On Efficient Replacement Policies for Cache Objects with Non-uniform Sizes and Costs

Ying Su
Computer Science
University of British Columbia
Email: suying00@gmail.com

Laks V.S. Lakshmanan
Computer Science
University of British Columbia
Email: laks@cs.ubc.ca

***Abstract*—Replacement policies for cache management have been studied extensively. Most policies proposed in the literature tend to be ad hoc and typically exploit the retrieval cost or latency of the objects, object sizes, popularity of requests and temporal correlations between requests. A recent paper [1] studied the caching problem and developed an optimal replacement policy C_0^* under the independent reference model (IRM), assuming non-uniform object retrieval costs. In this paper, we consider the more general setting where *both object sizes and their retrieval costs are non-uniform*. This setting arises in a number of applications such as web caching and view management in databases and in data warehouses.**

We consider static selection as a benchmark when evaluating the performance of replacement policies. Our first result is negative: no dynamic policy can achieve a better performance than the optimal static selection in terms of long run average metric. We also prove that a (dynamic) replacement policy attains this optimum iff the stochastic chain induced by it is irreducible. We show that previously studied optimal policies such as A_0 and C_0^* are special cases of our optimal policy.

This motivates the study of static selection. For the general case we are considering, static selection is NP-complete. Let K denote the maximum cache capacity and let K' be the sum of sizes of all objects minus the cache capacity K. We propose a polynomial time algorithm that is both K-approximate w.r.t. the fractional optimum solution and $H_{K'}$-approximate w.r.t. the integral optimum solution, where $H_{K'}$ is the K'-th Harmonic number. In addition, we develop a K-competitive dynamic policy and show that K is the best possible approximation ratio for both static algorithms and dynamic policies.

I. Introduction

Choice of a proper replacement policy is key to the performance of any caching system, be it a regular paging system, file caching, or web caching, or the system managing views in the cache of a data warehouse application. A static version of the caching problem asks what is the best set of objects to cache subject to a total cache capacity so that the expected retrieval cost over any sequence of requests obeying a workload distribution is minimized. Clearly, in this problem, termed as static selection, once an object is cached, it stays in the cache. The static selection is widely used in the view management problem in database community. On the other hand, dynamic caching policies, like a web proxy, adjusts the cache content based on the information such as the objects' sizes, their retrieval costs, popularity of reference, etc. while serving the requests. Efficient caching policies play an important role in achieving better performance. The problem of finding good policies (for any dynamic caching system) can be described as below:

> Given a set F of N files $f_1, f_2, \ldots, f_N$, where each file f_i has the following specified values
> - size $|f_i|$;
> - retrieval cost from remote server e_i;
> - retrieval cost from the cache r_i;
> - and a probability estimation β_i (subject to $\sum_{i=1}^{n} \beta_i = 1$)
>
> and a sequence of file requests, maintain a cache with capacity K so as to satisfy the requests while minimizing the total (or average) retrieval cost.

Note that the probability estimation about requests is optional and applies when the so-called Independent Reference Model [8] is assumed for object requests.

The problem definition can be easily adapted to allow other performance metric. E.g., to use the hit rate metric, we just need to set e_i as 1 and r_i as 0; to use the byte hit rate metric, set e_i as the size $|fi|$ and r_i as 0. However in this paper we mainly consider the user-perceived retrieval cost(latency) criterion for the requests since it is a most commonly used and generalized criterion.

Belady showed an optimal offline algorithm B_0 for the paging problem in his seminal work [3], in which uniform size and retrieval cost are assumed. The algorithm assumes the future is known and evicts the page with the longest forward distance, i.e., will be accessed furthest into the future. B_0 is optimal for every request sequence. But to the best of our knowledge, there is no simple algorithm known to be optimal for the case where the objects can have arbitrary sizes and arbitrary costs.

To study the performance of paging algorithms in terms of expected cost, Coffman and Denning introduced the Independent Reference Model (IRM) [8]. It was a simple probability model under which optimal policies (in terms of expected cost) can be found. The model assumes the request sequence $q_1 q_2 \cdots q_t \cdots$ is a sequence of independent random variables with the common, stationary distribution $\phi(\cdot) = \beta_1, \beta_2, \ldots, \beta_N$ such that $P[q_t = f_i] = \beta_i$ for all $t \geq 1$. Though not a realistic representation of request patterns, it provides insights for analyzing algorithmic behavior and designing algorithms under non-stationary models.

For the uniform sizes and uniform costs case, Coffman and Denning presented a policy called A_0 which is optimal in terms of fault rate [8]. A_0 prescribes: When the cache is full,

$$\text{Evict } f_i \text{ if } i = arg_i \min(\beta_i : f_i \text{ in cache})$$

A_0 is optimal only for the paging problem, where the requested objects share the same size and retrieval cost. But for general caching problem such as web caching, both the object sizes and costs are not guaranteed to be equal and the distinction directly affects the effectiveness of the policies. In many cases, the retrieval costs from remote servers are neither uniform nor proportional to the size due to factors like the network traffic, computational ability of the remote server, etc. Bahat and Makowski [1] fully recognized the distinction caused by arbitrary retrieval costs and showed a policy C_0^* that can minimize the expected cost when the objects have uniform sizes and arbitrary costs:

$$\text{Evict } f_i \text{ if } i = arg \min(\beta_i c_i : f_i \text{ in cache or requested})$$

Unlike the algorithms used in hierarchical paged memory systems, C_0^* assumes optional admission, which is well suited for applications like web caching. Since policies with mandatory admission are a special case of the policies with optional admission, the latter must be no worse than the former.

Bahat also discussed the problem of finding optimal policies with multi-criteria aspects, for example, they studied the problem of minimizing an average cost under a single constraint in terms of another long-run average metric such as the miss rate. However, this extension cannot solve the *arbitrary sizes, arbitrary costs* case because the total size of the chosen objects cannot exceed a certain bound at any time, which is a stronger constraint than a single long run constraint. As a matter of fact, the size factor is important to web caching. Fiat and Rosen [9] showed that caching policies adapted from memory management applications that don't take size into consideration do not work well in practice. In this paper we take the study of this problem one step further by considering the case where both the sizes and costs are arbitrary. We make the following contributions:

1) We will show that for any replacement policy (random or deterministic) under the independent reference model, the optimal long-run average metric cannot be better than that of the static selection algorithm.
2) We then use the Markov Decision Process (MDP) framework to represent the problem and present a branch of history dependent replacement policy that is optimal in terms of the long-run average metric for objects with arbitrary sizes and costs, and show that policies A_0 and C_0 are special cases of this policy. We also show that a policy is optimal if and only if the induced stochastic process chain is a unichain that converges to the result of optimal static selection algorithm.
3) We study both the optimal structure and competitive structure under the IRM. We develop K-approximate static algorithms and replacement policies with regard to the fractional optimum solution and show that K is the best approximation ratio for both any replacement policies and static selection algorithms. Here K is the size of the cache. The algorithm is also $H_{K'}$-approximation to the integral optimum solution, where K' is the total size of all objects $(f_1, \cdots, f_N)$ minus K and $H_{K'}$ is the K'th harmonic number.

The paper is organized as follows: In section II, we will discuss the static selection problem and the relation between cost and benefit. Then we will present an Markov Decision Process (MDP) framework for the caching problem and discuss the relationships between the two problems. Then we present optimal policies for the *arbitrary sizes, arbitrary costs* case in section II and show that policy A_0 and C_0^* are special cases of it. In the next section, we will present our approximation algorithms for static selection and for replacement policies with an optimal approximation ratio K with regard to the fractional optimal value and $H_{K'}$ with regard to the integral optimal value where K' is $\sum_{i=1}^{N} |f_i| - K$. In section IV, we empirically evaluate the performance of the algorithms and replacement policies proposed here and compare them with well known replacement policies in the literature under a variety of settings. Our findings corroborate our analytical results. We conclude the paper in section V.

For lack of space, the proofs and some deductions of our analytical results in this paper have been omitted. They can be found in [17] or [18].

II. Optimal Algorithms Under IRM

The web proxy server services the requests of its clients by forwarding requests to remote servers and maintaining a dedicated disk space which we call cache to replicate web objects. When a request arrives, the system would first check the cache. When a requested file is not cached, we must decide whether this object is worth caching and whether some objects in the cache need to be evicted, and if yes, which ones. This can be seen as a sequential decision making process and hence can be precisely modeled by the MDP framework.

Note that above considerations apply just as well to a cache management system in a data warehouse. The cache manager has to make a decision whether to admit (evict) a view into (from) the cache in case of faults. We refer the reader to [13] [16] [12], etc. for related work in the data warehousing context. In our discussions henceforth, we use the term object to refer to any object that may be cached, be it a view or a document, or a file.

A. The MDP Framework

The MDP described in [1] assumed uniform sizes, so the decision at any time evicts at most one object. We use a slightly different MDP to incorporate the arbitrary sizes.

1) Decision Epochs: Decision epochs are defined as the instants when requests are posed. In our problem we assume infinite requests, so the set of decision epochs $\mathbf{T} = \{1, 2, \ldots\}$. Elements of $\mathbf{T}$ will be denoted by t and usually referred to as "time t".

2) State Sets: We define the system state θ_t at time t to be a pair (S_t, q_t) composed of the cache content information S_t and the request q_t. S_t is a subset of F with the constraint that its total size cannot exceed K: $\sum_{f_i \in S_t} |f_i| \leq K$. The *state space* $\mathbf{\Theta}$ is defined as $\{\mathbf{S} \times \mathbf{Q}\}$, where $\mathbf{S}$ is the space of all cache contents satisfying the space constraint, and $\mathbf{Q}$ is the space of all requests, in our case equal to F. Assuming N is finite, $\mathbf{\Theta}$ has finite number of elements, meaning that our MDP is a finite state MDP.

The MDP framework we propose consists of $|\mathbf{S}|N$ states, and these states are grouped into $|\mathbf{S}|$ sets. We call such a set a *group* defined as follows:

Group:A *group* of states is a set of states sharing the same cache content.

Each group contains N states corresponding to the N possible distinct requests. Let G_S denote the set of all states θ with cache content S, $G_S = \{\theta : \theta = (S, f_i), i = 1, 2, \ldots, N\}$.

3) Action Sets: By *actions*, we mean the steps taken when a fault occurs and there is not enough space to admit the requested object in the cache. For such a case, the system may decide to admit and evict some objects. We denote the set of admitted objects at time t as X_t and the set of evicted objects as Y_t, and the action A_t is composed of X_t and Y_t. Unlike [1], both X_t and Y_t in our system may contain multiple items, meaning that it is allowed to admit or evict multiple objects. If at any time t, X_t contains at most one object q_t, the policy is called a *demand policy* [8]. Given the state $\theta_t = (S_t, q_t)$ the resulting state θ_{t+1} can be written as

$$\begin{aligned} \theta_{t+1} &= (S_{t+1}, q_{t+1}) \qquad (1) \\ &= \begin{cases} (S_t, q_{t+1}), & q_t \in S_t \\ (S_t + q_t, q_{t+1}), & q_t \notin S_t, |q_t| \leq K - |S_t| \\ (S_t + X_t - Y_t, q_{t+1}), & q_t \notin S_t, |q_t| > K - |S_t| \end{cases} \end{aligned}$$

Here, $|S_t + X_t - Y_t| \leq K$.

In our MDP system the space for either X_t or Y_t is a subset of $\mathbf{S}$ and is finite, so the action space is also finite.

4) Rewards and Transition Probabilities: It is easily seen that for any state, when an action is made, the system would transfer to one of the N states in one (not necessarily a different one) group with the probability $\beta_1, \beta_2, \ldots, \beta_N$ defined in the Independent Reference Model respectively. Suppose at time t the system is in state θ and action a is taken, $\theta = (S_t, q_t)$, the transition probability from state θ to state θ' for action a of policy π is:

$$p_t^\pi(\theta' \mid \theta, a) = \beta_j \qquad (2)$$

where the state $\theta' = (S_{t+1}, f_j)$ is composed of cache content S_{t+1} and request f_j.

Similarly, the one step reward $w_t^\pi(\theta, a)$ and one step cost $c_t^\pi(\theta, a)$ for policy π are defined as the following:

$$w_t^\pi(\theta, a) = \begin{cases} e_i - r_i & \text{if } q_t \in S_t, q_t = f_i \\ 0 & \text{if } q_t \notin S_t \end{cases} \qquad (3)$$

$$c_t^\pi(\theta, a) = \begin{cases} r_i & \text{if } q_t \in S_t, q_t = f_i \\ e_i & \text{if } q_t \notin S_t \end{cases} \qquad (4)$$

If the request q_t causes a hit, we say that the system earns a reward amounting to $e_i - r_i$, or 0 if it is a fault. Alternatively we could say that a hit causes the system a cost r_i and a fault causes a cost e_i.

We can see that the transition probability, the reward and the cost are only relevant to the states but not the action and the time, so we can write them as $p^\pi(\theta' \mid \theta)$, $w^\pi(\theta)$ and $c^\pi(\theta)$.

B. The Objective Value of a Policy

In classic file cache management, we are mostly interested in the expected total cost or average cost for every state θ. However in our system the total cost is equal to ∞ for all policies, thus it cannot distinguish the policies. Therefore we will mainly focus on minimizing the average cost:

$$ac^\pi(\theta) = \lim_{T \to \infty} \frac{1}{T} E_\theta^\pi \{\sum_{t=1}^{T} c(\theta_t)\} \qquad (5)$$

or equivalently, maximizing the average reward:

$$aw^\pi(\theta) = \lim_{T \to \infty} \frac{1}{T} E_\theta^\pi \{\sum_{t=1}^{T} w(\theta_t)\} \qquad (6)$$

In order to calibrate the performance of the policies, we consider the static selection as a benchmark. As introduced in section I, the static selection problem is to select the best objects (files, documents, or views) subject to a cache capacity such that the expected cost C_S of answering a request using cache content S is minimized, or the expected benefit B_S is maximized. In such a setting, C_S and B_S can be mathematically described as below:

$$C_S = \sum_{f_i \in S} \beta_i r_i + \sum_{f_i \notin S} \beta_i e_i \qquad (7)$$

$$B_S = \sum_{f_i \in S} \beta_i (e_i - r_i) \qquad (8)$$

In the database community, [12] and many later papers [11] [10] presented static algorithms that can achieve a competitive ratio of the optimal benefit. In this paper, we would like to pin down the relative performance of static selection and dynamic replacement policies. Our first result is:

Theorem 1: $aw^{OPT}(\theta) \leq B_{max}$, and $ac^{OPT}(\theta) \leq C_{min}$ for any starting state θ, where $aw^{OPT}(\theta)$ and $ac^{OPT}(\theta)$ are the average reward and cost of the optimal policy OPT, B_{max} is the maximum expected benefit, and C_{min} is the minimum expected cost of the optimal static selection algorithm.

To prove the above theorem, we need to compute the average reward or cost of the MDP policies. Since computing the average reward is equivalent to computing the average cost, we will only consider average reward henceforth in this section. The limit in average reward might diverge in some systems, but when the state space and action space are finite or countable, which is exactly our case, the limit always exists under some stationary randomized policy π [7] [15], and the average reward or cost can be calculated [7] [2] [14] using the formula $aw^\pi(\theta) = P_*^\pi w^\pi(\theta)$, where

$w^\pi(\theta)$ is the one step reward for state θ under the stationary policy π, and P_*^π is the Cesaro limit of the transition matrix: $P_*^\pi = \lim_{M\to\infty}\frac{1}{M}\sum_{t=0}^{M-1}P_{(t)}^\pi$, where $P_{(t)}^\pi$ is the t step transition matrix.

Since the states in our MDP is finite, P_*^π must exist and be stochastic [14]. As $w^\pi(\theta)$ is known according to the definition in eq. (3), the only work left is to obtain P_*^π. P_*^π looks different when the induced chain has different structures, namely, irreducible, multichain, and unichain [15] [2] [14] and these three categories cover the whole chain space. Let us look at them respectively:

1) Irreducible: Suppose a stationary policy π induces an irreducible Markov chain. Because the chain is just one communicating class, the Cesero limit matrix P_*^π for such a chain is stochastic and its element $p_*^\pi(\theta'|\theta)$ which represents the limit transition probability from state θ to θ' must be strictly greater than 0 [14].

The chain consists of $|\mathbf{S}|$ groups. Now order the states in the transition matrix first by groups, then in each group by the requests of the states in that group from f_1 to f_N. For each group, the limiting distribution for the N states in it for any starting state must be linearly dependent with the vector $(\beta_1, \beta_2, \dots, \beta_N)$. This is because no matter what decisions are made at time t, the system would always transfer to some group (not necessarily different from the current one) with probability β_1 to β_N. So the i^{th} row of the limiting matrix P_*^π which represents the transition probability from state θ to all states is in the following form:

$$\begin{aligned}&[(a_{i1}\beta_1, a_{i1}\beta_2, \dots, a_{i1}\beta_N,), (a_{i2}\beta_1, a_{i2}\beta_2, \dots, a_{i2}\beta_N,),\\ &\qquad \dots, (a_{i|\mathbf{S}|}\beta_1, a_{i|\mathbf{S}|}\beta_2, \dots, a_{i|\mathbf{S}|}\beta_N,)]\end{aligned}$$

subject to $\sum_{k=1}^{|\mathbf{S}|} a_{ik} = 1, a_{ik} > 0$, where a_{ik} is the coefficient for row i and the k^{th} group.

Order the states $\theta \in \mathbf{\Theta}$ in the vector $w^\pi(\theta)$ in the same way so that the resulting vector is in the form:

$$\begin{aligned}&[(w^\pi(\theta^{11}), w^\pi(\theta^{21}), .., w^\pi(\theta^{N1})), (w^\pi(\theta^{12}), w^\pi(\theta^{22}), ..,\\ &\qquad w^\pi(\theta^{N2})), .., (w^\pi(\theta^{1|\mathbf{S}|}), w^\pi(\theta^{2|\mathbf{S}|}), .., w^\pi(\theta^{N|\mathbf{S}|}))]\end{aligned} \tag{9}$$

where θ^{jk} means the j^{th} state in the k^{th} group. We will call this form of the one step rewards vector the *canonical form* in the future. So the average reward for starting state θ can be calculated as

$$\begin{aligned} aw^\pi(\theta) &= P_*^\pi w^\pi(\theta)\\ &= \sum_{j=1,k=1}^{j=N,k=|\mathbf{S}|} a_{ik}\beta_j w(\theta^{jk})\\ &= \sum_{k=1}^{|\mathbf{S}|}(a_{ik}\sum_{j=1}^{N}\beta_j w(\theta^{jk}))\\ &= \sum_{k=1}^{|\mathbf{S}|}(a_{ik}\sum_{j=1}^{N}\beta_j(e_j - r_j))\\ &= \sum_{k=1}^{|\mathbf{S}|} a_{ik}B_{S_k}\end{aligned}$$

where S_k is the cache content of group k. Because $a_{ik} > 0$ and there must be some S_k such that $B_{S_k} < B_{max}$, so $\sum_{k=1}^{|\mathbf{S}|} a_{ik}B_{S_k} < B_{max}$, thus the policy π inducing an irreducible Markov chain must have an average reward strictly smaller than B_{max}.

2) Unichain: If a policy induces a unichain P_*^π, i.e., the induced chain must consist of one closed communicating class D and a (possibly empty) set of transient states T. Suppose D contains b groups. Then any row in P_*^π must be identical and take the following form:

$$\begin{aligned}&[(a_1\beta_1, a_1\beta_2, \dots, a_1\beta_N,), (a_2\beta_1, a_2\beta_2, \dots, a_2\beta_N,),\\ &\qquad \dots, (a_b\beta_1, a_b\beta_2, \dots, a_b\beta_N,), 0, 0, \dots, 0]\end{aligned}$$

subject to $\sum_{k=1}^{b} a_k = 1, a_k > 0$, where a_k is the coefficient for the k^{th} group in D. This result is based on the Markovian theory that the transition matrix for an induced unichain P_*^π must satisfy the following properties [14] :

1) P_*^π has equal rows;
2) Its element satisfies:
$$p_*^\pi(\theta'|\theta) = \begin{cases} v & \theta' \in D\\ 0 & \theta' \in T\end{cases} \tag{10}$$
where v is a non-zero value, and;
3) Suppose the row is denoted as a vector V, then it must satisfy $VP^\pi = P^\pi$, and $\sum_{v\in V} v = 1$.

The average reward for any starting state θ can then be calculated following the same rule as in section II-B1:

$$\begin{aligned} aw^\pi(\theta) &= P_*^\pi w^\pi(\theta)\\ &= \sum_{j=1,k=1}^{j=N,k=b} a_{ik}\beta_j w(\theta^{jk})\\ &= \sum_{k=1}^{b}(a_{ik}\sum_{j=1}^{N}\beta_j w(\theta^{jk}))\\ &= \sum_{k=1}^{b}(a_{ik}\sum_{j=1}^{N}\beta_j(e_j - r_j))\\ &= \sum_{k=1}^{b} a_{ik}B_{S_k}\\ &\leq B_{max}\end{aligned}$$

This shows that the resulting average reward is only relevant to the groups in the closed communicating class D. Specifically, if the groups contained in D all share the same optimal cache content S^{OPT} (might be multiple) by the static selection, then the average reward for every starting state θ can achieve optimum B_{max}.

3) Multichain: Suppose policy π partitions the induced chain into disjoint closed communicating classes $D_l, l = 1, 2, \ldots, L, L \leq |\mathbf{\Theta}|$, and possibly a set of transient states T. We can express any transition matrix P of such kind into its canonical form as below:

$$P^{\pi} = \begin{pmatrix} P_1 & 0 & 0 & \cdot & \cdot & 0 \\ 0 & P_2 & 0 & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ 0 & & & & P_L & 0 \\ q_1 & q_2 & \cdot & \cdot & q_L & q_{L+1} \end{pmatrix} \tag{11}$$

where P_l corresponds to transitions between states in D_l under policy π, q_l corresponds to transitions from states in T to D_l, and q_{L+1} to transitions between states within T. We call this form the *canonical form* of the transition matrix. The limiting matrix was proved to be in the following form [14]:

$$P^{\pi}_{*} = \begin{pmatrix} P^*_1 & 0 & 0 & \cdot & \cdot & 0 \\ 0 & P^*_2 & 0 & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & \\ \cdot & & & \cdot & & \\ 0 & & & & P^*_L & 0 \\ q^*_1 & q^*_2 & \cdot & \cdot & q^*_L & q^*_{L+1} \end{pmatrix} \tag{12}$$

where P^*_i and q^*_j are the limiting matrix for P_i and q_j respectively, and the rows in each of them have similar structure as described in the Irreducible case.

Write one step reward vector in its canonical form as in eq. (9), then the average rewards for the starting state θ_i in $D_l, l \in [1, L]$ must be

$$\sum_{G_{S_k} \in D_l} a_{ik} B_{S_k} \leq B_{max}$$

as we have discussed in the Irreducible case. Only when a closed communicating class is composed of states with cache content S^{OPT}, can the average reward be B_{max}. We don't even need to look at the limiting distribution for the transient states before concluding that a multichain cannot guarantee that every starting state obtains the optimal average reward B_{max}.

So far we have discussed all three kinds of chains induced by a stationary policy π. Regardless of the kind of chain induced by a policy, the average reward is no greater than B_{max}, which proves theorem 1. Furthermore, the proof lead to the following theorem:

Theorem 2: The optimal average reward $aw^{OPT}(\theta)$ a stationary policy can achieve is B_{max} for any starting state θ, and it is obtained only by a policy inducing a unichain whose only closed communicating class is made of the optimal groups. A group is optimal if its N states share the cache content S^{OPT}.

The proof follows naturally from the discussion in proving theorem 1.

The theorem also holds for the case where dependence relationship exists among the objects, such as the case in the data cube in OLAP. Our result is in contrast with the result in Dynamat [13] in which the authors argued that the dynamic view management system using Smallest Penalty First (SPF) policy is better than the static selection even when the queries form a uniform distribution. It's worth asking why their experiment results do not agree with our theoretical results. Some reasons might include:

1) Though they also tested the uniform distribution, they omitted the first 10% of queries.
2) They used the DCSR per view metric which is good for any dynamic algorithm but bad for the static algorithm and it doesn't guarantee the overall query answering cost.

C. A branch of Optimal Policies

The stationary optimal policy can be found using the standard value iteration algorithm [4]. However, the value iteration algorithm requires the information about all states before it can answer the request query sequence. This is in contrast to policy A_0, B_0 and C^*_0 in which the optimal decisions can be made solely on the information of the current cache content and the request. This is because A_0, B_0 and C^*_0 are stack algorithms [8] and can progressively converge to the optimum cache content by evicting the object with smallest $\beta_i(e_i - r_i)/|f_i|$ when $|f_i|$ is uniform. These kinds of algorithms share the inclusion property that for any query sequence $\varpi = \{q_1, q_2, \ldots\}$ and any cache capacity k: $S_t(k, \varpi) \subseteq S_t(k+1, \varpi)$, where $S_t(k, \varpi)$ is the cache content for a cache with capacity k and query sequence ϖ at time t. However when the sizes $|f_i|$ are arbitrary, the optimal algorithms for this case do not have the inclusion property. Nevertheless we can manage to remember the seen objects so far and make the cache content converge to the most optimal states for the time being. Such an algorithm called D_0 is shown in fig. 1, in which c_i represents $\beta_i(e_i - r_i)$ for each object in fig. 1.

```
1. Set $B^{(0)}(j) = 0$ for every $j \in [1, K]$
2. If a new object $f_i$ is requested, set $n = n + 1$,
     update for each $j$
     $B^{(n)}(j) = B^{(n-1)}(j)$ if $|f_i| > j$
     $B^{(n)}(j) = max\{B^{(n-1)}(j), c_i + B^{(n-1)}(j - |f_i|)\}$
     if $|f_i| \leq j$
3. If $B^{(n)}(K) \neq B^{(n-1)}(K)$
     Update the cache content to be $B^{(n)}(K)$
4. Loop back to step 2 when seeing the next request.
```

Fig. 1. The D_0 Algorithm

D_0 reaches the optimal cache content when all the N objects are met, and from that point on the system remains with the optimal static cache content. The time taken to reach that point is $\sum_{i=1}^{N} 1/\beta_i$ and is finite, and the average reward of policy D_0 is also B_{max}.

Though this policy needs to remember the objects seen in the past and hence is a history dependent policy, it has a smaller time complexity than the value iteration algorithm [4]. The space consumption is $O(K)$. Each time a new object is seen, the algorithm incurs a cost of $O(K)$,so the total time complexity incurred by the cache content update is $O(NK)$.

This policy, however, has a drawback that it is not a demand policy and may load objects which would be evicted without being referenced at all. Its demand version, which we call D_0^*, remembers the state of D_0 but defers the admissions and evictions of objects until they are requested. As proved in [8], the demand version of a policy is no worse than itself under the total reward criteria.

As a summary of the above analysis, we can see that all optimal policies under IRM would progress to the optimal cache content over time, regardless whether the sizes and retrieval costs are arbitrary. The difference is, when the sizes are uniform, such convergence only needs to examine the current cache content and current request because of the inclusion property. We call this branch of optimal policies the *convergence policy,* and C_0^* and A_0^* are simple special cases of it.

III. Approximation Algorithms

In general, static selection is NP-hard. It was shown in [12] for the case of choosing the best views to materialize in a data warehouse subject to a total storage capacity. This result extends to a general setting of static selection where the sizes and costs of objects are arbitrary. This hardness result motivates approximation algorithms for static selection. And given our result that dynamic replacement policies cannot exceed the performance of static selection (Theorem 1), the hardness result essentially issues a call for approximation algorithms for replacement policies as well.

Using the same set of notations, the static selection problem can be interpreted as an integer programming problem as follows:

$$\begin{aligned}&\text{Minimize } C : \sum_{i=1}^{N}(\beta_i e_i x_i + \beta_i r_i(1-x_i))\\ &\text{Subject to } \sum_{i=1}^{N}|f_i|x_i \geq K',\\ &x_i = 0, 1\end{aligned}$$

where C is the expected cost to answer a request, $K' = \sum_{i=1}^{N}|f_i| - K$ and K is the cache capacity. $x_i = 0$ means f_i is cached, $x_i = 1$ means f_i is evicted.

If we use c_i to represent $\beta_i(e_i - r_i)$ for each object, the objective value in the above formulation can be simplified as:

$$\sum_{i=1}^{N}\beta_i e_i x_i + \beta_i r_i(1-x_i)) = \sum_{i=1}^{N}c_i x_i + \sum_{i=1}^{N}\beta_i r_i = \sum_{i=1}^{N}c_i x_i + \delta \tag{13}$$

where $\delta = \sum_{i=1}^{N}\beta_i r_i$ is a constant. So the objective value we want to minimize is just $\sum_{i=1}^{N}c_i x_i$. If we find a solution with approximation ratio k for it, then the Static Selection problem will also obtain a solution with approximation ratio k.

A. A Simple Greedy Algorithm

A standard approach for any integer program is to relax the integrality constraint to a linear one. In the linear relaxation of the problem, the requirement on x_i is relaxed to $0 \leq x_i \leq 1$. The solution of the LP problem can be computed in a very simple way: Sort the objects in terms of $\frac{c_i}{|f_i|}$ in non-decreasing order. Without loss of generality, assume

$$\frac{c_1}{|f_1|} \leq \frac{c_2}{|f_2|} \leq \dots \frac{c_N}{|f_N|}$$

An optimal solution is a vector $x^{LP} = (x_1^{LP}, x_2^{LP}, \dots, x_N^{LP})$ is defined as:

$$\begin{aligned}x_j^{LP} &= 1, j = 1, \dots, \varsigma - 1\\ x_\varsigma^{LP} &= \frac{K' - \sum_{j=1}^{\varsigma-1}|f_j|}{|f_\varsigma|}\\ x_j^{LP} &= 0, j = \varsigma + 1, \dots, N\end{aligned}$$

In other words, the solution evicts the whole objects from x_1 to $x_{\varsigma-1}$ and partial of x_ς while cache the remaining parts. Here we call f_ς as the *split object*. The corresponding solution value is then

$$C = \sum_{i=1}^{\varsigma-1}c_i + \frac{(K' - \sum_{j=1}^{\varsigma-1}|f_j|)}{|f_\varsigma|}c_\varsigma$$

Inspired by the optimal solution for the LP relaxation of the object selection/eviction problem, we now present an algorithm for the IP version of the problem in fig. 2.

```
1. Sort objects in non-decreasing order of $\frac{c_i}{|f_i|}$.
     Assume the resulting order is $j, j = 1, 2, \dots, N$
2. First assume all objects are cached, then evict the
     objects ranked from 1 to $\varsigma$, where $f_\varsigma$ is the split object.
```

Fig. 2. The Simple Greedy Algorithm

Theorem 3: The greedy algorithm in fig. 2, denoted as G, is K-approximate, i.e., $C_G \leq KC_{LP}^*$, where C_G is the expected cost to answer a query by algorithm G and C_{LP}^* is the fractional optimal expected cost.

Proof: Denote the first part of eq. (13) as $\Gamma[C]$, i.e.

$$\Gamma[C] = \sum_{i=1}^{N}c_i x_i$$

, its corresponding result by algorithm G as $\Gamma_G[C]$, and the optimal LP result $\Gamma^*_{LP}[C]$:

$$\begin{aligned}\Gamma_G[C] &= \sum_{i=1}^{\varsigma-1} c_i + c_\varsigma \\ &\leq \sum_{i=1}^{\varsigma-1} c_i + K\frac{(K' - \sum_{j=1}^{\varsigma-1}|f_j|)}{|f_\varsigma|}c_\varsigma \\ &\leq K\cdot(\sum_{i=1}^{\varsigma-1} c_i + \frac{(K' - \sum_{j=1}^{\varsigma-1}|f_j|)}{|f_\varsigma|}c_\varsigma) \\ &\leq K\cdot\Gamma^*_{LP}[C]\end{aligned}$$

Thus

$$\begin{aligned}C_G &= \delta + \Gamma_G[C] \\ &\leq \delta + K\cdot\Gamma^*_{LP}[C] \\ &\leq K\cdot(\delta + \Gamma^*_{LP}[C]) \\ &\leq K\cdot C^*_{LP}\end{aligned} \tag{14}$$

■

Since proving the actual cost C is k approximate is equal to proving $\Gamma[C]$ is k approximate, we will only care for the $\Gamma[C]$ part while ignoring the constant δ in the future.

B. The Lower Bound of The Approximation Ratios

Given a LP relaxation of any minimization problem, let $\chi^*_{LP}(I)$ be the objective function value of an optimal solution to the LP-relaxation, and let $\chi^*(I)$ be the objective function value of an optimal solution to the original IP problem, the integrality gap of the relaxation is defined as

$$\sup_I \frac{\chi^*(I)}{\chi^*_{LP}(I)}$$

If the objective function value of the solution found by the algorithm is compared directly with that of an optimal fractional solution, the best approximation factor we can hope to prove is the integrality gap of the relaxation [19]. In our case the task is to examine the integrality gap on the expected cost to see if it is smaller than K. The result is given in the following theorem:

Theorem 4: The integrality gap on the expected cost for the object selection/eviction problem, defined as $\sup_I \frac{C^*(I)}{C^*_{LP}(I)}$, is at least K, where I refers to an instance, $C^*(I)$ refers to the integral optimal cost on I, and $C^*_{LP}(I)$ means the fractional optimal cost on I.

Proof: Consider the following instance in table I:

object	size $\lvert f_i\rvert$	cost c_i	$c_i/\lvert f_i\rvert$
f_1	K	K	1
f_2	1	$K+1$	$K+1$

TABLE I
AN INSTANCE WHERE THE INTEGRALITY GAP IS K

In this instance, $C^*(I) = K$ while $C^*_{LP}(I) = 1$, so the integrality gap (of all instances) is at least K. ■

This shows that the best approximation ratio for the object selection/eviction problem is K, meaning no polynomial algorithm can achieve a better ratio. The direct consequence of this theorem is the following corollary.

Corollary 1: The Greedy algorithm in fig. 2 can achieve the best approximation ratio for the object selection/eviction problem.

C. The KnapSack Revisited

Now let's revisit the well known 2-approximation KnapSack solution. It also sorts the objects in terms of $c_i/|f_i|$, but unlike the Greedy Algorithm in fig. 2, it solves the problem by selecting the objects to cache, not to evict. Recall, we use c_i to denote $\beta_i(e_i - r_i)$.

Like Greedy, KnapSack's 2-approximation algorithm does not cache any of the objects in $\{f_1, f_2, ..., f_{\varsigma-1}\}$. Instead, it picks the better of $\{f_{\varsigma+1}, f_{\varsigma+2}, ..., f_N\}$ and $\{f_\varsigma\}$ to cache. Since we know the Greedy algorithm always evicts all the objects in $\{f_1, f_2, ..., f_\varsigma\}$, KnapSack's 2-approximation algorithm must be better than the Greedy. This means that KnapSack's 2-approximation algorithm not only has the best approximation ratio K on expected costs, but also achieves at least half of the optimal benefit and must outperform Greedy. Note that in general an algorithm that achieves an approximation ratio on the expected benefit does not guarantee any approximation ratio on the expected cost. For example, in the data cube case [12], where the views are correlated to each other, there is no polynomial algorithm that can achieve an approximation ratio on the expected cost, but there exists such algorithms that can achieve an approximation ratio on the expected benefit. The distinction is caused by the extra hardness introduced by the relationships among the views.

D. An Extended Greedy Algorithm

In section III-B we have shown that the Simple Greedy algorithm in fig. 2 achieves the best approximation ratio K. But that doesn't mean it performs well under all circumstances. In this section we present another approximation algorithm, called *Extended Greedy*, that might outperform the Simple Greedy algorithm in some cases.

We have known that the problem can be seen as evicting a subset of the objects in F. Let's call this subset $\overline{S}$. The Extended Greedy algorithm iteratively picks the least cost-expensive object into $\overline{S}$ until the size $|\overline{S}|$ is no less than K', where $K' = \sum_{f_i\in F}|f_i| - K$. Suppose the object picked in iteration j is denoted as f_j and the set selected by the end of iteration j is $\overline{S}_j$, the *cost-expensiveness* for f_j is defined as follows:

(Cost-Expensiveness) The cost-expensiveness for f_j in iteration j is

$$\frac{c_i}{\min\{|f_j|, K' - |\overline{S}_{j-1}|\}}$$

, where $\overline{S}_{j-1}$ is the current subset to evict when iteration $j-1$ ends. $\overline{S}_0 = \phi$.

The Extended Greedy algorithm is defined formally in fig. 3.

```
1. $j = 0; \overline{S}_j = \phi$
2. While $|\overline{S}_j| \le K'$
    j = j+1;
    Find the object $f_j$ with the smallest cost-expensiveness,
    where $j$ is the index of current iteration.
    (Break ties arbitrarily)
    $\overline{S}_j = \overline{S}_{j-1} \cup f_j$
3. Cache the objects in $F - \overline{S}_j$.
```

Fig. 3. The Extended Greedy Algorithm

In any iteration j, the selected object f_j covers the space of size $\min\{|f_j|, K' - |\overline{S}_{j-1}|\}$ with cost c_j, where $|\overline{S}_{j-1}|$ is the total size of the objects in $\overline{S}_{j-1}$. Let the cost per unit space, or cost density incurred by these objects be: $\rho_1, \rho_2, \ldots, \rho_{K'}$ such that $C = \Sigma_{j=1}^{K'} \rho_j$. The following lemma holds:

Lemma 1:

$$\rho_j \le \frac{C^*}{K' - j + 1}, j = 1, 2, \ldots, K'$$

where C^* is the integral optimal expected cost for the IP static selection problem.

Proof: Suppose in iteration j, the selected object f_j covers the space indexed from $|\overline{S}_{j-1}| + 1$ to $|\overline{S}_j|$, ($|\overline{S}_j| = |\overline{S}_{j-1}| + |f_j|$). The cost densities incurred by it are denoted as $\rho_{|\overline{S}_{j-1}|+1}, \rho_{|\overline{S}_{j-1}|+1}, \ldots, \rho_{\overline{S}_j}$. As they belong to one object, these values are the identical. Without losing generality, suppose the order in which the objects are chosen to evict is the same as their original numbering, i.e., the first object chosen is f_1, the second chosen is f_2, etc.

Let the subset to evict selected by the optimal IP solution be $\overline{S}_*$. In any iteration j when the Extended Greedy algorithm in fig. 3 is picking a object to evict, there must be some objects belonging to $\overline{S}_*$ that have not been picked yet. Denote these objects set O_j, and $O_j = \overline{S}_* - \overline{S}_{j-1}$. objects in O_j can fill the left uncovered space at a cost $\sum_{f_i \in O_j} c_i$, which is no greater than the optimum solution's total cost $\overline{S}_*$.

$$\begin{aligned}
\rho_{|\overline{S}_{j-1}|+1} &= \rho_{|\overline{S}_{j-1}|+2} = \ldots = \rho_{\overline{S}_j} \\
&= \frac{c_j}{\min\{|f_j|, K' - |\overline{S}_{j-1}|\}} \\
&\le \text{any} \frac{c_i}{\min\{|f_i|, K' - |\overline{S}_{j-1}|\}}, f_i \in O_j \\
&\le \frac{\sum_{f_i \in O_j} c_i}{\sum_{f_i \in O_j} |f_i|} \\
&\le \frac{C^*}{\sum_{f_i \in O_j} |f_i|} \\
&\le \frac{C^*}{K' - |\overline{S}_{j-1}|} \\
&\le \frac{C^*}{K' - |\overline{S}_{j-1}| - |f_j| + 1}
\end{aligned}$$

In a more straightforward way, we can write this result for iteration j into:

$$\begin{aligned}
\rho_{|\overline{S}_{j-1}|+1} &\le \frac{C^*}{K' - |\overline{S}_{j-1}|} \\
\rho_{|\overline{S}_{j-1}|+2} &\le \frac{C^*}{K' - |\overline{S}_{j-1}| - 1} \\
\cdots & \quad \cdots \\
\rho_{|\overline{S}_{j-1}|+|f_j|} &\le \frac{C^*}{K' - |\overline{S}_{j-1}| - |f_j| + 1}
\end{aligned}$$

Now let's start from the first view:

$$\begin{aligned}
\rho_1 &\le \frac{C^*}{K'} \\
\rho_2 &\le \frac{C^*}{K' - 1} \\
\cdots & \quad \cdots \\
\rho_{K'} &\le C^*
\end{aligned}$$

The lemma is proved.

■

The lemma leads naturally to the following theorem:

Theorem 5: The Extended Greedy algorithm is $H_{K'}$-approximate with respect to the optimal IP solution, where $H_{K'}$ is the K'th harmonic number.

Proof: Denote the first part of eq. (13) by the Extended Greedy algorithm as C_{EG}. Adding up ρ_j from $j = 1$ to K', we get the following:

$$\begin{aligned}
C_{EG} &= \sum_{j=1}^{K'} \rho_j \\
&\le \sum_{j=1}^{K'} \frac{C^*}{K' - j + 1} \\
&\le H_{K'} C^*
\end{aligned}$$

■

Note that the approximation ratio of the Extended Greedy $H_{K'}$ is with respect to the solution of the IP problem. However this doesn't mean that Extended Greedy is worse than Greedy. It can outperform both the Greedy and the Kanpsack's 2-approximation algorithms in some cases. Here is an example of such an instance:

In this example the system has four objects and their features are shown in table II.

object	cost c_i	size $\|f_i\|$	$c_i/\|f_i\|$
f_1	20	40	0.5
f_2	40	40	1
f_3	20	10	2
f_4	15	5	3

TABLE II
AN INSTANCE WHERE EXTENDED GREEDY IS THE BEST

Suppose the cache capacity is 45, the execution of the three algorithms are respectively shown in table III

Algorithm	Evicted object	Resulting Cost
KS 2-app	f_1, f_3, f_4	55
Greedy	f_1, f_2	60
Extended Greedy	f_1, f_3	40

TABLE III
THE RESULT OF THE THREE ALGORITHMS

E. A Mediated Greedy Algorithm

Just like what we did in the Greedy algorithm and the 2-approximation knapsack algorithm, let's sort the objects in terms of the cost density $\frac{c_i}{|f_i|}$ in non-increasing order. Without losing generality, suppose the result objects' indexes are conforming to their original indexes, i.e.,

$$\frac{c_1}{|f_1|} \leq \frac{c_2}{|f_2|} \leq \ldots \leq \frac{c_N}{|f_N|}$$

The split object is denoted as f_ς. By analyzing the above three algorithms, we found that they all must evict the first $\varsigma - 1$ objects. The difference is how they deal with the objects from f_ς to f_N. Thus we can mediate the three algorithms so that the new algorithm not only achieves the best approximation ratio K on the expected cost and approximation ratio 2 on the benefit, but also performs no worse than the Extended Greedy. We call the new algorithm *Mediated Greedy* algorithm. Like in other algorithms, the objects are first sorted by $c_i/|f_i|$ in non-decreasing order. We demarcate the objects into three parts: 1) objects below the split object, denoted as F_{low}; 2) The split object; 3) objects above the split object, denoted as F_{high}. The algorithm now calculates which objects need to be evicted in addition to the first $\varsigma - 1$ objects and we denote it as $\overline{S}$. The remaining part of the algorithm is shown in fig. 4, in which we use notation $C(S)$ to represent $\sum_{f_i \in S} c_i$.

```
1. Get $C(F_{high})$,
2. $\overline{S} = \phi$.
3. While ( $|\overline{S}| \leq K' - |F_{low}|$ and
      $C(\overline{S}) < \min\{C(F_{high}), C(\{f_\varsigma\})\}$ )
      In $F_{high} \cup \{f_\varsigma\}$,find the object $f_j$ with the smallest
      cost-expensiveness. (Break ties arbitrarily)
      $\overline{S} = \overline{S} \cup \{f_j\}$
4. If $C(\overline{S}) \geq \min\{C(F_{high}), C(\{f_\varsigma\})\}$
        Cache the better of $F_{high}$ or $f_\varsigma$, evict all others.
   Else
        Cache the objects in $F - \overline{S}$ - $F_{low}$, evict all others.
```

Fig. 4. The Mediated Greedy Algorithm

The algorithm is based on the Extended Greedy but it first identifies those objects that must be evicted (f_1 to $f_{\varsigma-1}$), then it iteratively picks objects in $F_{high} \cup f_\varsigma$ to evict. It keeps a bound of $\min\{C(F_{high}), C(\{f_\varsigma\})\}$ when choosing the objects to evict, so that if the cost incurred by the chosen objects exceeds the bound, it stops and choose the better of F_{high} or f_ς to evict.

F. The K-Competitive Policy

Following previous sections, we introduce *competitive MDP policies* defined below:

In an MDP system with the objective value function $ov^\pi(\theta)$, where π is a policy and θ is the starting state, if $ov^\pi(\theta) \leq k \cdot ov^{OPT}(\theta) + \xi$ holds for any starting state s, where ξ is a constant and $ov^{OPT}(\theta)$ is the optimal objective value, we say policy π is *k-competitive*.

Suppose at time t, the system is in state $\theta_t = (S_{t-1}, q_t)$ where S_{t-1} is the cache content at the beginning of time t and q_t is the new request. Like in section II, we only consider the situation of faults. The policy is given in fig. 5. Note that the D1 policy uses non-mandatory replacement.

```
Whenever a fault occurs on request $q_t$ at time t and
there's not enough room for $q_t$ in the cache:
   While the request $q_t$ doesn't fit into the cache,
      Evict the object with the smallest $\frac{c_i}{|f_o|}$ from $S_{t-1} \cup \{q_t\}$
```

Fig. 5. The D_1 Policy

Theorem 6: The policy in fig. 5, denoted as D_1 , is K-competitive to the optimal fractional solution of the static selection problem, i.e.,

$$ac^{D_1}(\theta) \leq K \cdot C^*_{LP} \text{ for all starting state } \theta$$

where K is the capacity of the cache, and C^*_{LP} is the expected cost for the optimal fractional solution to the static object selection problem.

Proof: Under policy D_1, the states converge to the group G_S whose N states share the same cache content $S = (\{f_{\varsigma+1}, f_{\varsigma+2}, \ldots, f_N\}$ for any starting state within finite time. So this policy is a stationary policy that induces a unichain Markov chain, with the only one closed communicating class being equal to group G_S. From the result in section II, we know the average cost $ac^\pi(\theta)$ for policy π is $P^\pi_* c^\pi(\theta) = B_S$. Since $B_S = \sum_{i=\varsigma+1}^N \beta_i(e_i - r_i) \leq K \cdot C^*_{LP}]$, the theorem is proved. ■

Scheuermann *et al.* introduced a similar policy in [16], and showed that the policy is nearly optimal when the number of objects is very large. However it does not come with a more strict theoretical analysis.

IV. Experiments

The replacement policies we tested include: optimal dynamic policy D_0, D^*_0, the competitive dynamic policy D^*_1, the classical Greedy-Duel Size algorithm [5] and its variant Greedy-Duel Size Frequency algorithm [6]. We also implemented the static algorithms integral OPT, denoted as S_0, the static fractional OPT, denoted as S^*_0, Simple Greedy and Mediated Greedy. The experiments were done under both the IRM and real web trace.

A. *The Independent Reference Model*

To simulate the Independent Reference Model, we built a random query generator which generates query sequences with the Zipf distribution. The query sequence is made of sequential requests to a set of simulated objects with their sizes uniformly distributed in $[1, 100]$. The retrieval costs from remote server are also uniformly distributed in [0,10], not related to the sizes. The retrieval costs from cache are set to be a small fraction of the retrieval costs from remote server plus some constant term to account for the I/O cost. Various query sequences were generated with different number of objects and query length. The cache capacity were chosen from 5% of total object sizes to 50% of total object sizes. Under each setting the test was repeated for 500 trials to get the expected query answering costs.

Fig. 6 shows a typical result of expected query answering costs (average cost) of different policies/algorithms. In this particular test, the query sequence is composed of 5000 queries on 100 objects. The results under different settings show similar patterns as described blow.

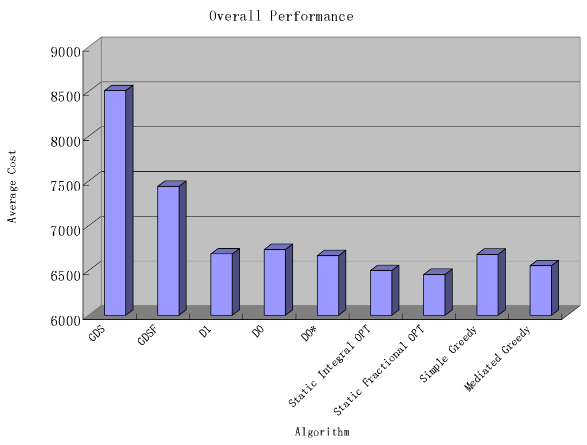

Fig. 6. The average cost to answer queries

Among the dynamic policies and static algorithms tested, the static integral OPT S_0 is basically the best except for the static fractional OPT S_0^*. The GDS policy performs poorly under the IRM. Though its variant GDSF improves a lot, it is still much worse compared to the other three dynamic policies like D_0, D_0^* and D_1. D_0, D_0^* and D_1 are very close to each other but they are all worse than S_0, which confirms theorem 1. Meanwhile, as expected, the demand version of dynamic OPT policy D_0^* is always better than D_0. The competitive greedy policy D_1 is slightly worse than D_0^* in this case, but as cache capacity and number of objects varies, it is very close to D_0^*.

For the static algorithms, Simple Greedy and Mediated Greedy are very close, with Mediated Greedy slightly better.

To study the dynamic policies' behavior, we also recorded their performance over time and compare it with S_0. In fig. 7, we use query sequences of length 10000 and number of objects 1000, and show the expected costs to answer 500 such sequences when time proceeds. The X coordinate, the time, which is made of the 10000 queries, is divided into 100 slots. The result shows that the query answering costs for S_0 remains steady, and D_0 and D_0* converge to S_0 within 1500 queries. In that period, D_0 performs a little worse than D_0*, because it may switch in some objects that are not referenced later. D_1 is almost as good as D_0* in this example.

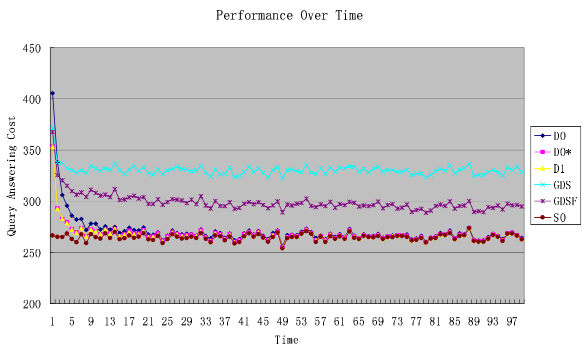

Fig. 7. The average cost to answer queries

To compare the cache content of the dynamic policies over time, we used the *correlation* factor defined as $|S_t^\pi \cap S_{S_0}|/|S_{S_0}|$, where S_t^π is the cache content for policy π at time t, and S_{S_0} is the cache content for algorithm S_0. Fig. 8 shows the correlation between different policies:

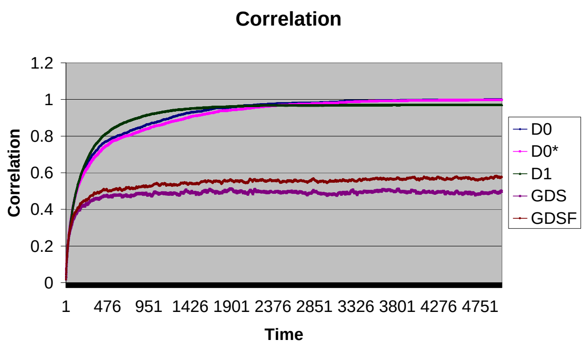

Fig. 8. The average cost to answer queries

D_0 and D_0* both converge to 1, while D_0 converges faster than D_0*. D_1 reaches a correlation value very close to 1 but could never reach 1 in this example. However when cache capacity or number of objects increases, D_1 could reach 1 as well. GDS and GDSF have a much lower correlation, showing that their cache contents are quite different from that of S_0's.

In summary, the experiment results for IRM basically satisfy our expectations, exemplifying the correctness of our theory.

B. *Experiments on the weblogs*

To test our policies under a real workload, we used our departmental weblogs as the underlying data. We chose the web request logs on two random consecutive days, with the query sequence on the first day as the probation period to gather the information required to run the algorithms, and the query sequence on the next day as real workload. According to our analysis, the queries distribution is highly skewed. The number of queries on the first day is 480480 and they are cast on 66197 distinct web objects. The number of queries on the second day is 345659 on 112380 distinct web objects. There are 46183 new distinct objects requested on the second day, but most of them were referenced only once.

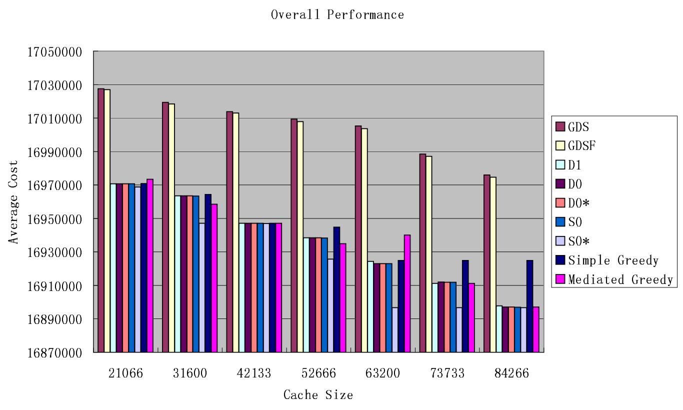

Fig. 9. The average cost to answer queries

The overall performance is shown in fig. 9. S_0 is still the best except for S_0^* in most of the cases. GDS and GDSF are still worse than other policies in this test, probably because of the extremely skew distribution of the queries. The remaining three dynamic policies perform very close to S_0. The static greedy algorithms show more variations in terms of overall performance. Overall, these policies have very similar performance, but when talking about the algorithm running times, the optimal static algorithm and policies are far too slower than the competitive ones. The stable performance of D_1 and its excellent running time suggests it to be a good choice.

Unlike the IRM, the query answering cost pattern does not follow a specific curve. The following figure shows S_0's query answering cost over time, but all the other policies/algorithms show the similar pattern.

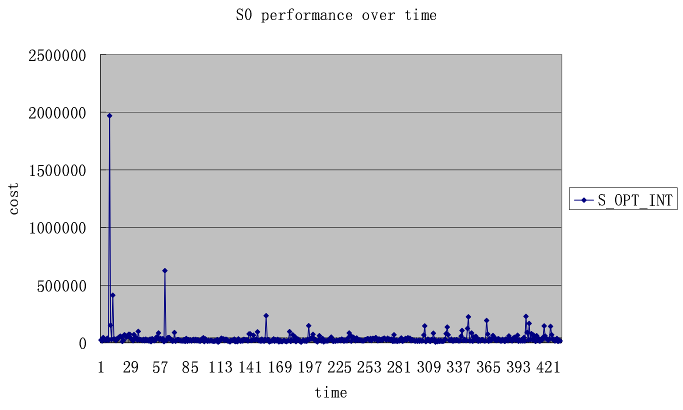

Fig. 10. The average cost to answer queries

However, the cache content of D_0 and D_0^* do converge to that of S_0, though it shows some jitter in the initial phase.

V. Conclusions

Cache management is a critical component of several systems including file managers, web proxies, and data warehouses to name a few. Static selection focuses on choosing the best possible objects to cache (subject to a constraint on capacity) and leaving them in the cache forever. Dynamic cache management relies on replacement policies which decide whether, when, and which objects to admit to or evict from the cache.

In this paper, we considered a general setting whereby objects requested can have arbitrary sizes and arbitrary retrieval (or evaluation) cost. Under the IRM, we showed the surprising result that dynamic policies cannot exceed the performance of static selection in terms of expected average reward over any request sequence obeying the given probability distribution over requests. Our result shows that a dynamic policy approaches the performance of (optimal) static selection iff the stochastic chain induced by it is irreducible. Given that optimal static selection is NP-hard in general. we develop polynomial time approximation algorithms for both static selection and for dynamic replacement. The algorithms are K-approximate w.r.t. the fractional optimum solution for static selection and H'_K-approximate for w.r.t. the integral optimum solution for static selection. We also show that these results are tight by showing that K is the best possible approximation ratio for both static selection and for dynamic replacement. Finding optimum (static or dynamic) solutions without the IRM assumption is an open problem.

References

[1] O. Bahat and M. Makowski. Optimal replacement policies for non-uniform cache objects with optional eviction. In *IEEE INFOCOM Conference*, volume 1, pages 427–437, 2003.

[2] J. Bather. Optimal decision procedures for finite markov chains. *Advances in Applied Probability*, 5(2):328–339, 1973.

[3] L. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.

[4] R. Bellman. A markovian decision process. In *Journal of Mathematics and Mechanics*, volume 6, 1957.

[5] P. Cao and S. Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.

[6] L. Cherkasova. Improving www proxies performance with greedy-dual-size-frequency caching policy. Technical Report HPL-98-69R, HP Laboratories, 1998.

[7] K. L. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer, 1960.

[8] E. Coffman and P. Denning. *Operating Systems Theory*. Prentice Hall, 1973.

[9] A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics. In *8th ACM-SIAM Symp. on Discrete Algorithms*, 1997.

[10] H. Gupta. Selection of views to materialize in a data warehouse. In *IEEE Transactions on Knowledge and Data Engineering*, volume 17, pages 24–43, 2005.

[11] H. Gupta and D. Srivastava. The data warehouse of newsgroups. In *The 7th International Conference on Database Theory*, pages 471–488, 1999.

[12] V. Harinarayan, A. Rajaraman, and D. Ullman. Implementing data cubes efficiently. In *ACM SIGMOD*, volume 25, pages 205–216, 1996.

[13] Y. Kotidis and N. Roussopoulos. Dynamat: A dynamic view management system for data warehouses. In *In Proc. of the ACM SIGMOD Conference*, pages 371–382, 1999.

[14] M. L. Puterman. *Markov Decision Process: Discrete Stochastic Dynamic Programming*. John Wiley & SONs,INC, 1994.

[15] S. M. Ross. *Introduction to Stochastic Dynamic Programming: Probability and Mathematical*. Academic Press, Inc., 1983.

[16] P. Scheuermann, J. Shim, and V. Radek. Watchman: A data warehouse intelligent cache manager. In *Proceedings of the 22nd VLDB Conference*, 1996.

[17] Y. Su. On managing visibility of resources in social networking sites. Master's thesis, University of British Columbia, November 2008.

[18] Y. Su and L. V. Lakshmanan. On efficient replacement policies for cache objects with non-uniform sizes and costs http://www.cs.ubc.ca/~laks/cache.pdf. Technical report, University of British Columbia, July 2009.

[19] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.