^{a_m}_{s_m s_{m+1}}$ under action $a_m = 1$ is given by

$$P\{q_{m+1} = r_i, d_{m+1} = k_2 \mid q_m = r_h, d_m = k_1, a_m = 1\} = \frac{\mu^{k_2}\lambda_i}{\left(\mu + \sum_{j=1}^{n}\lambda_j\right)^{k_2+1}}, \tag{5}$$

where i, $h = 1, 2, \ldots, n$ and k_1, $k_2 \in Z^+$.

Derivation of Equations (4) and (5) is provided in Appendix A.

Let $c(s_m, a_m)$ be the system cost incurred at time m with system state s_m under action a_m. Without running KDD at time m, a request at time m suffers from knowledge loss and the cost of knowledge loss is incurred, denoted as c^l_m. On the other hand, running KDD at time m incurs the cost of running KDD, denoted as c^k_m, but saves the cost of knowledge loss, assuming that knowledge loss is reduced to zero instantaneously once KDD is run. Relaxation of the assumption is studied in Appendix E. $c(s_m, a_m)$ is expressed as

$$c(s_m, a_m) = \begin{cases} c^l_m & \text{if } a_m = 0, \\ c^k_m & \text{if } a_m = 1. \end{cases} \tag{6}$$

Let c_{r_i}, $i = 1, 2, \ldots, n$, denote the cost incurred if a type r_i request is served by a fully outdated knowledge base (i.e., knowledge loss equals 1). Estimation of c_{r_i} is application dependent. For example, product promotion decisions supported by a fully outdated knowledge base are ineffective. In this example, c_{r_i} might be estimated as the cost due to unrealized promotion objectives. c^l_m is modeled as

$$c^l_m = c_{q_m}(l_m)^{\phi}, \tag{7}$$

where $q_m \in \{r_i\}$, $i = 1, 2, \ldots, n$, l_m is knowledge loss at time m and $l_m = F(d_m)$ by (2). Shape parameter $\phi > 0$ governs how c^l_m increases as l_m increases and provides the flexibility to model the cost of knowledge loss in different situations. The value of ϕ is application dependent. The cost of running KDD consists of both the computation cost of running KDD and the personnel cost of running KDD (e.g., personnel cost of managing a KDD process). For real-world applications, the personnel cost of running KDD could be estimated using the number of man-hours involved in a KDD process multiplied by pay rates and the computation cost of running KDD could be estimated as the cost of using computing resources for a KDD process. We treat the cost of running KDD as a fixed cost c^k. The treatment is based on a popular procedure of running KDD over evolving data sources (Altiparmak et al. 2008, Gaber et al. 2005), according to which KDD is run on a fixed amount of most recent data.

A knowledge-refreshing policy π is defined as $\pi = (\delta_1, \delta_2, \ldots, \delta_M)$, where δ_m denotes the decision rule at time m such that $a_m = \delta_m(s_m)$, $m = 1, 2, \ldots, M$. The expected total system cost over a time horizon under π, EC_π, is expressed as

$$\mathrm{EC}_\pi = E\left\{\sum_{m=1}^{M}\left[\delta_m(s_m)c^k_m + (1 - \delta_m(s_m))c^l_m\right]\right\}. \tag{8}$$

The objective of the knowledge-refreshing problem is to find the optimal knowledge-refreshing policy π^* that minimizes EC_π.

4. Optimal Knowledge-Refreshing Policy

4.1. Analysis of Optimal Knowledge-Refreshing Policy

Finding the optimal knowledge-refreshing policy π^* by examining every system state at each decision point is computationally prohibitive, considering that the state space is infinite. In this subsection, we analyze key model components and characterize π^*. Based on the analysis in this subsection, an algorithm for computing π^* is proposed in §4.2.

The following concept and definitions are used in our analysis. Let $J(s_m)$ be the minimum expected total system cost from time m, with state $s_m \in S$, to the end of a time horizon, $m = 1, 2, \ldots, M$. By the principle of dynamic optimality (Bellman and Dreyfus 1962),

$$J(s_m) = \min_{a_m \in A} \left\{ c(s_m, a_m) + \sum_{s_{m+1} \in S} P^{a_m}_{s_m s_{m+1}} J(s_{m+1}) \right\}, \tag{9}$$

where $J(s_{m+1})$ is the minimum expected total system cost from time $m+1$, with state s_{m+1}, to the end of a time horizon and $J(s_{M+1}) = 0$ for all $s_{M+1} \in S$.

Definition 1 (Order of Actions). For two actions a^1, $a^2 \in A$, we define $a^1 > a^2$ if and only if $a^1 = 1$ (i.e., running KDD) and $a^2 = 0$ (i.e., not running KDD); $a^1 = a^2$ if and only if $a^1 = 1$ and $a^2 = 1$, or $a^1 = 0$ and $a^2 = 0$.

Definition 2 (Order of System States). For two system states $s^1, s^2 \in S$, we define $s^1 > s^2$ if and only if the cost of knowledge loss incurred at state s^1 is greater than the cost of knowledge loss incurred at state s^2. Similarly, we define $s^1 = s^2$ if and only if the cost of knowledge loss incurred at state s^1 is equal to the cost of knowledge loss incurred at state s^2. The cost of knowledge loss is given by Equation (7).

Definition 3 (Subspace). We define a subspace S_i, $i = 1, 2, \ldots, n$, as a set of system states $s = (q, d)$, where request type $q = r_i$ and amount of new data $d \in Z^+$. System states belonging to a subspace have the same request type but different amounts of new data. S_i is a partition of the state space S by request type, i.e., $S_i = \{s \mid s \in S, q = r_i\}$, and $S = \bigcup_{i=1}^{n} S_i$.

Definition 4 (Superadditive Function). A function $K(x, y)$ is superadditive if for $x_1 \geqslant x_2$ and $y_1 \geqslant y_2$ in its domain of definition, $K(x_1, y_1) + K(x_2, y_2) \geqslant K(x_1, y_2) + K(x_2, y_1)$ (Block et al. 1989).

We start our analysis by indentifying properties of key model components in Lemmas 1–4. Based on the identified properties, the optimal knowledge-refreshing policy is characterized in Theorem 1. The lemmas are primarily developed for the derivation of the theorem. In particular, Lemmas 1 and 2 establish the monotonicity property regarding the transition probability $P^{a_m}_{s_m s_{m+1}}$ and the minimum expected total system cost $J(s_m)$, respectively. Lemmas 3 and 4 establish the superadditive property regarding the cost function $c(s_m, a_m)$ and the transition probability $P^{a_m}_{s_m s_{m+1}}$, respectively.

Lemma 1. *Let $f(s_m)$ be a function of $s_m \in S_i$, where S_i is a subspace, $i = 1, 2, \ldots, n$. Given a constant system state $s^k \in S$, $f(s_m) = \sum_{s_{m+1} \geqslant s^k, s_{m+1} \in S_j} P^{a_m}_{s_m s_{m+1}}$, where S_j is a subspace, $j = 1, 2, \ldots, n$. For $m = 1, 2, \ldots, M$, $f(s_m)$ is monotonically increasing[2] in s_m for all $s^k \in S$ and $a_m \in A$.*

See Appendix B for the proof.

Lemma 1 implies that the probability that system state $s_{m+1} \in S_j$ at time $m+1$ exceeds certain state s^k is higher or unchanged when system state $s_m \in S_i$ at time m is greater, where system states are ordered according to Definition 2.

Lemma 2. *For $m = 1, 2, \ldots, M$, $J(s_m)$ is monotonically increasing in $s_m \in S_i$, where S_i is a subspace, $i = 1, 2, \ldots, n$.*

See Appendix B for the proof.

Lemma 2 means that the minimum expected total system cost $J(s_m)$ from time m to the end of a time horizon is higher or unchanged when system state $s_m \in S_i$ at time m is greater, where system states are ordered according to Definition 2. For $s_m \in S_i$, by Definitions 2 and 3, the increasing of s_m implies the increasing of d_m and vice versa because system states belonging to a subspace have same request type. Hence, Lemma 2 also indicates that, within each subspace, $J(s_m)$ is higher or unchanged when d_m is larger.

Lemma 3. *Let $h(s_m, a_m)$ be a function on $S_i \times A$ and $h(s_m, a_m) = -c(s_m, a_m)$, where S_i is a subspace, $i = 1, 2, \ldots, n$ and A is the action space. $h(s_m, a_m)$ is a superadditive function for $m = 1, 2, \ldots, M$.*

See Appendix B for the proof.

Lemma 4. *Let $g(s_m, a_m)$ be a function on $S_i \times A$, where S_i is a subspace, $i = 1, 2, \ldots, n$, and A is the action space. Given a constant system state $s^k \in S$, let $g(s_m, a_m) = -\sum_{s_{m+1} \geqslant s^k, s_{m+1} \in S_j} P^{a_m}_{s_m s_{m+1}}$, where S_j is a subspace, $j = 1, 2, \ldots, n$. For $m = 1, 2, \ldots, M$, $g(s_m, a_m)$ is a superadditive function for all $s^k \in S$.*

See Appendix B for the proof.

We derive the following theorem based on the lemmas.

Theorem 1. *The optimal knowledge-refreshing policy is monotonically increasing in the system state within each subspace.*

See Appendix B for the proof.

Theorem 1 means that optimal decision rule $\delta^*_m(s_m)$ of the optimal knowledge-refreshing policy is increasing in $s_m \in S_i$ for $m = 1, 2, \ldots, M$, where S_i is a subspace, $i = 1, 2, \ldots, n$. Recall that optimal action $a^*_m = \delta^*_m(s_m)$. Hence, optimal action a^*_m is increasing in $s_m \in S_i$ for $m = 1,$

$2, \ldots, M$, where actions are ordered according to Definition 1. The action space consists of only two actions: 0 denoting not running KDD and 1 denoting running KDD. Therefore, at every decision point, there exists a threshold system state within each subspace. If the current system state is less than the threshold state, optimal action is not running KDD; otherwise, it is optimal to run KDD, where system states are ordered according to Definition 2. Theorem 1 justifies that optimal knowledge-refreshing policy can be computed by finding the threshold state within each subspace at every decision point, which forms the basis of the computation method proposed in the next subsection.

4.2. Optimal Knowledge-Refreshing Policy: Computation Method and Properties

We derive the following proposition from Theorem 1.

PROPOSITION 1. *At time m, $m = 1, 2, \ldots, M$, for request type r_i, $i = 1, 2, \ldots, n$, there exists a threshold $k_{m,i} \in Z^+$ and optimal action a_m^* for system state $s_m = (q_m, d_m)$ is characterized as*

$$a_m^* = \begin{cases} 0 & \text{if } d_m < k_{m,i} \text{ and } q_m = r_i, \\ 1 & \text{if } d_m \geqslant k_{m,i} \text{ and } q_m = r_i. \end{cases} \tag{10}$$

See Appendix C for the proof.

According to Proposition 1, the optimal knowledge-refreshing policy π^* can be computed by finding $k_{m,i}$ for each request type r_i, $i = 1, 2, \ldots, n$, at time m, $m = 1, 2, \ldots, M$. By Proposition 1 and Equation (9), $k_{m,i}$ is equal to the minimum d_m satisfying $Q(s_m, a_m = 0) > Q(s_m, a_m = 1)$ and

$$Q(s_m, a_m) = c(s_m, a_m) + \sum_{s_{m+1} \in S} P_{s_m s_{m+1}}^{a_m} J(s_{m+1}), \tag{11}$$

where $s_m = (q_m, d_m)$ and $q_m = r_i$. In the rest of the subsection, we analyze $Q(s_m, a_m)$ in Propositions 2 and 3 and identify additional properties of the optimal knowledge-refreshing policy π^* in Propositions 4 and 5. An algorithm for computing π^* is developed based on these propositions.

To search for $k_{m,i}$, $Q(s_m, a_m)$ is critical. However, computing $Q(s_m, a_m)$ using (11) could be highly time consuming, except for $m = M$. One difficulty associated with the computation is the infinite state space S. The following proposition identifies a property of $J(s_m)$ and shows that $Q(s_m, a_m)$ can be computed in finite time by applying the property.

PROPOSITION 2. (a) *At time m, $m = 1, 2, \ldots, M$, $J(s_m)$ for all system states $s_m = (q_m, d_m)$ satisfying $q_m = r_i$ and $d_m \geqslant k_{m,i}$, $i = 1, 2, \ldots, n$, are the same.*

(b) *The time complexity of computing $Q(s_m, a_m)$ using (11) is $O(k_{m+1})$, where $k_{m+1} = \sum_{i=1}^{n} k_{m+1,i}$, $k_{m+1,i}$ is the threshold for request type r_i at time $m+1$, and $m = 1, 2, \ldots, M-1$.*

See Appendix C for the proof.

The following proposition analyzes relationships among different $Q(s_m, a_m)$. Applying the identified relationships, the time complexity of computing $Q(s_m, a_m)$ can be reduced from $O(k_{m+1})$ to as much as $O(1)$.

PROPOSITION 3. *We have the following properties about $Q(s_m, a_m)$, where $s_m = (q_m, d_m)$, $m = 1, 2, \ldots, M$.*

(a) *For any i, $j = 1, 2, \ldots, n$ and $d \in Z^+$,*

$$\begin{aligned} &Q(q_m = r_j, d_m = d, a_m = 0) \\ &\quad = Q(q_m = r_i, d_m = d, a_m = 0) + (c_{r_j} - c_{r_i})(F(d))^{\phi}. \end{aligned} \tag{12}$$

(b) *For any $i = 1, 2, \ldots, n$ and $d \in Z^+$,*

$$\begin{aligned} &Q(q_m = r_i, d_m = d+1, a_m = 0) \\ &\quad = \Bigg[Q(q_m = r_i, d_m = d, a_m = 0) - c_{r_i}(F(d))^{\phi} \\ &\qquad - \sum_{l=1}^{n} \frac{\lambda_l}{(\mu + \sum_{j=1}^{n} \lambda_j)} J(q_{m+1} = r_l, d_{m+1} = d) \Bigg] \\ &\qquad \cdot \left(1 + \frac{\sum_{j=1}^{n} \lambda_j}{\mu}\right) + c_{r_i}(F(d+1))^{\phi}. \end{aligned} \tag{13}$$

(c) *For any i, $j = 1, 2, \ldots, n$ and $d \in Z^+$,*

$$\begin{aligned} &Q(q_m = r_j, d_m = d, a_m = 1) \\ &\quad = Q(q_m = r_i, d_m = 0, a_m = 0) + c^k. \end{aligned} \tag{14}$$

See Appendix C for the proof.

By proposition 3, $Q(q_m = r_j, d_m = d, a_m = 0)$ can be calculated using (12) from readily computed $Q(q_m = r_i, d_m = d, a_m = 0)$ instead of (11). As a result, the time complexity is reduced from $O(k_{m+1})$ to $O(1)$. Similarly, $Q(q_m = r_i, d_m = d+1, a_m = 0)$ can be computed from $Q(q_m = r_i, d_m = d, a_m = 0)$ using (13) and $Q(q_m, d_m, a_m = 1)$ can be computed from $Q(q_m = r_i, d_m = 0, a_m = 0)$ using (14). Applying Proposition 3, we initially compute $Q(q_m = r_i, d_m = 0, a_m = 0)$ using (11), where r_i is a randomly picked request type. All other $Q(s_m, a_m)$ are then computed using Equations (12)–(14) instead of (11).

The optimal knowledge-refreshing policy π^* is represented using $k_{m,i}$, $m = 1, 2, \ldots, M$, $i = 1, 2, \ldots, n$, which generally changes over time (i.e., from time 1 to time M) and varies among different request types. It is therefore interesting to analyze π^* by studying (a) how $k_{m,i}$ changes over time in Proposition 4; and (b) how $k_{m,i}$ varies among different request types in Proposition 5.

PROPOSITION 4. *For a request type r_i, $i = 1, 2, \ldots, n$, and its threshold $k_{m,i}$ at time m, $m = 1, 2, \ldots, M$, $k_{m,i} \leqslant k_{M,i}$ for $m = 1, 2, \ldots, M-1$.*

See Appendix C for the proof.

Proposition 4 suggests that $k_{m,i}$ attains its maximum when $m = M$. By Proposition 1, Equations (11), (6), (7),

Figure 2. An algorithm for computing the optimal knowledge-refreshing policy.

```
calculate k_{M,i} for i = 1, 2, ..., n by solving (15);
for m = M − 1 to 1 step −1
  calculate Q(q_m = r_1, d_m = 0, a_m = 0) using (11);
  calculate Q(s_m, a_m = 1) using (14);
  for i = 1 to n step 1
    d_m = 0;
    while (d_m ⩾ 0)
      calculate Q(s_m, a_m = 0) using (12) or (13),
        where s_m = (q_m, d_m) and q_m = r_i;
      if (Q(s_m, a_m = 0) > Q(s_m, a_m = 1))
        J(s_m) = Q(s_m, a_m = 1);
        k_{m,i} = d_m;
        break;
      end if
      J(s_m) = Q(s_m, a_m = 0);
      d_m = d_m + 1;
    end while
  end for
end for
```

and $J(s_{M+1}) = 0$ for all $s_{M+1} \in S$, $k_{M,i}$, $i = 1, 2, \ldots, n$, equals the minimum d_M satisfying

$$c_{r_i}(F(d_M))^{\phi} > c^k. \tag{15}$$

If there exist finite-valued $k_{M,i}$, $i = 1, 2, \ldots, n$, by Proposition 4, $k_{m,i}$, $m = 1, 2, \ldots, M-1$, $i = 1, 2, \ldots, n$, should be finite and the optimal knowledge-refreshing policy is computed using the algorithm shown in Figure 2. Otherwise means that the cost of running KDD is no less than the cost of knowledge loss no matter how much amount of new data is accumulated. For this case, not running KDD is optimal. Based on the analysis in Propositions 1–4, an algorithm for computing the optimal knowledge-refreshing policy π^* is given in Figure 2. As shown in the figure, $k_{M,i}$ is firstly calculated for $i = 1, 2, \ldots, n$ by solving (15). The algorithm then computes $k_{m,i}$ for each request type r_i, $i = 1, 2, \ldots, n$, from $m = M-1$ to $m = 1$. Given $k_{M,i}$ is finite, $k_{m,i}$ at $m = 1, 2, \ldots, M-1$, which is no more than $k_{M,i}$ by Proposition 4, should be finite. Hence, the algorithm terminates in finite time. Both the storage and the time complexity of the algorithm is $O(\kappa)$, where $\kappa = \sum_{m=1}^{M}\sum_{i=1}^{n} k_{m,i}$ and $k_{m,i}$ is the threshold for request type r_i at time m.[3]

The optimal expected total system cost over the time horizon, EC_{π^*}, is given by

$$\mathrm{EC}_{\pi^*} = \sum_{s_1 \in S} P_{s_1} J(s_1), \tag{16}$$

where P_{s_1} is the probability of system state being s_1 at time 1. P_{s_1} is given by

$$P_{s_1} = P\{q_1 = r_i, d_1 = k\} \quad \text{let } s_1 = (q_1, d_1)$$
$$= P\{q_1 = r_i, y_{0,1} = k\} = \frac{\mu^k \lambda_i}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k+1}}, \quad \text{by (A1) in Appendix A,}$$

where $i = 1, 2, \ldots, n$ and $k \in Z^+$.

A request type is described by two system parameters: the arrival rate of the request type λ_i and the cost of knowledge loss incurred by the request type c_{r_i}. The following proposition analyzes how λ_i and c_{r_i} impact $k_{m,i}$.

Proposition 5. *For request types r_i and r_j, $i, j = 1, 2, \ldots, n$ and $i \neq j$, their respective arrival rate λ_i and λ_j, their respectively incurred cost of knowledge loss c_{r_i} and c_{r_j}, and their respective threshold $k_{m,i}$ and $k_{m,j}$ at time m, $m = 1, 2, \ldots, M$, we have,*

(a) *if $c_{r_i} > c_{r_j}$, $k_{m,i} \leqslant k_{m,j}$ for $m = 1, 2, \ldots, M$, regardless of their arrival rates*;

(b) *if $c_{r_i} = c_{r_j}$, $k_{m,i} = k_{m,j}$ for $m = 1, 2, \ldots, M$, regardless of their arrival rates*; *and*

(c) *if $c_{r_i} = c_{r_j}$, $J(q_m = r_i, d_m = d) = J(q_m = r_j, d_m = d)$ for $m = 1, 2, \ldots, M$ and $d \in Z^+$, regardless of their arrival rates.*

See Appendix C for the proof.

By Proposition 5, if several request types incur the same cost of knowledge loss, regardless of their arrival rates, we only need to compute $k_{m,i}$ and $J(s_m)$ for one of these request types. The computed $k_{m,i}$ and $J(s_m)$ can be applied to the rest of the request types, which reduces the time of computing π^*.

5. Empirical Analysis

We examine the effectiveness and the robustness of the optimal knowledge-refreshing policy π^* using a real-world knowledge-refreshing problem. Model parameters are calibrated in §5.1 and knowledge loss is estimated in §5.2. The effectiveness and the robustness of π^* are analyzed in §§5.3 and 5.4, respectively. We also study the relaxation of the model assumption regarding KDD run in Appendix E.

5.1. Calibration of Model Parameters

We study a real-world knowledge-refreshing problem in the Department of Household Items at a leading U.S. online retailer, namely, the retail knowledge-refreshing problem. In this department, knowledge about products frequently purchased together is used to support product promotions in an email campaign, and online product recommendations. The knowledge is discovered by mining large itemsets (Agrawal and Srikant 1994) from a database of purchase transactions. The retail knowledge-refreshing problem is to determine appropriate times to run KDD so that the total cost, including costs of running KDD and costs of knowledge loss, is minimized. We set parameter values based on the transaction database and according to our interviews with domain experts in the department.

The arrival rate μ of data was set to 386, which was the average number of transactions arrived daily according to the transaction database. In this problem, one time unit corresponded to a day. Two different types of knowledge requests, r_1 and r_2, were considered. Type r_1 requested knowledge supporting product promotion decisions in an

email campaign, which was generally launched every five days. Therefore, we set the arrival rate λ_1 of type r_1 knowledge requests to 1/5. Calculated from the transaction database, the average revenue generated by an email campaign was \$4,915.41. The basic setup of c_{r_1} (i.e., cost of knowledge loss incurred if a type r_1 request is served by a fully outdated knowledge base) was \$4,915.41 because products promoted based on a fully outdated knowledge base were not attractive to customers and the revenue could not be realized. To ensure the reliability of our empirical analysis results, we conducted experiments using the basic setup of c_{r_1} as well as its variations (i.e., $\rho \times 4{,}915.41$, $\rho > 0$).[4] Type r_2 requested knowledge used by an intelligent system for online product recommendations. The intelligent system was generally maintained every seven days and requested knowledge from the knowledge base during the maintenance. Hence, we set the arrival rate λ_2 of type r_2 knowledge requests to 1/7. Calculated from the transaction database, the average revenue generated through online product recommendations over a seven-day period was \$10,426.63. The basic setup of c_{r_2} was \$10,426.63 because products recommended using knowledge from a fully outdated knowledge base were not attractive and the revenue could not be realized. Similarly, we conducted experiments using the basic setup of c_{r_2} as well as its variations (i.e., $\rho \times 10{,}426.63$, $\rho > 0$). The cost of running KDD consisted of the personnel cost of running KDD and the computation cost of running KDD. According to domain experts, a typical KDD project in the department required 24 man-hours specializing in data mining, database, and programming with an average hourly rate of \$35, and 16 man-hours specializing in marketing and sales with an average hourly rate of \$37.5. The personnel cost of running KDD was set to \$1,440 (i.e., $24 \times 35 + 16 \times 37.5$). According to domain experts, the computation cost of running KDD was set to \$343, which was the operating expense of using hardware and software (e.g., KDD software) for KDD. The cost of running KDD c^k was set to \$1,783 (i.e., $1{,}440 + 343$). We conducted experiments using the basic setup of c^k as well as its variations (i.e., $\vartheta \times 1{,}783$, $\vartheta > 0$). Table 2 summarizes the parameter values.

5.2. Estimation of Knowledge Loss

The exact value of knowledge loss l_t at time t is usually unknown because $\hat{K}_t$ in (1) is generally unavailable at time t. However, the amount of new data d_t is readily available at time t. We could calculate l_t from d_t using (2), which further requires us to estimate the knowledge loss function $F(\cdot)$. The following experiment was designed to estimate $F(\cdot)$. In the experiment, knowledge about products frequently purchased together was initially discovered by running KDD on a data set of n_b purchase transactions, namely, the base knowledge. Given the base knowledge as the latest knowledge one had, we then studied knowledge loss through a three-step procedure. In step one of the procedure, to emulate incoming new data, n_i transactions, which were conducted chronologically later than the transactions in the data set, were added to the data set. Next, knowledge was discovered by running KDD on the updated data set, namely, the updated knowledge.[5] Finally, the value of knowledge loss was calculated using (1), where $\hat{K}_t$ in (1) corresponded to the updated knowledge and K_t in (1) corresponded to the base knowledge. We now obtained (calculated) one knowledge loss value and its corresponding amount of incoming new data. Continuing the three-step procedure by adding another set of n_i transactions, we obtained a set of knowledge loss values and their corresponding amount of incoming new data (i.e., n_i, $2n_i$, and $3n_i$ etc.).

Table 2. Parameter values for the retail knowledge-refreshing problem.

Parameter	Value
μ	386
λ_1	1/5
λ_2	1/7
c_{r_1} (\$)	4,915.41
c_{r_2} (\$)	10,426.63
c^k (\$)	1,783

We ran an experiment with a data set of $n_b = 11{,}580$ transactions (i.e., 30-day of transactions conducted at the Department of Household Items) and $n_i = 50$. Knowledge loss values obtained in this experiment were plotted against their corresponding amount of new data. The plotted knowledge loss curve suggested that knowledge loss for the retail knowledge-refreshing problem could be estimated using a two-parameter Weibull function (Huet et al. 1996). That is

$$l_t = 1 - e^{-(d_t/\alpha)^\beta} \quad \alpha > 0,\ \beta > 0. \tag{17}$$

Applying nonlinear regression to the knowledge loss data collected in the experiment, we obtained estimates of $\alpha = 10{,}505.1$ and $\beta = 0.53$. The fit between the Weibull estimating function and the knowledge loss data was statistically significant ($p < 0.0001$), and the function accounted for 99% of the variance ($R^2 = 0.99$). Additional experiments were conducted with $n_i = 25, 75, 100$, and 125, respectively. For each of the additional experiments, the fit between the Weibull estimating function and the knowledge loss data was statistically significant ($p < 0.0001$, $R^2 = 0.99$). Table 3 reports the estimates of α and β for all the experiments. The experimental results suggest that the Weibull function estimates knowledge loss for the retail knowledge-refreshing problem very well and the estimates of α and β are generally insensitive to the choice of n_i. For the experiments reported in the rest subsections, knowledge loss for the retail knowledge-refreshing problem was estimated using (17) with $\alpha = 10{,}505.1$ and $\beta = 0.53$. We also show how to quantify and estimate knowledge loss for the situation where the factors determining knowledge loss are weighted differently in Appendix D.

Table 3. Estimates of α and β.

n_i	Estimate of α	Estimate of β
25	10,488.1	0.53
	(196.5)	(0.007)
50	10,505.1	0.53
	(284.1)	(0.011)
75	10,540.3	0.53
	(343.1)	(0.013)
100	10,436.3	0.53
	(396.3)	(0.016)
125	10,670.1	0.53
	(426.1)	(0.016)

Note. Standard error of an estimate is listed in the parentheses below the estimate.

5.3. Effectiveness Analysis

The effectiveness of π^* was examined by analyzing cost savings generated by π^* over optimal fixed-interval policies for knowledge refreshing. In particular, we considered the following optimal fixed-interval policies: (a) executing KDD after the passage of a fixed chronological time interval (i.e., periodical policy), (b) executing KDD after the passage of a fixed number of requests (e.g., running KDD after every three requests), and (c) executing KDD after the accumulation of a fixed amount of new data. We chose optimal fixed-interval policies as benchmark policies based on the following considerations. First, fixed-interval policies such as the periodical policy are popularly applied in practice. Second, optimal fixed-interval policies have been well studied in the literature (e.g., Chandy et al. 1975).

We conducted experiments with parameter values listed in Table 2. We set M to 125 for the retail knowledge-refreshing problem, which is the typical number of knowledge requests arrived in a year. Let C_{π^*} be the total system cost resulting from implementing π^*, C_{p^*} be the total system cost resulting from implementing p^* (i.e., optimal periodical policy), C_{r^*} be the total system cost resulting from implementing r^* (i.e., optimal fixed-interval policy based on the number of requests), and C_{d^*} be the total system cost resulting from implementing d^* (i.e., optimal fixed-interval policy based on the amount of accumulated new data).[6] The cost differences between π^* and the benchmark optimal fixed-interval policies are measured as $\psi_1 = (C_{p^*} - C_{\pi^*})/C_{p^*}$, $\psi_2 = (C_{r^*} - C_{\pi^*})/C_{r^*}$, and $\psi_3 = (C_{d^*} - C_{\pi^*})/C_{d^*}$, respectively. The shape parameter ϕ in (7) was set to 1 in one experiment and was varied in other experiments. As shown in Table 4, π^* outperforms benchmark optimal fixed-interval policies p^*, r^*, and d^* with cost savings ranging from 28.7% to 61.3%, from 6.1% to 17.0%, and from 25.9% to 60.2%, respectively.[7] Across the experiments, the time of computing π^* on a machine with 1.7 GHz processor and 6 GB RAM ranges from 2.0 seconds to 206.0 seconds, with average 67.0 seconds and standard deviation 86.2 seconds; the time of computing r^* ranges from 3.0 seconds to 5.0 seconds, with average 4.0 seconds and standard deviation 1.0 second; the time of computing p^* ranges from 5.0 seconds to 7.0 seconds, with average 5.8 seconds and standard deviation 0.8 second; and the time of computing d^* ranges from 39.0 seconds to 40.0 seconds, with average 39.6 seconds and standard deviation 0.5 second.[8]

Additional experiments were conducted by varying the cost of running KDD from 70% of the cost listed in Table 2 to 130% of the cost. We also conducted experiments by varying the costs of knowledge loss from 70% of the costs listed in Table 2 to 130% of the costs. As shown in Tables 5–6, π^* outperforms p^*, r^*, and d^* substantially in all the experiments.

Extensive empirical results suggest that the proposed optimal knowledge-refreshing policy π^* significantly outperforms optimal fixed-interval knowledge-refreshing policies. The superiority of π^* is attributed to the fact that π^* is optimal among all possible knowledge-refreshing policies

Table 4. Performance comparisons between π^* and optimal fixed-interval policies.

ϕ	C_{π^*} (\$)	C_{p^*} (\$)	ψ_1 (%)	C_{r^*} (\$)	ψ_2 (%)	C_{d^*} (\$)	ψ_3 (%)
0.5	209,313.49	541,538.63	61.3	222,875.00	6.1	526,275.47	60.2
0.75	199,979.60	413,234.17	51.6	222,824.23	10.3	399,652.66	50.0
1	177,559.48	325,643.06	45.5	213,986.31	17.0	311,710.84	43.0
1.5	139,569.63	216,702.55	35.6	163,781.61	14.8	208,147.68	32.9
2	110,745.93	155,388.88	28.7	129,149.98	14.3	149,420.51	25.9

Table 5. Performance comparisons between π^* and optimal fixed-interval policies ($\phi = 1$, various c^k).

C^k	C_{π^*} (\$)	C_{p^*} (\$)	ψ_1 (%)	C_{r^*} (\$)	ψ_2 (%)	C_{d^*} (\$)	ψ_3 (%)
$0.7 \times 1{,}783$	136,309.56	294,450.39	53.7	156,066.25	12.7	281,169.71	51.5
$0.9 \times 1{,}783$	164,877.98	315,300.94	47.7	198,417.81	16.9	301,875.48	45.4
1,783	177,559.48	325,643.06	45.5	213,986.31	17.0	311,710.84	43.0
$1.1 \times 1{,}783$	190,754.84	334,772.29	43.0	225,423.69	15.4	321,890.76	40.7
$1.3 \times 1{,}783$	213,008.78	352,336.00	39.5	247,560.79	14.0	339,081.77	37.2

Table 6. Performance comparisons between π^* and optimal fixed-interval policies ($\phi = 1$, various c_{r_1}, c_{r_2}).

c_{r_1}	c_{r_2}	C_{π^*} (\$)	C_{p^*} (\$)	ψ_1 (%)	C_{r^*} (\$)	ψ_2 (%)	C_{d^*} (\$)	ψ_3 (%)
$0.7 \times 4{,}915.41$	$0.7 \times 10{,}426.63$	157,921.57	252,837.5	37.5	182,825.92	13.6	243,211.68	35.1
$0.9 \times 4{,}915.41$	$0.9 \times 10{,}426.63$	172,015.05	301,092.57	42.9	204,155.50	15.7	289,298.31	40.5
4,915.41	10,426.63	177,559.48	325,643.06	45.5	213,986.31	17.0	311,710.84	43.0
$1.1 \times 4{,}915.41$	$1.1 \times 10{,}426.63$	182,779.25	348,741.99	47.6	220,008.10	16.9	333,684.30	45.2
$1.3 \times 4{,}915.41$	$1.3 \times 10{,}426.63$	190,563.81	393,266.04	51.5	222,859.14	14.5	375,084.15	49.2

Table 7. Experimental results regarding the robustness of π^* with respect to the Poisson arrival assumptions.

		Hyperexponential		Erlang-2		Erlang-4		Uniform	
ϕ	Exponential cost (\$)	Cost (\$)	σ (%)	Cost (\$)	σ (%)	Cost (\$)	σ (%)	Cost (\$)	σ (%)
0.5	209,313.49	211,588.04	1.1	211,585.19	1.1	212,042.57	1.3	210,806.13	0.7
0.75	199,979.60	204,806.09	2.4	205,020.17	2.5	207,407.83	3.7	203,595.06	1.8
1	177,559.48	183,239.95	3.2	183,240.02	3.2	184,538.45	3.9	181,575.80	2.3
1.5	139,569.63	143,981.61	3.2	144,689.17	3.7	145,881.90	4.5	142,823.80	2.3
2	110,745.93	112,428.29	1.5	114,633.61	3.5	115,303.24	4.1	114,675.60	3.5

whereas an optimal fixed-interval knowledge-refreshing policy is the best policy within a reduced space containing only fixed-interval knowledge-refreshing policies.

5.4. Robustness Analysis

This subsection analyzes the robustness of π^* with respect to the Poisson arrival assumptions. We conducted experiments in which different distributions (exponential and nonexponential) were used to determine interarrival times of data and requests. Specifically, π^* was computed using the proposed algorithm based on the Poisson arrival assumptions. The computed policy was then implemented to determine running or not running KDD in an experiment with exponentially distributed interarrival times of data and requests and in several other experiments with nonexponentially distributed interarrival times of data and requests. We carefully selected nonexponential distributions with different coefficients of variation. The chosen nonexponential distributions, listed with their coefficients of variation in the parentheses, include the Erlang-4 distribution (0.5), the uniform distribution (0.58), the Erlang-2 distribution (0.71), and the hyperexponential distribution (1.41). Both the exponential distribution and the nonexponential distributions were set to have the same mean.

For the experiments, parameters were set as those listed in Table 2 and $M = 125$. The shape parameter ϕ was varied from 0.5 to 2. The difference, between the performance of π^* under nonexponentially distributed interarrival times of data and requests and the performance of π^* under exponentially distributed interarrival times of data and requests, represents the effect of the Poisson arrival assumptions on π^*. The difference σ is measured as $\sigma = |C^P_{\pi^*} - C^{NP}_{\pi^*}|/C^P_{\pi^*}$, where $C^P_{\pi^*}$ is the total system cost resulting from implementing π^* under exponentially distributed interarrival times of data and requests, and $C^{NP}_{\pi^*}$ is the total system cost resulting from implementing π^* under nonexponentially distributed interarrival times of data and requests. As shown in Table 7, σ is small in all the experiments and the experimental results indicate that the Poisson arrival assumptions have little impact on the performance of π^*.

6. Conclusions

This paper studies the problem of deciding the right times to run KDD to refresh knowledge. We analyze important characteristics of the optimal knowledge-refreshing policy and propose a method for computing the policy. The effectiveness and the robustness of the computed optimal knowledge-refreshing policy are examined through extensive empirical studies with a real-world knowledge-refreshing problem. By addressing a fundamental challenge faced by all KDD applications, we have made contributions to both the literature and the practice. On the literature side, we identify theoretical properties characterizing the optimal knowledge-refreshing policy and propose a method for its computation. Practically, the proposed method is implementable. We demonstrate the implementation of the proposed method in a real-world knowledge-refreshing scenario and show that the computed optimal knowledge-refreshing policy substantially outperforms optimal fixed-interval policies.

Our research can be extended in several directions. This study treats the cost of running KDD as fixed cost based on a popular procedure of running KDD in practice. Future research should study how the proposed model and method can be extended to handle the variable cost of running KDD. Another area merit worth of exploration is to extend our current study to accommodate the situation where data and request arrival rates change over time. A plausible extension is to divide the planning horizon into multiple

periods of constant data and request arrival rates and to compute the optimal knowledge-refreshing policy for each period. In addition, future research efforts are needed to better model and estimate cost parameters c_{r_i} for real-world knowledge-refreshing applications. In this regard, a model based on marketing analysis needs to be developed for the derivation of these cost parameters. The knowledge-refreshing model can be enhanced by incorporating this model for cost parameters. Finally, additional evaluations of the proposed method for computing optimal knowledge-refreshing policy should attempt to generate more empirical evidences about its effectiveness and practical value.

Electronic Companion

An electronic companion to this paper is available as part of the online version at http://dx.doi.org/10.1287/opre.1120.1148.

Endnotes

1. In this study, we focus on quantifying knowledge loss for KDD techniques based on major data-mining models. In the case of KDD techniques that are based on regression and statistical models, there might not exist apparent set representation of knowledge and other equations for quantifying knowledge loss need to be designed.
2. A function f is monotonically increasing if for $x < y$ in its domain one has $f(x) = f(y)$.
3. The algorithm needs to store $J(s_m)$ for $m = 1, 2, \ldots, M$ and the storage complexity of the algorithm is $O(\kappa)$. According to Proposition 2, executing the first line in the outer for-loop requires $O(k_{m+1})$ time, where $k_{m+1} = \sum_{i=1}^{n} k_{m+1,i}$, $k_{m+1,i}$ is the threshold for request type r_i at time $m+1$, and $m = 1, 2, \ldots, M-1$. The inner for-loop searches $k_{m,i}$ for each request type r_i, $i = 1, 2, \ldots, n$, and it requires $O(k_m)$ time, where $k_m = \sum_{i=1}^{n} k_{m,i}$, $k_{m,i}$ is the threshold for request type r_i at time m, and $m = 1, 2, \ldots, M-1$. Hence, the running time of the algorithm is $\sum_{m=1}^{M-1} (k_m + k_{m+1})$, which is of the order of $O(\kappa)$.
4. For example, $\rho = 0.9$ implies that there is still some revenue realized even if an email campaign is supported by a fully outdated knowledge base.
5. Following a popular procedure of running KDD over evolving data sources (Altiparmak et al. 2008, Gaber et al. 2005), KDD was run on the most recent n_b transactions in the updated data set.
6. The benchmark optimal fixed-interval policies and their costs were derived using exhaustive search.
7. Each result reported in Tables 4–7 represents an average of 200 experimental runs.
8. The time complexity of computing π^* is $O(\kappa)$, where $\kappa = \sum_{m=1}^{M} \sum_{i=1}^{n} k_{m,i}$. As ϕ increases, $k_{m,i}$ increases. Hence, the time of computing π^* increases as ϕ increases. The search space of computing an optimal fixed-interval policy does not depend on $k_{m,i}$ and thus the time of computing an optimal fixed-interval policy is stable across the experiments.

References

Adelberg B, Garcia-Molina H, Kao B (1995) Applying update streams in a soft real-time database system. *ACM SIGMOD Record* 24(2): 245–256.

Aggarwal C (2007) *Data Streams: Models and Algorithms* (Springer, New York).

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. *Proc. 1994 VLDB Conf., Santiago, Chile*.

Albert T, Goes PB, Gupta A (2004) GIST: A model for design and management of content and interactivity of customer-centric Web sites. *MIS Quart.* 28(2):161–181.

Altiparmak F, Tuncel E, Ferhatosmanoglu H (2008) Incremental maintenance of online summaries over multiple streams. *IEEE Trans. Knowledge Data Engrg.* 20(2):216–229.

Baesens B, Setiono R, Mues C, Vanthienen J (2003) Using neural network rule extraction and decision tables for credit risk evaluation. *Management Sci.* 49(3):312–329.

Bellman R, Dreyfus SE (1962) *Applied Dynamic Programming* (Princeton University Press, Princeton, NJ).

Bensoussan A, Mookerjee R, Mookerjee V, Yue WT (2009) Maintaining diagnostic knowledge-based systems: A control theoretic approach. *Management Sci.* 55(2):294–310.

Block HW, Griffith WS, Savits TH (1989) *L*-superadditive structure functions. *Adv. Appl. Probab.* 21:919–929.

Can F (1993) Incremental clustering for dynamic information processing. *ACM Trans. Inform. Systems* 11(2):143–164.

Chandy KM, Browne JC, Dissly CW, Uhrig WR (1975) Analytical models for rollback and recovery strategies in data base systems. *IEEE Trans. Software Engrg.* 1(1):100–110.

Chen H (2006) Intelligence and security informatics: Information systems perspective. *Decision Support Systems* 41(3):555–559.

Chen H, Zeng D (2009) AI for global disease surveillance. *IEEE Intelligent Systems* 24(6):66–69.

Cheung DW, Han J, Ng VT, Wong CY (1996) Maintenance of discovered association rules in large databases: An incremental updating technique. *Proc. 1996 Internat. Conf. Data Engrg., New Orleans*.

Cooper LG, Giuffrida G (2000) Turning data mining into a management science tool: New algorithms and empirical results. *Management Sci.* 46(2):249–264.

Dey D, Zhang Z, De P (2006) Optimal synchronization policies for data warehouses. *INFORMS J. Comput.* 18(2):229–242.

Domingos P, Hulten G (2000) Mining high-speed data streams. *Proc. 2000 ACM SIGKDD* (ACM, New York), 71–80.

Fang X, Rachamadugu R (2009) Policies for knowledge refreshing in databases. *Omega: The Internat. J. Management Sci.* 37(1):16–28.

Fayyad U, Piatetsky-Shapiro G, Smyth P (1996) The KDD process for extracting useful knowledge from volumes of data. *Comm. ACM* 39(11):27–34.

Gaber M, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: A review. *ACM SIGMOD Record* 34(2):19–26.

Ganti V, Gehrke J, Ramakrishinan R, Loh WY (1999) A framework for measuring changes in data characteristics. *Proc. 18th ACM SIGMOD Sympos. Principles of Database Systems*, 126–137.

Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L (2003) Clustering data streams: Theory and practice. *IEEE Trans. Knowledge and Data Engrg.* 15(3):515–528.

Han J, Kamber M (2006) *Data Mining: Concepts and Techniques* (Morgan Kaufmann, San Francisco).

Huet S, Bouvier A, Gruet M, Jolivet E (1996) *Statistical Tools for Nonlinear Regression* (Springer-Verlag, New York).

Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. *Proc. 2001 ACM SIGKDD* (ACM, New York).

King RW, Marks PV, McCoy S (2002) The most important issues in knowledge management. *Comm. ACM* 45(9):93–97.

Linden G, Smith B, York J (2003) Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Comput.* 7(1):76–80.

Manku GS, Motwani R (2002) Approximate frequency counts over data streams. *Proc. 28th VLDB Conf., Hong Kong*.

Park JS, Bartoszynski R, De P, Pirkul H (1990) Optimal reorganization policies for stationary and evolutionary databases. *Management Sci.* 36(5):613–631.

Quinlan JR (1993) *C 4.5: Programs for Machine Learning* (Morgan Kaufmann, San Francisco).

Rebbapragada R, Basu A, Semple J (2010) Use of data mining and revenue management methodologies in college admissions. *Comm. ACM* 53(4):128–133.

Sarkar S, Sriram RS (2001) Bayesian models for early warning of bank failures. *Management Sci.* 47(11):1457–1475.

Schlimmer JC, Fisher D (1986) A case study of incremental concept induction. *Proc. 5th Natl. Conf. Artificial Intelligence, Philadelphia*, 496–501.

Segev A, Fang W (1991) Optimal update policies for distributed materialized views. *Management Sci.* 37(7):851–870.

Srivastava J, Rotem D (1988) Analytical modeling of materialized view maintenance. *Proc. 7th Sympos. Principles of Database Systems, Austin, TX*, 126–134.

Thomas S, Bodagala S, Alsabti K, Ranka S (1997) An efficient algorithm for the incremental updation of association rules in large databases. *Proc. 1997 ACM SIGKDD* (ACM, New York), 263–266.

Utgoff PE (1989) Incremental induction of decision trees. *Machine Learn.* 4:161–186.

Xiao Fang is an assistant professor in the Department of Operations and Information Systems at the David Eccles School of Business, the University of Utah. His current research interests are in the areas of data mining, business analytics, social network analytics, and big data.

Olivia R. Liu Sheng is Presidential Professor and Emma Eccles Jones Presidential Chair of Information Systems at the David Eccles School of Business, and an adjunct professor at the School of Computing, University of Utah. Her research focuses on Web, text, and data mining, as well as optimization techniques for clickstream analysis, search marketing, crowd-based predictive analytics, behavior targeting, biomedical, digital government, telemedicine, and telework applications.

Paulo Goes is the Salter Distinguished Professor of Management and Technology and head of the Management Information Systems Department at the Eller College of Management, University of Arizona. His research interests are in the areas of design and evaluation of IT-enabled business models, innovation exploration, emerging technologies, e-commerce and online auctions, database technology and systems, and technology infrastructure.

Appendix A: Derivation of Equations (4) and (5)

In this appendix, we first compute the joint probability of q_m and $y_{m-1,m}$; we then derive equations (4) and (5).

1. The joint probability of q_m and $y_{m-1,m}$ is given by

$$P\{q_m = r_i, y_{m-1,m} = k\} = \frac{\mu^k \lambda_i}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k+1}} \qquad i = 1,2,\cdots,n \text{ and } k \in Z^+. \tag{A1}$$

Proof. Let $T_{m-1,m}$ denote the elapsed chronological time between system time *m*-1 and system time *m*. According to the independent Poisson arrival assumption for requests, the probability density function $f_{T_{m-1,m}}(\tau)$ of $T_{m-1,m}$ is

$$f_{T_{m-1,m}}(\tau) = \sum_{j=1}^{n} \lambda_j e^{-\sum_{j=1}^{n} \lambda_j \tau}. \tag{A2}$$

We have,

$$\begin{aligned} P\{y_{m-1,m} = k\} &= \int_0^\infty P\{y_{m-1,m} = k \mid T_{m-1,m} = \tau\} f_{T_{m-1,m}}(\tau) d\tau \\ &= \int_0^\infty e^{-\mu\tau} \frac{(\mu\tau)^k}{k!} f_{T_{m-1,m}}(\tau) d\tau \qquad \text{by poisson arrival for new data} \\ &= \int_0^\infty e^{-\mu\tau} \frac{(\mu\tau)^k}{k!} \sum_{j=1}^{n} \lambda_j e^{-\sum_{j=1}^{n} \lambda_j \tau} d\tau \qquad \text{by applying (A2)} \\ &= \frac{\mu^k \sum_{j=1}^{n} \lambda_j}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k+1}}. \end{aligned} \tag{A3}$$

According to the independence assumption between data arrivals and request arrivals, we have,

$$\begin{aligned} P\{y_{m-1,m}=k, q_{\mathrm{m}}=r_i\} &= P\{y_{m-1,m}=k\}\times P\{q_m=r_i\} \\ &= \frac{\mu^k \sum_{j=1}^{n}\lambda_j}{(\mu+\sum_{j=1}^{n}\lambda_j)^{k+1}} \times \frac{\lambda_i}{\sum_{j=1}^{n}\lambda_j} \qquad \text{by applying (A3)} \\ &= \frac{\mu^k \lambda_i}{(\mu+\sum_{j=1}^{n}\lambda_j)^{k+1}}. \end{aligned}$$

2. Derivation of equations (4) and (5)

Let us first derive (4). If $k_2 \geq k_1$, according to (3), the independent arrival assumption for requests and the independence assumption between data arrivals and requests arrivals, we have,

$$\begin{aligned} &P\{q_{m+1}=r_i, d_{m+1}=k_2 \mid q_m=r_h, d_m=k_1, a_m=0\} \\ &= P\{q_{m+1}=r_i, y_{m,m+1}=k_2-k_1\} \\ &= \frac{\mu^{k_2-k_1}\lambda_i}{(\mu+\sum_{j=1}^{n}\lambda_j)^{k_2-k_1+1}}. \qquad \text{by applying (A1)} \end{aligned}$$

If $k_2 < k_1$, the transition probability is 0 because, by equation (3), $d_{m+1} \geq d_m$ under action $a_m = 0$.

We then derive (5). According to (3), the independent arrival assumption for requests and the independence assumption between data arrivals and requests arrivals, we have,

$$\begin{aligned} &P\{q_{m+1}=r_i, d_{m+1}=k_2 \mid q_m=r_h, d_m=k_1, a_m=1\} \\ &= P\{q_{m+1}=r_i, y_{m,m+1}=k_2\} \\ &= \frac{\mu^{k_2}\lambda_i}{(\mu+\sum_{j=1}^{n}\lambda_j)^{k_2+1}}. \qquad \text{by applying (A1)} \end{aligned}$$

Appendix B: Proofs of Lemmas and Theorem

We first present Lemma B1 (Puterman 2005), which is employed in the proofs of Lemma 2 and Theorem 1. We then give the proofs of the Lemmas and the Theorem.

Lemma B1: Let $\{x_j\},\{x'_j\}$ be real-valued non-negative sequences satisfying

$$\sum_{j=k}^{\infty} x_j \geq \sum_{j=k}^{\infty} x'_j \tag{B1}$$

for all $k \in Z^+$. Suppose $v_{j+1} \geq v_j$ for $j = 0,1,\cdots,$ then

$$\sum_{j=0}^{\infty} v_j x_j \geq \sum_{j=0}^{\infty} v_j x'_j. \tag{B2}$$

See Puterman (2005) for the proof.

1. Proof of Lemma 1

Let $s_m = (q_m, d_m)$, $s_{m+1} = (q_{m+1}, d_{m+1})$, and $s^k = (q^k, d^k)$, where q^k denotes request type and d^k denotes amount of new data. For any $s^k \in S$ and a sub-space S_j, s^k is either in S_j or not in S_j. We consider each of the two cases.

Case 1: $s^k \in S_j$

By Definition 3, $s^k \in S_j$ implies $q^k = r_j$; similarly, $s_{m+1} \in S_j$ implies $q_{m+1} = r_j$. Hence, by Definition 2, $s_{m+1} \geq s^k$ implies $d_{m+1} \geq d^k$. We thus have

$$f(s_m) = \sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P^{a_m}_{s_m s_{m+1}} = \sum_{y=d^k}^{\infty} P\{q_{m+1} = r_j, d_{m+1} = y \mid q_m, d_m, a_m\}. \qquad \text{if } s^k \in S_j \tag{B3}$$

For $s_m \in S_i$, by Definitions 2 and 3, the increasing of s_m implies the increasing of d_m and vice versa because system states belonging to a sub-space have same request type. Hence, for $s_m \in S_i$, to prove $f(s_m) = \sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P_{s_m s_{m+1}}^{a_m}$ increasing[1] in s_m is equivalent to show $\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P_{s_m s_{m+1}}^{a_m}$ increasing in d_m.

We first consider the situation of $a_m = 0$. If $a_m = 0$ and $d_m < d^k$, by applying (4), (B3) becomes

$$\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P_{s_m s_{m+1}}^{a_m} = \sum_{y=d^k}^{\infty} \frac{\mu^{y-d_m} \lambda_j}{(\mu + \sum_{i=1}^{n} \lambda_i)^{y-d_m+1}} = \frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i} (\frac{\mu}{\mu + \sum_{i=1}^{n} \lambda_i})^{d^k - d_m}. \quad \text{(B4)}$$

If $a_m = 0$ and $d_m \geq d^k$, by applying (4), (B3) becomes

$$\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P_{s_m s_{m+1}}^{a_m} = \sum_{y=d_m}^{\infty} \frac{\mu^{y-d_m} \lambda_j}{(\mu + \sum_{i=1}^{n} \lambda_i)^{y-d_m+1}} = \frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i}. \quad \text{(B5)}$$

By observing (B4) and (B5), $\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P_{s_m s_{m+1}}^{a_m}$ is increasing in d_m for $d_m < d^k$ and $d_m \geq d^k$ respectively.

Further, $\frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i} (\frac{\mu}{\mu + \sum_{i=1}^{n} \lambda_i})^{d^k - d_m}$ in (B4) is less than $\frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i}$ in (B5). Therefore, if $a_m = 0$, $\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P_{s_m s_{m+1}}^{a_m}$ is increasing in d_m and it is increasing in s_m.

If $a_m = 1$, by applying (5), (B3) becomes

$$\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P_{s_m s_{m+1}}^{a_m} = \sum_{y=d^k}^{\infty} \frac{\mu^{y} \lambda_j}{(\mu + \sum_{i=1}^{n} \lambda_i)^{y+1}} = \frac{\lambda_j}{\sum_{i=1}^{n} \lambda_i} (\frac{\mu}{\mu + \sum_{i=1}^{n} \lambda_i})^{d^k}. \quad \text{(B6)}$$

[1] In Appendix B, the word "increasing" means monotonically increasing.

$\frac{\lambda_j}{\sum_{i=1}^{n}\lambda_i}(\frac{\mu}{\mu+\sum_{i=1}^{n}\lambda_i})^{d^k}$ in (B6) is a constant. Hence, if $a_m=1$, $\sum_{s_{m+1}\geq s^k, s_{m+1}\in S_j} P_{s_m s_{m+1}}^{a_m}$ is increasing in d_m and it is increasing in s_m. In summary, for the case of $s^k \in S_j$, $\sum_{s_{m+1}\geq s^k, s_{m+1}\in S_j} P_{s_m s_{m+1}}^{a_m}$ is increasing in s_m for all $a_m \in A$.

Case 2: $s^k \notin S_j$

By Definition 3, $s^k \notin S_j$ implies $q^k \neq r_j$ and $s_{m+1} \in S_j$ implies $q_{m+1}=r_j$.

By Definition 2, $s_{m+1} \geq s^k$ implies

$$c_{r_j}(F(d_{m+1}))^{\phi} \geq c_{q^k}(F(d^k))^{\phi}. \tag{B7}$$

Let d^z be the minimum d_{m+1} satisfying (B7). We have

$$f(s_m)=\sum_{s_{m+1}\geq s^k, s_{m+1}\in S_j} P_{s_m s_{m+1}}^{a_m} = \sum_{y=d^z}^{\infty} P\{q_{m+1}=r_j, d_{m+1}=y \mid q_m, d_m, a_m\}. \quad \text{if } s^k \notin S_j \tag{B8}$$

The only difference between (B8) and (B3) is the summation constant, which is d^z in (B8) but d^k in (B3). Therefore, we can prove $\sum_{s_{m+1}\geq s^k, s_{m+1}\in S_j} P_{s_m s_{m+1}}^{a_m}$ in (B8) is increasing in s_m for all $a_m \in A$ in a similar way to the proof of Case 1, which is not repeated for the sake of space. Overall, we have shown that $f(s_m)=\sum_{s_{m+1}\geq s^k, s_{m+1}\in S_j} P_{s_m s_{m+1}}^{a_m}$ is increasing in s_m for all $s^k \in S$ and $a_m \in A$. This completes the proof.

2. Proof of Lemma 2

We first show $c(s_m, a_m)$, $m=1,2,\cdots,M$, is increasing in $s_m \in S_i$, $i=1,2,\cdots,n$, for all $a_m \in A$. If $a_m=0$, according to (6), $c(s_m,a_m)=c^l(s_m)$, where $c^l(s_m)$ is the cost of knowledge loss incurred at state s_m. By Definition 2, $c^l(s_m)$ is increasing in s_m. Hence, $c(s_m,a_m)$ is increasing in $s_m \in S_i$ for

$a_m = 0$. If $a_m = 1$, according to (6), $c(s_m, a_m) = c^k$, which is a constant. Therefore, $c(s_m, a_m)$ is increasing in $s_m \in S_i$ for $a_m = 1$.

We use induction to prove Lemma 2. We first show $J(s_M)$ is increasing in $s_M \in S_i$, $i = 1,2,\cdots,n$. By (9),

$$J(s_M) = \min_{a_M \in A}\{c(s_M, a_M)\} = \begin{cases} c^l(s_M) & \text{if } c^l(s_M) \le c^k, \\ c^k & \text{if } c^l(s_M) > c^k, \end{cases}$$

where $c^l(s_M)$ is the cost of knowledge loss incurred at state s_M. By Definition 2, $c^l(s_M)$ is increasing in s_M. Hence, $J(s_M)$ is increasing in $s_M \in S_i$.

Next, we need to prove $J(s_b)$ is increasing in $s_b \in S_i$, assuming that $J(s_{b+1})$ is increasing in $s_{b+1} \in S_j$, where $b = 1,2,\cdots,M-1$ and $i, j = 1,2,\cdots,n$. By (9),

$$J(s_b) = \min_{a_b \in A}\{c(s_b, a_b) + \sum_{s_{b+1} \in S} P^{a_b}_{s_b s_{b+1}} J(s_{b+1})\}.$$

Let $Q(s_b, a_b) = c(s_b, a_b) + \sum_{s_{b+1} \in S} P^{a_b}_{s_b s_{b+1}} J(s_{b+1})$. We have

$$J(s_b) = \min_{a_b \in A}\{Q(s_b, a_b)\}. \tag{B9}$$

We show (i) $Q(s_b, a_b)$ is increasing $s_b \in S_i$ for all $a_b \in A$; and then prove (ii) $J(s_b)$ is increasing in $s_b \in S_i$, $i = 1,2,\cdots,n$.

(i) By Definition 3, $S = \bigcup_{j=1}^{n} S_j$, we can rewrite $Q(s_b, a_b)$ as

$$Q(s_b, a_b) = c(s_b, a_b) + \sum_{j=1}^{n} \sum_{s_{b+1} \in S_j} P^{a_b}_{s_b s_{b+1}} J(s_{b+1}). \tag{B10}$$

Let $s_{b+1} = (q_{b+1}, d_{b+1})$ and $s^k = (q^k, d^k) \in S$. By (B3) and (B8),

$$\sum_{s_{b+1}\geq s^k, s_{b+1}\in S_j} P_{s_b s_{b+1}}^{a_b} = \sum_{y=d}^{\infty} P\{q_{b+1}=r_j, d_{b+1}=y \mid s_b, a_b\},$$

where $d = d^k$ if $s^k \in S_j$ and $d = d^z$ if $s^k \notin S_j$.

By Lemma 1, $\sum_{s_{b+1}\geq s^k, s_{b+1}\in S_j} P_{s_b s_{b+1}}^{a_b}$ is increasing in $s_b \in S_i$ for all $s^k \in S$ and $a_b \in A$. Hence, for $s^1 > s^2$ in S_i, we have, for all $a_b \in A$ and $d \in Z^+$,

$$\sum_{y=d}^{\infty} P\{q_{b+1}=r_j, d_{b+1}=y \mid s_b=s^1, a_b\} \geq \sum_{y=d}^{\infty} P\{q_{b+1}=r_j, d_{b+1}=y \mid s_b=s^2, a_b\}.$$

By the induction assumption, $J(s_{b+1})$ is increasing in $s_{b+1} \in S_j$, $j=1,2,\cdots,n$. Applying Lemma B1, we have, for all $a_b \in A$,

$$\begin{aligned}\sum_{y=0}^{\infty} P\{q_{b+1}=r_j, d_{b+1}=y \mid s_b=s^1, a_b\} J(q_{b+1}=r_j, d_{b+1}=y) \\ \geq \sum_{y=0}^{\infty} P\{q_{b+1}=r_j, d_{b+1}=y \mid s_b=s^2, a_b\} J(q_{b+1}=r_j, d_{b+1}=y),\end{aligned} \tag{B11}$$

where $s^1, s^2 \in S_i$ and $s^1 > s^2$.

Since $$\sum_{s_{b+1}\in S_j} P_{s_b s_{b+1}}^{a_b} J(s_{b+1}) = \sum_{y=0}^{\infty} P\{q_{b+1}=r_j, d_{b+1}=y \mid s_b, a_b\} J(q_{b+1}=r_j, d_{b+1}=y), \tag{B11}$$

indicates that $\sum_{s_{b+1}\in S_j} P_{s_b s_{b+1}}^{a_b} J(s_{b+1})$, $j=1,2,\cdots,n$, is increasing in $s_b \in S_i$ for all $a_b \in A$. We have already shown $c(s_b, a_b)$ is increasing in $s_b \in S_i$ for all $a_b \in A$. Therefore, every component of $Q(s_b, a_b)$ in (B10) is increasing in $s_b \in S_i$ for all $a_b \in A$. Consequently, $Q(s_b, a_b)$ is increasing in $s_b \in S_i$ for all $a_b \in A$.

(ii) For $s^1 > s^2$ in S_i, let $a^* \in A$ be the optimal action chosen for $s_b = s^1$. We have

$$\begin{aligned} J(s_b = s^1) &= Q(s_b = s^1, a_b = a^*) && \text{by B(9)} \\ &\geq Q(s_b = s^2, a_b = a^*) && \text{by } Q(s_b, a_b) \text{ is increasing in } s_b \in S_i \\ &\geq J(s_b = s^2). && \text{by B(9)} \end{aligned}$$

Therefore, $J(s_b)$ is increasing in $s_b \in S_i$, $i = 1,2,\cdots,n$. This completes the proof.

3. Proof of Lemma 3

To show $h(s_m, a_m)$ is superadditive, we need to prove, for $s^1 \geq s^2$ in S_i and $a^1 \geq a^2$ in A,

$$h(s^1, a^1) + h(s^2, a^2) \geq h(s^1, a^2) + h(s^2, a^1). \qquad \text{by Definition 4}$$

Since $h(s_m, a_m) = -c(s_m, a_m)$, it is equivalent to show,

$$-c(s^1, a^1) - c(s^2, a^2) \geq -c(s^1, a^2) - c(s^2, a^1). \qquad \text{(B12)}$$

If $a^1 > a^2$, i.e., $a^1 = 1$ and $a^2 = 0$ according to Definition 1, the left hand side of (B12) becomes

$$-c(s^1, a^1 = 1) - c(s^2, a^2 = 0) = -c^k - c^l(s^2), \qquad \text{by (6)}$$

where $c^l(s^2)$ is the cost of knowledge loss incurred at state s^2.

Similarly, the right hand side of (B12) becomes,

$$-c(s^1, a^2 = 0) - c(s^2, a^1 = 1) = -c^l(s^1) - c^k, \qquad \text{by (6)}$$

where $c^l(s^1)$ is the cost of knowledge loss incurred at state s^1. By Definition 2 and $s^1 \geq s^2$, we have $c^l(s^1) \geq c^l(s^2)$. Hence, (B12) holds for $a^1 > a^2$ and $s^1 \geq s^2$.

If $a^1 = a^2$ and $s^1 \geq s^2$, it is obvious that

$$-c(s^1, a^1) - c(s^2, a^2) = -c(s^1, a^2) - c(s^2, a^1).$$

This completes the proof.

4. Proof of Lemma 4

To show $g(s_m, a_m)$ is superadditive, we need to prove, for $s^1 \geq s^2$ in S_i and $a^1 \geq a^2$ in A,

$$g(s^1, a^1) + g(s^2, a^2) \geq g(s^1, a^2) + g(s^2, a^1). \quad \text{by Definition 4} \quad \text{(B13)}$$

If $a^1 > a^2$, i.e., $a^1 = 1$ and $a^2 = 0$ according to Definition 1, we need to show

$$g(s^1, a^1 = 1) + g(s^2, a^2 = 0) \geq g(s^1, a^2 = 0) + g(s^2, a^1 = 1).$$

By observing (B6), $\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P^{a_m}_{s_m s_{m+1}}$ is a constant for $a_m = 1$. Therefore, $g(s^1, a^1 = 1) = g(s^2, a^1 = 1)$.

By observing (B4) and (B5), $\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P^{a_m}_{s_m s_{m+1}}$ is increasing in s_m for $a_m = 0$. Hence, $g(s_m, a_m = 0)$ is decreasing in s_m for $a_m = 0$, where $g(s_m, a_m) = -\sum_{s_{m+1} \geq s^k, s_{m+1} \in S_j} P^{a_m}_{s_m s_{m+1}}$. Consequently, for $s^1 \geq s^2$, we have $g(s^2, a^2 = 0) \geq g(s^1, a^2 = 0)$. Therefore, (B13) holds for $a^1 > a^2$ and $s^1 \geq s^2$.

If $a^1 = a^2$ and $s^1 \geq s^2$, it is obvious that

$$g(s^1, a^1) + g(s^2, a^2) = g(s^1, a^2) + g(s^2, a^1).$$

This completes the proof.

5. Proof of Theorem 1

We first present a condition under which a monotonically increasing optimal policy exists for a discrete time Markov decision process and then prove Theorem 1 by showing that the condition is satisfied by the knowledge refreshing problem.

The condition: For a discrete time Markov decision process with an objective to maximize rewards, let the value function be the reward received at current decision point plus the expected optimal reward from next decision point to the end of the planning horizon. If the value function is superadditive

at each decision point, there exists an optimal policy monotonically increasing in the system state (Serfozo 1976, Puterman 2005).

Since the objective of the knowledge refreshing problem is to minimize costs, we need to redefine the objective without changing the problem. For the knowledge refreshing problem, let the reward received at time m, with system state s_m and action a_m, be $-c(s_m, a_m)$, $m = 1,2,\cdots,M$. Now the objective of the problem becomes maximizing rewards. Let $R(s_m)$ be the optimal expected total reward from time m, with state s_m, to the end of the time horizon, $m = 1,2,\cdots,M$, and $R(s_{M+1}) = 0$ for all $s_{M+1} \in S$. For $m = 1,2,\cdots,M$,

$$R(s_m) = \max_{a_m \in A}\{-c(s_m, a_m) + \sum_{s_{m+1} \in S} P^{a_m}_{s_m s_{m+1}} R(s_{m+1})\}. \tag{B14}$$

Since the reward is the negation of the system cost, we have

$$R(s_m) = -J(s_m). \tag{B15}$$

Let $V(s_m, a_m)$ be the value function on $S_i \times A$, where S_i is a sub-space, $i = 1,2,\cdots,n$, and A is the action space. For $m = 1,2,\cdots,M$, we have

$$V(s_m, a_m) = -c(s_m, a_m) + \sum_{s_{m+1} \in S} P^{a_m}_{s_m s_{m+1}} R(s_{m+1}). \tag{B16}$$

By the condition for the monotonically increasing optimal policy, to prove Theorem 1, we need to show $V(s_m, a_m)$ is superadditive for $m = 1,2,\cdots,M$.

For $m = M$, $V(s_M, a_M) = -c(s_M, a_M)$. By Lemma 3, $V(s_M, a_M)$ is superadditive on $S_i \times A$.

For $m = 1,2,\cdots,M-1,$ by Definition 3, $S = \bigcup_{j=1}^{n} S_j$, we can rewrite $V(s_m, a_m)$ as

$$V(s_m, a_m) = -c(s_m, a_m) + \sum_{j=1}^{n} \sum_{s_{m+1} \in S_j} P^{a_m}_{s_m s_{m+1}} R(s_{m+1}). \tag{B17}$$

To prove $V(s_m,a_m)$ is superadditive, we need to show each component of $V(s_m,a_m)$ expressed in (B17) is superadditive. By Lemma 3, $-c(s_m,a_m)$ is superadditive on $S_i\times A$. We are left to show $\sum_{s_{m+1}\in S_j} P^{a_m}_{s_m s_{m+1}} R(s_{m+1})$ is superadditive on $S_i\times A$, where $s_m\in S_i$ and $a_m\in A$.

Let $s_{m+1}=(q_{m+1},d_{m+1})$ and $s^k=(q^k,d^k)\in S$, by (B3) and (B8),

$$\sum_{s_{m+1}\geq s^k, s_{m+1}\in S_j} P^{a_m}_{s_m s_{m+1}} = \sum_{y=d}^{\infty} P\{q_{m+1}=r_j, d_{m+1}=y \mid s_m,a_m\}, \tag{B18}$$

where $d=d^k$ if $s^k\in S_j$ and $d=d^z$ if $s^k\notin S_j$. By Lemma 4, $-\sum_{s_{m+1}\geq s^k, s_{m+1}\in S_j} P^{a_m}_{s_m s_{m+1}}$ is superadditive on $S_i\times A$, where $s_m\in S_i$ and $a_m\in A$. Hence, for $s^1\geq s^2$ in S_i and $a^1\geq a^2$ in A, by Definition 4 and (B18), we have, for all $d\in Z^+$,

$$-\sum_{y=d}^{\infty}(P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m=s^1,a_m=a^1\}+P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m=s^2,a_m=a^2\})$$
$$\geq -\sum_{y=d}^{\infty}(P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m=s^1,a_m=a^2\}+P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m=s^2,a_m=a^1\}).$$

By Lemma 2 and (B15), $-R(s_{m+1})$ is increasing in $s_{m+1}\in S_j$, $j=1,2,\cdots,n$. Applying Lemma B1, we have

$$\sum_{y=0}^{\infty}(P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m=s^1,a_m=a^1\}+P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m=s^2,a_m=a^2\})R(q_{m+1}=r_j,d_{m+1}=y)$$
$$\geq$$
$$\sum_{y=0}^{\infty}(P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m=s^1,a_m=a^2\}+P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m=s^2,a_m=a^1\})R(q_{m+1}=r_j,d_{m+1}=y). \tag{B19}$$

We know

$$\sum_{s_{m+1}\in S_j} P^{a_m}_{s_m s_{m+1}} R(s_{m+1}) = \sum_{y=0}^{\infty} P\{q_{m+1}=r_j,d_{m+1}=y\mid s_m,a_m\}R(q_{m+1}=r_j,d_{m+1}=y).$$

Therefore (B19) shows that $\sum_{s_{m+1}\in S_j} P^{a_m}_{s_m s_{m+1}} R(s_{m+1})$ is superadditive on $S_i\times A$, where $s_m\in S_i$ and $a_m\in A$.

Since all the components of $V(s_m, a_m)$ in (B17) are superadditive and the addition of superadditive functions is still superadditive, $V(s_m, a_m)$ is superadditive on $S_i \times A$ for $m = 1,2,\cdots, M-1$. This completes the proof.

References

Puterman, M. L. 2005. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley and Sons, New York, NY.

Serfozo, R. F. 1976. Monotone optimal policies for Markov decision processes**.** *Mathematical Programming Study* 6, 202-215.

Appendix C: Proofs of Propositions

1. Proof of Proposition 1

By Theorem 1, optimal action a_m^* is increasing in $s_m \in S_i$ for $m = 1,2,\cdots,M$, where S_i is a sub-space, $i = 1,2,\cdots,n$ and actions are ordered according to Definition 1. For $s_m \in S_i$, by Definitions 2 and 3, the increasing of s_m implies the increasing of d_m and vice versa because system states belonging to a sub-space have same request type. Hence, within each sub-space, optimal action a_m^* is increasing in d_m for $m = 1,2,\cdots,M$. Considering that the action space consists of only two actions, Proposition 1 follows.

2. Proof of Proposition 2

(2.1) Proof of Proposition 2(a)

At time m, $m = 1,2,\cdots,M$, optimal actions for all system states $s_m = (q_m, d_m)$ satisfying $q_m = r_i$ and $d_m \geq k_{m,i}$, $i = 1,2,\cdots,n$, are the same (i.e., running KDD), according to Proposition 1. By (9) and (6), we have

$$J(s_m) = c^k + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid s_m, a_m = 1\} J(s_{m+1}). \qquad \text{if } q_m = r_i \text{ and } d_m \geq k_{m,i} \qquad \text{(C1)}$$

By observing (5), $P\{s_{m+1} \mid s_m, a_m = 1\}$ is independent of s_m. Consequently, $J(s_m)$ in (C1) is independent of s_m. Therefore, $J(s_m)$ for all system states $s_m = (q_m, d_m)$ satisfying $q_m = r_i$ and $d_m \geq k_{m,i}$, $i = 1,2,\cdots,n,$ are the same.

(2.2) Proof of Proposition 2(b)

$$\begin{aligned} Q(s_m, a_m) &= c(s_m, a_m) + \sum_{s_{m+1} \in S} P_{s_m s_{m+1}}^{a_m} J(s_{m+1}) && \text{by (11)} \\ &= c(s_m, a_m) + \sum_{i=1}^{n} \sum_{s_{m+1} \in S_i} P_{s_m s_{m+1}}^{a_m} J(s_{m+1}) && \text{by Definition 3, } S = \bigcup_{i=1}^{n} S_i \qquad \text{(C2)} \end{aligned}$$

Let $s_{m+1}=(q_{m+},d_{m+1})$ and $s_m=(q_m,d_m)$. By Definition 3, $s_{m+1}\in S_i$ implies $q_{m+1}=r_i$. (C2) can be rewritten as

$$Q(s_m,a_m)=c(s_m,a_m)+\sum_{i=1}^{n}\sum_{d=0}^{\infty}P\{q_{m+1}=r_i,d_{m+1}=d\mid q_m,d_m,a_m\}J(q_{m+1}=r_i,d_{m+1}=d). \qquad \text{(C3)}$$

By Proposition 2(a), $J(s_{m+1})$ for all system states $s_{m+1}=(q_{m+1},d_{m+1})$ satisfying $q_{m+1}=r_i$ and $d_{m+1}\ge k_{m+1,i}$, $i=1,2,\cdots,n,$ are the same. Randomly picking a system state $s_{m+1}=(q_{m+1},d_{m+1})$ satisfying $q_{m+1}=r_i$ and $d_{m+1}\ge k_{m+1,i}$, say $q_{m+1}=r_1$ and $d_{m+1}=k_{m+1,1}$. $J(s_{m+1})$ for all system states $s_{m+1}=(q_{m+1},d_{m+1})$ satisfying $q_{m+1}=r_i$ and $d_{m+1}\ge k_{m+1,i}$, $i=1,2,\cdots,n,$ is equal to $J(q_{m+1}=r_1,d_{m+1}=k_{m+1,1})$. Let P_k be the probability of system state at time $m+1$, $s_{m+1}=(q_{m+1},d_{m+1}),$ satisfying $q_{m+1}=r_i$ and $d_{m+1}\ge k_{m+1,i}$, $i=1,2,\cdots,n,$ given s_m,a_m. P_k can be calculated using (4) and (5). We can rewrite (C3) as

$$\begin{aligned}Q(s_m,a_m)=c(s_m,a_m)&+\sum_{i=1}^{n}\sum_{d=0}^{k_{m+1,i}-1}P\{q_{m+1}=r_i,d_{m+1}=d\mid q_m,d_m,a_m\}J(q_{m+1}=r_i,d_{m+1}=d)\\&+P_kJ(q_{m+1}=r_1,d_{m+1}=k_{m+1,1}). \qquad \text{(C4)}\end{aligned}$$

By observing (C4), the time complexity of computing $Q(s_m,a_m)$ is $O(k_{m+1})$ where $k_{m+1}=\sum_{i=1}^{n}k_{m+1,i}$.

This completes the proof.

3. Proof of Proposition 3

(3.1) Proof of Proposition 3(a)

By (11), (6), and (7), we have,

$$Q(q_m=r_i,d_m=d,a_m=0)=c_{r_i}(F(d))^{\phi}+\sum_{s_{m+1}\in S}P\{s_{m+1}\mid q_m=r_i,d_m=d,a_m=0\}J(s_{m+1});$$

$$Q(q_m=r_j,d_m=d,a_m=0)=c_{r_j}(F(d))^{\phi}+\sum_{s_{m+1}\in S}P\{s_{m+1}\mid q_m=r_j,d_m=d,a_m=0\}J(s_{m+1}).$$

By (4), $P\{s_{m+1} \mid q_m, d_m = d, a_m = 0\}$ is independent of q_m. Hence

$$Q(q_m = r_j, d_m = d, a_m = 0) - c_{r_j}(F(d))^{\phi} = Q(q_m = r_i, d_m = d, a_m = 0) - c_{r_i}(F(d))^{\phi},$$

and Proposition 3(a) follows.

(3.2) Proof of Proposition 3(b)

By (11), (6), (7), and (4), we have

$$\begin{aligned}
&Q(q_m = r_i, d_m = d, a_m = 0) \\
&= c_{r_i}(F(d))^{\phi} + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_i, d_m = d, a_m = 0\} J(s_{m+1}) \\
&= c_{r_i}(F(d))^{\phi} + \sum_{l=1}^{n} \frac{\lambda_l}{(\mu + \sum_{j=1}^{n} \lambda_j)} J(q_{m+1} = r_l, d_{m+1} = d) \\
&+ \frac{\mu}{(\mu + \sum_{j=1}^{n} \lambda_j)} \underbrace{\sum_{l=1}^{n} [\frac{\lambda_l}{(\mu + \sum_{j=1}^{n} \lambda_j)} J(q_{m+1} = r_l, d_{m+1} = d+1) + \ldots + \frac{\lambda_l}{\sum_{j=1}^{n} \lambda_j} (\frac{\mu}{\mu + \sum_{j=1}^{n} \lambda_j})^{k_{m+1,l} - d - 1} J(q_{m+1} = r_l, d_{m+1} = k_{m+1,l})]}_{\nabla} \quad \text{(C5)}
\end{aligned}$$

Similarly,

$$\begin{aligned}
&Q(q_m = r_i, d_m = d+1, a_m = 0) \\
&= c_{r_i}(F(d+1))^{\phi} + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_i, d_m = d+1, a_m = 0\} J(s_{m+1}) \\
&= c_{r_i}(F(d+1))^{\phi} + \underbrace{\sum_{l=1}^{n} [\frac{\lambda_l}{(\mu + \sum_{j=1}^{n} \lambda_j)} J(q_{m+1} = r_l, d_{m+1} = d+1) + \ldots + \frac{\lambda_l}{\sum_{j=1}^{n} \lambda_j} (\frac{\mu}{\mu + \sum_{j=1}^{n} \lambda_j})^{k_{m+1,l} - d - 1} J(q_{m+1} = r_l, d_{m+1} = k_{m+1,l})]}_{\nabla}. \quad \text{(C6)}
\end{aligned}$$

By (C6), we can substitute ∇ in (C5) using $Q(q_m = r_i, d_m = d+1, a_m = 0) - c_{r_i}(F(d+1))^{\phi}$. We thus have

$$\begin{aligned}
&Q(q_m = r_i, d_m = d, a_m = 0) \\
&= c_{r_i}(F(d))^{\phi} + \sum_{l=1}^{n} \frac{\lambda_l}{(\mu + \sum_{j=1}^{n} \lambda_j)} J(q_{m+1} = r_l, d_{m+1} = d) \\
&\quad + \frac{\mu}{(\mu + \sum_{j=1}^{n} \lambda_j)} [Q(q_m = r_i, d_m = d+1, a_m = 0) - c_{r_i}(F(d+1))^{\phi}]. \qquad \text{(C7)}
\end{aligned}$$

It is easy to derive Proposition 3(b) from (C7).

(3.3) Proof of Proposition 3(c)

$$
\begin{aligned}
&Q(q_m = r_j, d_m = d, a_m = 1) \\
&= c^k + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_j, d_m = d, a_m = 1\} J(s_{m+1}) \qquad \text{by (11), (6), (7)} \\
&= c^k + \sum_{l=1}^{n} \sum_{v=0}^{\infty} P\{q_{m+1} = r_l, d_{m+1} = v \mid q_m = r_j, d_m = d, a_m = 1\} J(q_{m+1} = r_l, d_{m+1} = v) \\
&= c^k + \underbrace{\sum_{l=1}^{n} \sum_{v=0}^{\infty} \frac{\mu^v \lambda_l}{(\mu + \sum_{k=1}^{n} \lambda_k)^{v+1}} J(q_{m+1} = r_l, d_{m+1} = v)}_{} \qquad \text{by (5)} \qquad \text{(C8)}
\end{aligned}
$$

$$
\begin{aligned}
&Q(q_m = r_i, d_m = 0, a_m = 0) \\
&= c_{r_i} (F(d_m = 0))^{\phi} + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_i, d_m = 0, a_m = 0\} J(s_{m+1}) \qquad \text{by (11), (6), (7)}
\end{aligned}
$$

In this study, knowledge loss is the result of incoming new data. If there is no new data, i.e., $d_m = 0$, there is no knowledge loss, i.e., $F(d_m = 0) = 0$. Hence,

$$
\begin{aligned}
&Q(q_m = r_i, d_m = 0, a_m = 0) \\
&= \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_i, d_m = 0, a_m = 0\} J(s_{m+1}) \\
&= \sum_{l=1}^{n} \sum_{v=0}^{\infty} P\{q_{m+1} = r_l, d_{m+1} = v \mid q_m = r_i, d_m = 0, a_m = 0\} J(q_{m+1} = r_l, d_{m+1} = v) \\
&= \underbrace{\sum_{l=1}^{n} \sum_{v=0}^{\infty} \frac{\mu^v \lambda_l}{(\mu + \sum_{k=1}^{n} \lambda_k)^{v+1}} J(q_{m+1} = r_l, d_{m+1} = v)}_{} \qquad \text{by (4)} \qquad \text{(C9)}
\end{aligned}
$$

Comparing (C8) and (C9), Proposition 3(c) follows.

4. Proof of Proposition 4

By Proposition 1, at time m, $m = 1,2,\cdots,M$, for request type r_i, $i = 1,2,\cdots,n$, $k_{m,i}$ equals the minimum d_m satisfying $Q(s_m, a_m = 0) > Q(s_m, a_m = 1)$, where $s_m = (q_m, d_m)$ and $q_m = r_i$. By (11), (6) and (7), $k_{m,i}$ equals the minimum d_m satisfying,

$$c_{r_i}(F(d_m))^{\phi} + \sum_{s_{m+1}\in S} P_{s_m s_{m+1}}^{a_m=0} J(s_{m+1}) > \; c^k + \sum_{s_{m+1}\in S} P_{s_m s_{m+1}}^{a_m=1} J(s_{m+1}). \tag{C10}$$

By Definition 3, $S = \bigcup_{j=1}^{n} S_j$, and we can rewrite (C10) as,

$$c_{r_i}(F(d_m))^{\phi} + \sum_{j=1}^{n} \sum_{s_{m+1}\in S_j} P_{s_m s_{m+1}}^{a_m=0} J(s_{m+1}) > \; c^k + \sum_{j=1}^{n} \sum_{s_{m+1}\in S_j} P_{s_m s_{m+1}}^{a_m=1} J(s_{m+1}). \tag{C11}$$

Let $s_m = (q_m, d_m)$ and $s_{m+1} = (q_{m+1}, d_{m+1})$. By (4) and (5), we can rewrite (C11) as,

$$\begin{aligned} c_{r_i}(F(d_m))^{\phi} + \sum_{j=1}^{n} \sum_{d=d_m}^{\infty} \frac{\mu^{d-d_m}\lambda_j}{(\mu + \sum_{l=1}^{n}\lambda_l)^{d-d_m+1}} J(q_{m+1} = r_j, d_{m+1} = d\,) \; > \\ c^k + \sum_{j=1}^{n} \sum_{g=0}^{\infty} \frac{\mu^{g}\lambda_j}{(\mu + \sum_{l=1}^{n}\lambda_l)^{g+1}} J(q_{m+1} = r_j, d_{m+1} = g\,) \end{aligned} \tag{C12}$$

Let $z = d - d_m$ and (C12) becomes,

$$\begin{aligned} c_{r_i}(F(d_m))^{\phi} + \sum_{j=1}^{n} \sum_{z=0}^{\infty} \frac{\mu^{z}\lambda_j}{(\mu + \sum_{l=1}^{n}\lambda_l)^{z+1}} J(q_{m+1} = r_j, d_{m+1} = z + d_m\,) \; > \\ c^k + \sum_{j=1}^{n} \sum_{g=0}^{\infty} \frac{\mu^{g}\lambda_j}{(\mu + \sum_{l=1}^{n}\lambda_l)^{g+1}} J(q_{m+1} = r_j, d_{m+1} = g\,) \end{aligned} \tag{C13}$$

Rewriting (C13), $k_{m,i}$ equals the minimum d_m satisfying,

$$c_{r_i}(F(d_m))^{\phi} + \sum_{j=1}^{n} \sum_{z=0}^{\infty} \frac{\mu^{z}\lambda_j}{(\mu + \sum_{l=1}^{n}\lambda_l)^{z+1}} \left[J(\, q_{m+1} = r_j, d_{m+1} = z + d_m) - J(q_{m+1} = r_j, d_{m+1} = z) \right] > c^k. \tag{C14}$$

Recall that $J(s_{M+1}) = 0$ for all $s_{M+1} \in S$. Hence, for $m = M$, by (C14), $k_{M,i}$ equals the minimum d_M satisfying $c_{r_i}(F(d_M))^{\phi} > c^k$. We have

$$c_{r_i}(F(k_{M,i}))^{\phi} > c^k. \tag{C15}$$

For $m=1,2,\cdots,M-1$, let

$$h(d_m)=\sum_{j=1}^{n}\sum_{z=0}^{\infty}\frac{\mu^z\lambda_j}{(\mu+\sum_{l=1}^{n}\lambda_l)^{z+1}}\left[J(\,q_{m+1}=r_j,d_{m+1}=z+d_m)-J(q_{m+1}=r_j,d_{m+1}=z)\right].$$

By Lemma 2, $J(q_{m+1}=r_j,d_{m+1}=z+d_m)\geq J(q_{m+1}=r_j,d_{m+1}=z)$. Hence, $h(d_m)\geq 0$ for $m=1,2,\cdots,M-1$.

For $m=1,2,\cdots,M-1$, let $d_m=k_{M,i}$ and we have

$$\begin{aligned} c_{r_i}(F(d_m=k_{M,i}))^{\phi}+h(d_m=k_{M,i}) &\geq c_{r_i}(F(k_{M,i}))^{\phi} && \text{by } h(d_m)\geq 0 \text{ for } m=1,2,\cdots M-1\\ &> c^k. && \text{by (C15)}\end{aligned}$$

Hence, for $m=1,2,\cdots,M-1$, (C14) is satisfied when $d_m=k_{M,i}$. Since $k_{m,i}$ equals the minimum d_m satisfying (C14) for $m=1,2,\cdots,M-1$, we have $k_{m,i}\leq k_{M,i}$ for $m=1,2,\cdots,M-1$.

5. Proof of Proposition 5

Let $s_m=(q_m,d_m)$. By Proposition 1, at time m, $m=1,2,\cdots,M$, the threshold for request type q_m equals the minimum d_m satisfying $Q(s_m,a_m=0)>Q(s_m,a_m=1)$. By (11), (6) and (7), the threshold for q_m equals the minimum d_m satisfying

$$\begin{aligned} c_{q_m}(F(d_m))^{\phi}+\sum_{s_{m+1}\in S}P\{s_{m+1}\mid q_m,d_m,a_m=0\}J(s_{m+1})>&\\ c^k+\sum_{s_{m+1}\in S}P\{s_{m+1}\mid q_m,d_m,a_m=1\}J(s_{m+1}).&\end{aligned} \tag{C16}$$

Let $s_{m+1}=(q_{m+1},d_{m+1})$. By Definition 3, $S=\bigcup_{h=1}^{n}S_h$, we can rewrite (C16) as

$$\begin{aligned} c_{q_m}(F(d_m))^{\phi}+\sum_{h=1}^{n}\sum_{s_{m+1}\in S_h}P\{s_{m+1}\mid q_m,d_m,a_m=0\}J(s_{m+1})>&\\ c^k+\sum_{h=1}^{n}\sum_{s_{m+1}\in S_h}P\{s_{m+1}\mid q_m,d_m,a_m=1\}J(s_{m+1}).&\end{aligned}$$

Applying (4) and (5), the above inequality becomes

$$c_{q_m}(F(d_m))^{\phi} + \sum_{h=1}^{n}\sum_{d=d_m}^{\infty} \frac{\mu^{d-d_m}\lambda_h}{(\mu+\sum_{l=1}^{n}\lambda_l)^{d-d_m+1}} J(q_{m+1}=r_h, d_{m+1}=d\ ) > c^k + \sum_{h=1}^{n}\sum_{d=0}^{\infty} \frac{\mu^{d}\lambda_h}{(\mu+\sum_{l=1}^{n}\lambda_l)^{d+1}} J(q_{m+1}=r_h, d_{m+1}=d\ ). \quad \text{(C17)}$$

Let $g(d_m) = \sum_{h=1}^{n}\sum_{d=d_m}^{\infty} \frac{\mu^{d-d_m}\lambda_h}{(\mu+\sum_{l=1}^{n}\lambda_l)^{d-d_m+1}} J(q_{m+1}=r_h, d_{m+1}=d\ )$, where $g(.)$ depends on d_m but is independent of q_m. Let $R = \sum_{h=1}^{n}\sum_{d=0}^{\infty} \frac{\mu^{d}\lambda_h}{(\mu+\sum_{l=1}^{n}\lambda_l)^{d+1}} J(q_{m+1}=r_h, d_{m+1}=d\ )$, where R is independent of both q_m and d_m. Rewriting (C17), the threshold for q_m equals the minimum d_m satisfying

$$c_{q_m}(F(d_m))^{\phi} + g(d_m) > c^k + R. \quad \text{(C18)}$$

By (C18), the threshold for q_m is determined by c_{q_m} not by the arrival rate of q_m .

(5.1) Proof of Proposition 5(a)

By (C18), we have

$$c_{r_j}(F(d_m = k_{m,j}))^{\phi} + g(d_m = k_{m,j}) > c^k + R.$$

Since $c_{r_i} > c_{r_j}$ and $F(d_m) \geq 0$, we have,

$$c_{r_i}(F(d_m = k_{m,j}))^{\phi} + g(d_m = k_{m,j}) \geq c_{r_j}(F(d_m = k_{m,j}))^{\phi} + g(d_m = k_{m,j}) > c^k + R.$$

The above inequality indicates that (C18) is satisfied for request type r_i when $d_m = k_{m,j}$. Since $k_{m,i}$ equals the minimum d_m satisfying (C18) for request type r_i, we have $k_{m,i} \leq k_{m,j}$.

(5.2) Proposition 5(b) is readily followed from (C18).

(5.3) Proof of Proposition 5(c)

By Proposition 5(b), if $c_{r_i} = c_{r_j}$, we have $k_{m,i} = k_{m,j}$. Let $k_{m,i} = k_{m,j} = \kappa$.

We first consider $d \geq \kappa$. By Proposition 1, optimal actions for both request types r_i and r_j are 1 if $d \geq \kappa$. Hence, according to (9),

$$J(q_m = r_i, d_m = d) = c^k + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_i, d_m = d, a_m = 1\} J(s_{m+1}) \quad \text{if } d \geq \kappa;$$

$$J(q_m = r_j, d_m = d) = c^k + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_j, d_m = d, a_m = 1\} J(s_{m+1}) \quad \text{if } d \geq \kappa.$$

By (5), $P\{s_{m+1} \mid q_m, d_m, a_m = 1\}$ is independent of q_m. Hence,

$$P\{s_{m+1} \mid q_m = r_i, d_m = d, a_m = 1\} = P\{s_{m+1} \mid q_m = r_j, d_m = d, a_m = 1\}.$$

Consequently, $J(q_m = r_i, d_m = d) = J(q_m = r_j, d_m = d)$ for $d \geq \kappa$.

If $d < \kappa$, by Proposition 1, optimal actions for both request types r_i and r_j are 0. According to (9),

$$J(q_m = r_i, d_m = d) = c_{r_i}(F(d_m = d))^{\phi} + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_i, d_m = d, a_m = 0\} J(s_{m+1}) \quad \text{if } d < \kappa;$$

$$J(q_m = r_j, d_m = d) = c_{r_j}(F(d_m = d))^{\phi} + \sum_{s_{m+1} \in S} P\{s_{m+1} \mid q_m = r_j, d_m = d, a_m = 0\} J(s_{m+1}) \quad \text{if } d < \kappa.$$

By (4), $P\{s_{m+1} \mid q_m, d_m, a_m = 0\}$ is independent of q_m. Hence,

$$P\{s_{m+1} \mid q_m = r_i, d_m = d, a_m = 0\} = P\{s_{m+1} \mid q_m = r_j, d_m = d, a_m = 0\}.$$

Since $c_{r_i} = c_{r_j}$ we have

$$c_{r_i}(F(d_m = d))^{\phi} = c_{r_j}(F(d_m = d))^{\phi}.$$

Therefore, $J(q_m = r_i, d_m = d) = J(q_m = r_j, d_m = d)$ for $d < \kappa$. This completes the proof.

Appendix D: Quantification and Estimation of Knowledge Loss When Its Determining Factors Weight Differently

Factors determining knowledge loss may weight differently. For example, the loss caused by not using emerging new knowledge may be higher than the loss attributed to the use of invalid knowledge. Let $w_1 > 0$, $w_2 > 0$, and $w_3 > 0$ be the weight for valid knowledge, invalid knowledge, and emerging new knowledge respectively. In consideration of these weights, knowledge loss l_t at time t becomes

$$l_t = \frac{w_2\left|K_t \setminus \hat{K}_t\right| + w_3\left|\hat{K}_t \setminus K_t\right|}{w_1\left|K_t \cap \hat{K}_t\right| + w_2\left|K_t \setminus \hat{K}_t\right| + w_3\left|\hat{K}_t \setminus K_t\right|}, \tag{D1}$$

where K_t denotes the knowledge one has at time t, which was discovered by the latest KDD run before time t, $\hat{K}_t$ denotes the knowledge discovered if KDD *were* run at time t, and $|\bullet|$ denotes the cardinality of a set. Equation (D1) is a general form of measuring knowledge loss while equation (1) is a special case of equation (D1) where $w_1 = w_2 = w_3 = 1$.

Given this general form of measuring knowledge loss, we conducted experiments to estimate the knowledge loss function *F*(.) for the retail knowledge refreshing problem. Our experiments followed the same procedure as that described in §5.2 except that knowledge loss was calculated using (D1) instead of (1). Two weighting scenarios were considered: (a) emerging new knowledge weighted more than the other factors; and (b) invalid knowledge weighted more. Accordingly, we set w_1, w_2, and w_3 to 1,1, and 2 respectively for scenario (a) and set them to 1,2,1 respectively for scenario (b). The data set used in the experiments consisted of 11580 transactions (i.e., 30-day of transactions conducted at the Department of Household Items) and n_i was varied from 25 to 125 in these experiments. The experimental results shown in Tables D1 and D2 suggest that the Weibull function (i.e., equation (17)) still estimates knowledge loss for the retail knowledge refreshing problem very well ($p < 0.0001, R^2 = 0.99$), when knowledge loss is calculated using (D1). The estimates of α and β listed in Tables D1 and D2 are different from those in Table 3 because knowledge loss is calculated using (D1) rather than (1).

n_i	Estimate of α	Estimate of β
25	5,635.1	0.50
	(77.6)	(0.008)
50	5,636.8	0.50
	(113.0)	(0.011)
75	5,645.4	0.49
	(136.0)	(0.014)
100	5,628.2	0.50
	(159.1)	(0.016)
125	5,694.5	0.49
	(170.4)	(0.016)

Note: Standard error of an estimate is listed in the parentheses below the estimate.

Table D1: Estimates of α and β

(Knowledge loss is calculated using (D1) with $w_1 = 1, w_2 = 1$, and $w_3 = 2$)

n_i	Estimate of α	Estimate of β
25	6,673.8	0.51
	(109.4)	(0.008)
50	6,675.8	0.51
	(156.2)	(0.012)
75	6,689.0	0.51
	(189.8)	(0.015)
100	6,651.6	0.52
	(220.6)	(0.018)
125	6,725.1	0.51
	(241.0)	(0.019)

Note: Standard error of an estimate is listed in the parentheses below the estimate.

Table D2: Estimates of α and β

(Knowledge loss is calculated using (D1) with $w_1 = 1, w_2 = 2$, and $w_3 = 1$)

Appendix E: Relaxation of Model Assumption Regarding KDD Run

The model proposed in §3.2 assumes that knowledge loss is reduced to zero instantaneously once KDD is run. Relaxing the assumption, we show how to compute knowledge refreshing policy and examine the effectiveness of the computed policy. If the assumption is relaxed, the delay between the request for running KDD and the receival of the discovered knowledge needs to be considered. The delay consists of the time of waiting for KDD service and KDD running time. Due to rapid advancement of computing power and the criticality of knowledge discovered by KDD for decision making, many organizations have provided sufficient capacity for running KDD such that waiting time for KDD service is generally negligible. Hence, we treat waiting time for KDD service as 0 and focus on delay caused by KDD running time. Let KDD running time be T_k time units. Once KDD is run in response to a request, the request will receive the discovered knowledge T_k time units later. Consequently, the discovered knowledge suffers from knowledge loss due to new data arrived during the period of T_k time units. For a knowledge request of type r_i, $i=1,2,\cdots,n$, the expected cost L_{r_i} of knowledge loss incurred during the period of running KDD is

$$L_{r_i} = c_{r_i} \sum_{j=0}^{\infty} \left(F(j)\right)^{\phi} e^{-\mu \mathrm{T}_k} \frac{(\mu \mathrm{T}_k)^j}{j!}. \qquad \text{by equations (7) and (2)} \tag{E1}$$

Here j is the amount of new data arrived during the period of running KDD. By (17), we have $F(j) = 1 - e^{-(\frac{j}{\alpha})^{\beta}}$ for the retail knowledge refreshing problem. L_{r_i} can be approximated using numerical methods since a closed-form expression cannot be found. Relaxing the assumption about KDD run, we need to change the cost under the action of running KDD from c^k to $c^k + L_{r_i}$. Knowledge refreshing policy π_2 can be computed using the algorithm shown in Figure 2 with the modified cost function.

We examine the effectiveness of π_2 by analyzing cost savings generated by π_2 over optimal fixed-interval policies. Let C_{π_2} be the total system cost resulting from implementing π_2, $C_{p_2^*}$ be the total system cost resulting from implementing p_2^* (i.e., optimal periodical policy), $C_{r_2^*}$ be the total system cost resulting from implementing r_2^* (i.e., optimal fixed-interval policy based on the number of requests), and $C_{d_2^*}$ be the total system cost resulting from implementing d_2^* (i.e., optimal fixed-interval policy based on the amount of accumulated new data)[2]. The cost differences between π_2 and the benchmark policies are measured as $\nu_1 = \frac{(C_{p_2^*} - C_{\pi_2})}{C_{p_2^*}}$, $\nu_2 = \frac{(C_{r_2^*} - C_{\pi_2})}{C_{r_2^*}}$, and $\nu_3 = \frac{(C_{d_2^*} - C_{\pi_2})}{C_{d_2^*}}$. We conducted two sets of experiments with parameter values listed in Table 2 and $M = 125$. In the first set of experiments, ϕ was set to 1 and T_k was varied from 0.1 time units to 0.5 time units; in the second set of experiments, T_k was set to 0.3 time units and ϕ was varied from 0.5 to 2. According to the experimental results reported in Tables E1-E2, π_2 substantially outperforms optimal fixed-interval policies across a range of T_k and ϕ[3]. Overall, we have shown that relaxing the assumption about KDD run only needs a simple modification to the cost function. The knowledge refreshing policy computed using the algorithm shown in Figure 2 with the modified cost function substantially outperforms optimal fixed-interval policies.

T_k	C_{π_2}	$C_{p_2^*}$	$\nu_1(\%)$	$C_{r_2^*}$	$\nu_2(\%)$	$C_{d_2^*}$	$\nu_3(\%)$
0.1	\$206,851.71	\$331,149.28	37.5	\$237,611.55	12.9	\$318,879.77	35.1
0.2	\$216,441.84	\$334,410.64	35.3	\$245,839.57	12.0	\$322,185.94	32.8
0.3	\$226,496.24	\$339,961.24	33.4	\$254,509.10	11.0	\$328,189.93	31.0
0.4	\$231,964.33	\$343,422.20	32.5	\$259,551.25	10.6	\$331,905.81	30.1
0.5	\$238,285.13	\$348,184.57	31.6	\$265,331.69	10.2	\$337,186.63	29.3

Table E1: Performance Comparisons Between π_2 and Optimal Fixed-interval Policies (ϕ=1)

[2] The cost incurred during the period of running KDD is also considered for the benchmark policies.
[3] Each result reported in Tables E1-E2 represents an average of 200 experimental runs.

ϕ	C_{π_2}	$C_{p_2^*}$	$\nu_1(\%)$	$C_{r_2^*}$	$\nu_2(\%)$	$C_{d_2^*}$	$\nu_3(\%)$
0.5	\$422,561.95	\$556,620.75	24.1	\$452,943.08	6.7	\$545,065.95	22.5
0.75	\$298,850.32	\$429,893.10	30.5	\$329,422.26	9.3	\$417,662.45	28.4
1	\$226,496.24	\$339,961.24	33.4	\$254,509.10	11.0	\$328,189.93	31.0
1.5	\$149,421.49	\$225,286.80	33.7	\$173,316.16	13.8	\$215,828.78	30.8
2	\$112,401.52	\$160,827.58	30.1	\$130,752.73	14.0	\$153,533.61	26.8

Table E2: Performance Comparisons Between π_2 and Optimal Fixed-interval Policies (T_k =0.3)

Tackling the Velocity of Big Data: Optimal Policy for Refreshing Knowledge Discovered from Data

Knowledge extracted using a broad variety of data mining techniques from massive amounts of consumer behavioral, social networking, business transaction, law enforcement or clinical data has increased the value of behavior targeting marketing campaigns, improved the quality of health, customer and government services, and facilitated supply chain and other day-to-day management decision making. To sustain these data driven values, Google and IBM, for example, have indicated that one of the main issues is to maintain the currency of knowledge over evolving customer and transaction data. In "When Is the Right Time to Refresh Knowledge Discovered from Data?", X. Fang, O. R. Liu Sheng, and P. Goes address this issue from the perspective of deciding on when to re-discover knowledge – namely the problem of knowledge refreshing. The paper has two main managerial implications. First, the in-depth analysis of knowledge loss provides knowledge-driven organizations the understanding of the need and the foundation for managing knowledge loss. Second, the paper presents a practical, automated knowledge management tool to decide on when to refresh discovered knowledge by simultaneously minimizing knowledge loss and knowledge refreshing cost. The tool will greatly sustain the value of fast growing data mining applications.