On Managing Visibility of Resources in Social Networking Sites

by

Ying Su

B. Engineering, Tsinghua University, Beijing, China, 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES
(Computer Science)

The University Of British Columbia
(Vancouver)

October 2008
© Ying Su
Abstract

The social network sites have been very popular these days. In these systems, resources are associated with each user (i.e., network node) in the network, and these resources include both soft and physical resources, such as pictures, videos, books, etc. Users would like to control who can see their resources, e.g., any one, only friends, friends and families, friends of friends, etc. The rules defining who can see the resources are called visibility views, and in our system they are identified by the owner nodes and regular path expressions. The users can issue queries for discovering resources, e.g., “Find all lawnmowers in my three degree network that are visible to me”. The evaluation process needs to check all candidate resources to see if they are visible to the query issuer, and this requires computing the visibility views associated with every candidate resource. The visibility views are expensive to evaluate. In order to facilitate the visibility query answering, it is necessary to pre-compute and materialize the views in the so-called “cache”. But because of the tremendous volume of the whole view data, we have to select only a portion of them to materialize, based on metrics such as the views’ sizes, the costs to evaluate them, the popularity of references, the temporal locality of requested views, and the correlation relationships among them. The problem of trying to select the views in a static way is called the static view selection problem, and it has been studied extensively by the database community, mostly for answering OLAP queries in data warehouses. We call it “static” because it pre-computes and materializes the views in one lump-sum, based on statistics accumulated before the query sequence starts.

This problem has a deep root from the caching problem studied in the theory and system communities. The researches done on caching policies solve the problem in an ad-hoc dynamic way by making decisions at each request, which
make use of the temporal locality. Many ad-hoc policies have been proposed in the literature. Specifically, there is a branch of research done based on the so-called Independent Reference Model, in which view requests are assumed to form a fixed time-independent sequence. It was shown that under this model optimal policies can be found. Coffman and Denning [13] presented a policy called $A_0$ that is optimal with respect the long-run average metric for paging, where the cached objects (i.e., memory pages) have uniform sizes and retrieval costs. Bahat [7] later presented a stationary Markov replacement policy $C_0^*$, which is optimal for cases in which the retrieval costs for the cached objects are arbitrary but their sizes are uniform.

In this thesis we address the problem one step further: to assume both the sizes and costs are arbitrary. We reveal the intrinsic relationship between the static view selection problem and the caching problem, and show that, still under the independent reference model, any dynamic replacement policy (random or deterministic) are no better than the optimal static selection algorithm in terms of the long-run average metric. We then present a branch of optimal history dependent replacement policies for the arbitrary sizes and arbitrary costs case, and show that policy $A_0$ and $C_0$ are special cases of our policy. Furthermore, we prove that a policy is optimal if and only if the induced stochastic process is a unichain. We also study the approximate static algorithms and policies: we bring up polynomial static algorithms that are both $K$-approximate with regard to the fractional optimum solution and $H_{K'}$-approximate to the integral optimum solution, where $K$ is the size of the cache, $K'$ is the total size of all view objects minus $K$, and $H_{K'}$ is the harmonic number of $K'$. In addition, we present a $K$-competitive dynamic policy and show that $K$ is the best approximation ratio for both the static algorithms and dynamic policies.
# Table of Contents

Abstract ................................................. ii

Table of Contents ........................................ iv

List of Tables ............................................. vi

List of Figures ............................................. vii

Glossary ................................................ viii

Acknowledgments .......................................... xii

1 Introduction .............................................. 1
   1.1 The Social Networking System .................... 1
      1.1.1 Logical Model .................................... 2
      1.1.2 Visibility Views ................................... 4
   1.2 Problem Definition and Related Work ............. 5

2 Optimal Algorithms Under IRM ......................... 13
   2.1 The Static Selection Problem ..................... 14
   2.2 Finding The Optimal Policies For The Caching Problem .............. 17
      2.2.1 The MDP Framework ............................. 17
      2.2.2 The Objective Value of a Policy ................. 22
      2.2.3 Comparing the Objective Values ................. 25
      2.2.4 The Way to Calculate The Average Reward ...... 28
      2.2.5 Finding The Optimal Policies ................... 37
3 Approximation Algorithms ...................................... 42
  3.1 Approximate Static Selection Algorithms .............. 43
    3.1.1 A Greedy Algorithm With Bound $K$ .............. 43
    3.1.2 The Lower Bound of The Approximate Ratios ...... 48
    3.1.3 The Knapsack Revisited ........................... 49
    3.1.4 An Extended Greedy Algorithm .................... 50
    3.1.5 A Mediated Greedy Algorithm ...................... 53
  3.2 Competitive Policies .................................... 57
    3.2.1 The $K$-Competitive Policy ....................... 58
    3.2.2 Comment To Related Works ........................ 61

4 Conclusion and Future Work ................................. 62
  4.1 The Problem Remaining .................................. 62
    4.1.1 Overtaking Optimal Policies ....................... 62
    4.1.2 The Problem For the Naive Version Problem, Online Case 63
    4.1.3 The Problem For The Dependence Case .......... 63

Bibliography .................................................. 65
List of Tables

1.1 An instance where Algorithm A is not optimal .................. 11

3.1 An instance where the integrality gap is $K$ ....................... 48
3.2 An instance where Extended Greedy is the best ................... 54
3.3 The result of the three algorithms ............................... 54
List of Figures

2.1 The Value Iteration Algorithm ......................... 37
2.2 The $D_0$ Algorithm ................................ 40
2.3 The $D_1$ Algorithm ................................ 40

3.1 Greedy Algorithm ................................ 47
3.2 Knapsack’s Polynomial 2-Approximation Algorithm .......... 50
3.3 The Extended Greedy Algorithm ........................ 51
3.4 The Mediated Greedy Algorithm ...................... 57
3.5 The $K$-competitive Policy .......................... 58
Glossary

\( G \) The social network graph model. \( G = (\mathcal{V}, \mathcal{E}) \)

\( \mathcal{V} \) The set of all nodes in the social network graph.

\( \mathcal{E} \) The set of all edges in the social network graph.

\( \mathcal{B} \) The visibility view definition

\( \Sigma \) The set of all tags in the system

\( V \) The set of all views

\( N \) Total number of distinct views in \( V \)

\( e_i \) The evaluation cost of view \( v_i \)

\( r_i \) The retrieval cost of view \( v_i \)

\( b_i \) The benefit by caching view \( v_i \). \( b_i = e_i - r_i \)

\( \beta_i \) The probability that view \( v_i \) will occur under the Independent Reference Model

\( c_i (e_i - r_i)p_i \) for view \( v_i \)

\( |v_i| \) Size of view \( v_i \)

\( K \) Capacity of the cache
\[ K' = \sum_{i=1}^{N} |\nu_i| - K \]

\( S \) The subset of \( V \) that is selected for materialization by the static selection.

\( C \) Random variable representing the cost of answering a view request

\( E(C) \) The expected cost of answering a view request

\( E^*(C) \) The expected cost of answering a view request for the optimal integral solution

\( E^*_Lp(C) \) The expected cost of answering a view request for the fractional integral solution

\( \alpha \) A constant equal to \( \sum_{i=1}^{N} p_i e_i \)

\( \delta \) A constant equal to \( \sum_{i=1}^{N} p_i r_i \)

\( B_S \) The expected benefit for cache content \( S \) of the static selection solution.

\( B_{max} \) The maximum expected benefit of the static selection solutions.

\( C_{min} \) The minimum expected cost of the static selection solutions. \( C_{min} = \alpha - B_{max} \)

\( t \) The decision epoch time \( t \)

\( T \) The set of decision epochs

\( T \) A constant used for time

\( Q_t \) The query at time \( t \)

\( S_t \) The cache content at time \( t \)

\( G_s \) The set of all states \( \Theta \) with cache content \( s \)

\( S \) The cache content space

\( \Theta_t \) The random variable of MDP state at time \( t \)
\( \Theta \): The state space

\( \theta \): Variable representing an MDP state

\( A_\theta \): The action space for state \( \theta \)

\( A \): The action space for the MDP system

\( p_t(\theta_j | \theta_i, a) \): The transition probability from state \( \theta_i \) to state \( \theta_j \) for action \( a \) at time \( t \)

\( w_t(\theta, a) \): One step reward when taking action \( a \) at state \( \theta \)

\( c_t(\theta, a) \): One step cost when taking action \( a \) at state \( \theta \)

\( d_t(\theta) \): The decision at state \( \theta \) at time \( t \)

\( h_t \): The history up to time \( t \) for some policy

\( H_t \): The space of the history up to time \( t \) for some policy

\( tw^\pi(s) \): The expected total reward of policy \( \pi \in \Pi^HR \)

\( tc^\pi(s) \): The expected total cost of policy \( \pi \in \Pi^HR \)

\( aw^\pi(s) \): The average reward of policy \( \pi \in \Pi^HR \)

\( ac^\pi(s) \): The average cost of policy \( \pi \in \Pi^HR \)

\( \omega \): Sample path, a sequence of states and actions for a policy

\( \Omega \): The space of all sample paths for a policy

\( \sigma \): A query sequence = \( \{Q_1, Q_2, \ldots \} \)

\( P_\pi \): The one-step transition matrix of the induced Markov chain by some stationary policy \( \pi \)

\( p_\pi^{(1)}(\theta_j | \theta_i) \): \( P_\pi \)'s element at row \( i \) and column \( j \) representing the probability of one step state transfer from \( \theta_i \) to \( \theta_j \) by the decision of policy \( \pi \) on state \( \theta_i \)
$P_\pi^*$ Cesaro limit of $P_\pi$

$D_l$ The $l^{th}$ closed communicating class

$L$ A constant representing the number of closed communicating classes

$P_l$ The transitions between states in closed communicating class $D_l$ under a policy

$P_l^*$ The limiting transitions between states in closed communicating class $D_l$ under a policy

$\nu_\xi$ The split view

$\rho$ The cost density: $(e_i - r_l)\beta_l/|\nu_l|$
Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor, Professor Laks V.S. Lakshmanan, for more than two years of guidance, inspiration, and support. He always discusses the research topics with me in great details and give me valuable directions. Without his support, this work would not have been possible. Meanwhile, I must add my appreciation to Dr. Son,Vuong, my co-supervisor, who supported me in the first phase of my research work and gave me the maximum freedom in my research.

I am also grateful to Dr. Norm Hutchinson, the second reader of my thesis, for his technical and editorial support. Dr. Hutchinson agreed to read my thesis in his sabbatical days, his kindness and valuable suggestions have helped me a lot.

I would also like to thank Ms. Hermie Lam, Ms. Holly Kwan, and Ms. Joyce Poon, for all the help they have provided. Moreover, I would like to extend my thanks to all members of the DB lab and the NIC lab for the useful feedback and fun times.

Most of all, I would like to thank my family for their unconditional love and support.
Chapter 1

Introduction

In this thesis we will be working on the view management problem which emerges in building an effective and efficient social networking system. The system we are building supports not only facilities for social networking and community building, but also supports transactions including listing items (e.g., books, DVD players, and lawnmowers) that you wish to trade with, and creating a list of items you wish to buy/borrow (i.e., your wish list). In particular, it allows the users to set up flexible control over access to their resources or even links, and permits efficient querying of social neighborhoods for potential contacts, items of interest, etc., taking access control into account. We call a system featuring these functionalities a social network database system. In the first section we will briefly introduce the system model, then the visibility view in particular. In the next section we formally introduce the view selection problem and its connection with the file caching problem, and some related work done on them.

1.1 The Social Networking System

Recently, there has been a spurt of activities in the development of popular web sites that enable the sharing/exchanging of ideas, opinions, bookmarks, and blogs, as well as personal multimedia data such as pictures, music, and video, between like minded individuals. In addition, some systems support online communication such as instant messaging. Examples include MySpace [4], Flickr [2], YouTube
[6], Del.icio.us [1], LinkedIn [3], and Yahoo!360 [5]. All of these exploit an underlying general framework of a social network. The Wikipedia defines a social network as “a social structure made of nodes which are generally individuals or organizations. The nodes are connected through various social formalities ranging from casual acquaintance to close familial bonds.” The term was first coined by Barnes in 1954 in his classic work [8]. Some of the systems above emphasize the social aspect more than others, but in the end, they all use a social network of some kind as the backbone.

A generic social network system should allow the users in the network to own and share resources like images, videos, etc., and define their neighbors with tags created by themselves. In addition, the system should support item transactions (rent, borrow, etc.) and efficient matching of the users wish items and the items listed by other users. Specifically, the users should be able to control who can access their resources so that when other users are querying the resources, only those visible ones will be returned. For example, the following query should only return the items within Mary’s 2-degree network that she can see:

\[ Q1 : \text{"find all items that are listed by Mary’s 2-degree network, are visible to Mary, and satisfy the constraint } item\text{Name} = \text{DVD Player } \& \text{ year} > 2000." \]

The system should be able to efficiently process ad hoc queries using constraints on the item properties as well as constraints on users/nodes based on their social neighborhood, taking visibility into account. Thus we require a model which permits effective representation of information and a query language which is easy to use, is powerful and captures use cases such as the above naturally, and admits efficient evaluation. So we will briefly introduce our model and language in the following subsections.

1.1.1 Logical Model

Our logical model for a social network database system consists of a directed graph \( G = (\mathcal{V}, \mathcal{E}) \), whose nodes \( \mathcal{V} \) correspond to individuals that we refer to as users. There is an edge \((u, v) \in \mathcal{E}\) whenever \( u \) and \( v \) are involved in some kind of social
relationship. We henceforth refer to edges as links. We assume the graph is non-symmetric: the link \((u, v)\) is in \(\mathcal{E}\) doesn't mean that \((v, u)\) is also in \(\mathcal{E}\). This is to allow one-way relationship, e.g., one user may use another user's articles as her/his web feed without building a two-way relationship.

We assume that there is a "user profile" associated with each node. In addition, each node contains a collection of resources, which could be electronic belongings such as images, music, etc., or the items the user listed for trade, or wishlists regarding the items the user wants to get. A node may have zero or more resources. The user profile, resources and links are all modeled by a set of attribute-value pairs of the form \(A = v\), together with some keywords. An example of an item is "DVD, Oscar nominee, Denzel Washington, thriller, Year = 2006, Title = Deja Vu, Rating = five star, Director = Tony Scott." There is no assumption about the "types" of items nor their schema. E.g., different instances of DVD in the same or different nodes may have different sets of attributes defined for them. The set of keywords or the set of attribute-value pairs describing an item may be empty.

Specifically, we use a kind of link attribute called link labels or tags to model the relationships. E.g., a link \((u, v)\) with label friend indicates that \(u\) considers \(v\) its friend. We permit any number (no less than one) of tags on a link. E.g., a link \((u, v)\) with labels \{friend, doctor\} means \(v\) is \(u\)'s friend as well as doctor.

The one-degree network of a user \(u\) is the set of all users adjacent to it, i.e., \(\text{onedegree}(u) = \{v \mid (u, v) \in \mathcal{E}\}\). We regard a user as the owner of its one-degree network, with the user at the center and the members at its spokes. For any \(k \geq 1\), the \(k\)-degree network \((k\text{degree}(u))\) and any degree network \((\text{anydegree}(u))\) are defined in the obvious way: \(k\text{degree}(u)\) consists of nodes reachable from \(u\) by paths of length at most \(k\), while \(\text{anydegree}(u)\) consists of nodes reachable from \(u\). By convention, we exclude \(u\) from these networks as a technical convenience. Notice that the \(k\) degree network of \(u\) includes nodes reachable by paths of length \(< k\).

Visibility in this model is achieved by permitting a user to dictate who can possibly access its items, wishes or other resources. The specification can take the form of a list of users or predefined groups (like Yahoo! groups). Additionally, we permit the notion of visibility views, which are queries expressed in a simple intuitive query language, to be discussed shortly.
1.1.2 Visibility Views

Visibility of a user's items and wishes is controlled by the user by means of specifying a visibility view. In addition, a user may want to dictate who can see its outgoing links. As an example of the latter, John has an outgoing link to Mary labeled closefriend which he wants to be visible to only his close friends and their close friends.

We achieve both of these by means of visibility views. A visibility view for a specific user is a restricted regular path expression $\mathcal{D}$ defined as follows:

$$
\begin{align*}
\mathcal{I} &= \text{tag} | \Sigma | \text{tag}^* | \Sigma^* \\
\mathcal{I} &= \mathcal{I} \cdot \mathcal{I} \\
\mathcal{D} &= \mathcal{I} | \mathcal{I} + \mathcal{D} \\
\mathcal{D} &= \mathcal{D} | 'private' | 'public'
\end{align*}
$$

In the above definition $\cdot$ means concatenation, $+$ means alteration, $*$ is the Kleene star, and $\Sigma$ is the alphabet of all tags in the system. Basically, $\mathcal{D}$ represents queries which are either the constant visibility specs 'public' and 'private' or a union of conjunctive queries, and the conjunctive term could be a specific link tag, a wildcard tag, and the transitive closure a specific link tag or the wildcard tag. This definition allows a limited form of recursion, e.g., a recursive view for user Mary might be:

Mary: friend$^*$ · doctor

Informally, the semantics of a visibility view $\mathcal{D}$ at a node $u$ is the set of nodes related to $u$ in a manner that satisfies $\mathcal{D}$. More precisely, $|\mathcal{D}(u)|$, the set of nodes satisfying the expression $\mathcal{D}$ at node $u$, is defined as follows.

- When $\mathcal{D}$ is 'public', $|\mathcal{D}(u)| = \{v \in G\}$, i.e., all nodes in the network.
- When $\mathcal{D}$ is 'private', $|\mathcal{D}(u)| = \{u\}$.
- When $\mathcal{D}$ is $\Sigma$, $|\mathcal{D}(u)| = \{v \in G \mid v$ is the one degree network of $u\}$
• When $\mathcal{D}$ is $\Sigma^*$, $|\mathcal{D}(u)| = \{v \in G \mid v \text{ is reachable from } u\}$.

• When $\mathcal{D}$ is $t$, where $t$ is a tag, $|\mathcal{D}(u)| = \{v \in G \mid \exists \text{ a link } (u, v) \in G, \text{ with tag } t\}$.

• When $\mathcal{D}$ is $t.C$, where $t$ is a tag, then $|\mathcal{D}(u)| = \{v \in G \mid \exists w \in G : G \text{ contains a link } (u, w) \text{ with label } t \& v \in |C(w)|\}$.

• When $\mathcal{D}$ is $C_1 + C_2 + \cdots + C_k$, $|\mathcal{D}(u)| = |C_1(u)| \cup \cdots \cup |C_k(u)|$.

The most expensive part of evaluating queries with visibility constraints is the evaluation of visibility views, or more precisely, finding these graph patterns described by the regular expressions. This is because a regular expression can describe arbitrary long paths in the graph which means in turn an arbitrary number of physical access to the database. The evaluation of the views defined here shares many similarities with the XML regular path query evaluation. The ability to answer a query using the answers to views can significantly decrease query evaluation time. For example, in query Q1 if all the visibility rules in Mary’s 2-degree network are evaluated and materialized, then the evaluation process just needs to check whether Mary is in the resulting node sets of the resources within Mary’s 2-degree network, and does not need to evaluate the rules from scratch. If the views materialized can not completely answer a query, they can, nevertheless, speed up query processing, because some views needed for the query processing may be obtained through other views using the query rewriting technique. On the other hand, view maintaining can be computationally expensive, and the amount of space that is available to store view results could be bounded. These constraints lead to the natural optimization problem, known as the view selection problem, which is, roughly speaking, asking for a set of views that can answer given queries and satisfy specified constraints while minimizing the average expected cost of query evaluation. In the next section we will formally introduce the problem and its related work.

1.2 Problem Definition and Related Work

As mentioned in last section, query evaluation can be thought of as checking whether the user that issued the query belongs to the visibility view associated with
each item satisfying the query. Thus the view selection problem aims at reducing the query answering cost (in terms of the query execution time). As a convenience for theoretical analysis, we can regard that the queries are made of a very long view request sequences. The objective is to minimize the expected cost of answering any view request. The view selection problem is defined formally as follows:

**Definition (The Static View Selection Problem)** Given a set $V$ of $N$ views $v_1, v_2, \ldots, v_N$, where each view $v_i$ has the following specified functions

- size $|v_i|$;
- evaluation cost $c_i$;
- retrieval cost if materialized $r_i$;
- probability estimation $\beta_i$ (subject to $\sum_{i=1}^{N} \beta_i = 1$)

and possibly a partial order relation on $V$ representing the dependence relation among views, find a subset $S$ of $V$ to materialize, such that the average time taken to evaluate a view is minimized, with the constraint that the total size of $S$ is no more than an integer constant $K$.

This problem emerged in answering OLAP queries in the data warehouse environment, whereby the views form a data cube lattice. Plenty of work has been done on it, such as [21],[20],[19],[23],[18]. Among them, the pioneering work that established the problem was [21], most of others were follow-up works based on it. In that paper the authors argued that the problem, with the dependence relation existing among the views, is NP-complete even when the view sizes and probabilities are uniform. In our case, however, a visibility view for a specified resource can be treated as a unit though it can also be decomposed to subviews among which dependence relation exists. Since solving the problem of no dependency can be seen as a basis for solving the problem of the dependent case, we will start from the simpler case and only talk about the no dependence case in this thesis.

The static view selection problem is closely related to the (file) caching problem defined below.
**Definition (The Caching Problem)** 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 $|v_i|$;
- retrieval cost from remote server $e_i$;
- retrieval cost from the cache $r_i$;
- possibly a probability estimation $\beta_i$ (subject to $\sum_{i=1}^{n} \beta_i = 1$)

and a sequence of file requests, maintain files in the cache of size $K$ to satisfy the requests while minimizing the total (or average) retrieval cost.

This is a well-studied problem in both the systems and databases area since it plays a key role in many applications from the memory replacement policies in operating systems to the web proxy file caching. To allow the term to be general enough to catch all these applications, we simply call it the "caching problem". The caching problem can have objectives other than the cost, such as the byte hit rate, network traffic, etc. However in this thesis we mainly consider the total/average cost criteria for the queries since it is a most natural and commonly used criteria. Unlike the static view selection problem, the caching problem aims at finding a replacement algorithm/policy that can dynamically adjust the cache content so as to minimize the total/average cost. So we can find out that the two problems are indeed seeking the same goal but with different ways which are dedicated by the application requirements.

We now take a look at several variants of the caching problem and introduce some related work. In order to understand how good a replacement algorithm/policy is, [35] introduced the notion of competitiveness here:

**Definition (Competitiveness)** A replacement policy $g$ is said to be $k$-competitive if the following holds:

$$\text{cost}(g) \leq k \cdot \text{cost}(OPT) + c$$

where OPT is the optimal policy and $c$ is a constant.
The notion of competitiveness is sometimes referred to as approximation. Following the conventions, we will use the word “competitive” for dynamic replacement policies and “approximate” for the static selection algorithms.

Uniform sizes, uniform costs  With the restrictions that all file sizes and costs are the same, the problem is just paging. Paging requires mandatory admission, i.e., every requested page must be admitted into the cache. Sleater and Tarjan showed that Least Recently Used (LRU) and First In First Out (FIFO) are $K$-competitive, assuming the optimal algorithm uses the same size of cache. Here $K$ is the size of cache. They also showed that $K$ is the best competitive ratio for deterministic on-line algorithms. The algorithm Least Frequently Used (LFU) evicts the page in the cache that was requested the fewest number of times while not considering the recency information. LFU seems to perform well on Web request sequences, but it is not a competitive policy if the query sequence does not presume some distribution pattern. Lee et al [26] designed a spectrum of hybrid policies between LRU and LFU. They named these policies LRFU$_{\lambda}$ with the parameter $\lambda$ varying between $\lambda = 0$ (LRU) and $\lambda = 1$ (LFU). Cohen [15] proved that the competitive ratio of LRFU is $K + \left[\frac{\log(1-\lambda)}{\log\lambda}\right] - 1$, where $0.5 \leq \lambda < 1$ is the decay parameter used by the LRFU algorithm, and $K$ is the size of the cache. O’Neil presented an algorithm called LRU-K in [27] which keeps track of the times of the last $K$ references to popular pages.

Uniform sizes, arbitrary costs  This special case is called weighted caching. There are two well known algorithms for this problem: Balance[11] by Chrobak, Karloff, Payne, and Vishwanathan, and GreedyDual[34] by Young. Both of these two algorithms were shown to be $K$-competitive.

arbitrary sizes, cost=1 or cost=size  This is motivated by the importance of file size in caching for world-wide-web applications. For this problem, Irani gave $O(\log^2 k)$-competitive algorithms for both the cost=1 and cost=size cases[22].
**arbitrary sizes, arbitrary costs** This problem is called *file caching* or *web caching*. Interestingly enough, this problem was also shown to have solution with the competitive ratio $K$. In [10] Cao and Irani presented the algorithm GreedyDual-Size, in another independent work, Young brought up a similar algorithm LANDLORD, and proved it is $K$-competitive using potential functions.

The research results introduced above were mostly competitiveness results when the future is unknown, i.e., they were focusing on finding the competitive algorithms/policies. For most of the problems, it was shown that the best competitive ratio for deterministic algorithms is $K$ where $K$ is the cache size. We need to mention that when the sizes are arbitrary, the competitive ratios of these algorithms are with respect to the *integral* optimal solution in which no partial views can be selected/admitted. Some policies such as the Least Frequently Used would perform arbitrarily bad for some specific query sequence, because the query sequences are not confined to any rules or probabilistic models and it's always possible to make up a sequence that is bad for it while good for the optimal solution. But what if we knew the future sequence (offline case), or even some partial information about it? Can we find out the optimal policies for it?

Belady showed an optimal offline algorithm $B_0$ for the paging problem in his seminal work [9]. 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 query sequence. But as far as the author is aware, for the arbitrary sizes, arbitrary costs case, there is no simple algorithm known to be optimal except the brute force dynamic programming.

To study the performance of paging algorithms in terms of expected cost, Coffman and Denning introduced the Independent Reference Model [13]. It was a simple probability model under which optimal policies (in terms of expected cost) can be found. The model assumes the query sequence $Q_1Q_2\ldots Q_t$ is a sequence of independent random variables with the common, stationary distribution $\phi(\cdot) = \beta_1, \beta_2, \ldots, \beta_n$ such that $P[Q_t = i] = \beta_i$ for all $t \geq 1$. Though not a realistic representation of query 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 [13]. $A_0$ prescribes: When the cache is full,

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

The idea behind policy $A_0$ comes in conforming with policy $B_0$. Let the random variable $d_i(x)$ denote the forward distance to page $x$ just after $Q_t$. Under the independent reference model, $d_i(x)$ follows the stationary geometric distribution

$$P[d_i(x) = k] = \beta_x (1 - \beta_x)^{k-1}, \ k = 1, 2, \ldots$$

The mean value of $d_i(x)$ is $1/\beta_x$, so evicting the file with smallest $\beta_x$ is equivalent to evicting the file with the greatest expected forward distance.

$A_0$ is optimal only for the paging problem. But for either the view selection problem or the file caching problem, both the sizes and costs are important. Fiat and Rosen [17] showed that caching policies adapted from memory management applications that don't take size into consideration do not work well in practice. Costs are important as well. In many cases, the costs are neither uniform nor proportional to the size. An example might be the views containing the Kleene closures, e.g. the view $\text{Mary} : \text{closefriend}^*$ might have a much higher cost but smaller size than view $\text{Mary} : \Sigma$.

Bahat and Makowski showed a policy $C_0^*$ for the uniform sizes and arbitrary costs case:

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

Unlike the paging and weighted caching problem, $C_0^*$ assumes optional eviction. Policies with mandatory eviction are subsets of the policies with optional eviction, therefore $C_0^*$ is no worse than the policy $C_0$ described below:

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

Bahat also discussed the problem of finding optimal policies with multi-criteria aspects, specifically, 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, the arbitrary sizes, arbitrary costs case requires the total size of the chosen views cannot exceed a certain bound at any time, which is a stronger constraint than the long run constraint. Thus using the average cache size as the long-run average metric doesn’t solve the problem.

Seeing $C_0$, it is tempting to think that the following offline algorithm is optimal for deterministic weighted caching:

**Algorithm A** At time $t$, evict the view with smallest $(e_i - r_i) / dist(t)$, where $dist(t)$ is the forward distance of view $v_i$ at time $t$.

However we can prove that it is not the case. Consider the following instance:

Suppose there are two algorithms A and A', and they behave the same from time 1 to time $t-1$, so the cache content for A and A' at the beginning of time $t$ are the same, denoted as $S_t$. At time $t$ a fault occurs by request $q_t = v_i$. Suppose in $S_t$, $v_1$ is the view with smallest $(e_i - r_i) / dist(t)$ in $S_t \cup q_t$. Algorithm A evicts $v_1$ and A' evicts some other view $v_2$, and from now on the two algorithms behave the same and there’s no fault except at some time $v_2$ or $v_1$ occurs again. Then at time $t$ the information regarding $v_1$ and $v_2$ is in Table 1.1

<table>
<thead>
<tr>
<th>view</th>
<th>size</th>
<th>$e_i - r_i$</th>
<th>$dist(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>1</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>$v_2$</td>
<td>1</td>
<td>20</td>
<td>15</td>
</tr>
</tbody>
</table>

**Table 1.1:** An instance where Algorithm A is not optimal

In this query sequence instance, $v_1$ and $v_2$ are requested just once from time $t$ on at distances 30 and 15 respectively. So A and A' will both fault once but the cost for A is 30 and for A' is just 20, thus showing that the algorithm evicting the view with smallest $(e_i - r_i) / dist(t)$ is not offline optimal.

In this thesis we address the problem one step further: to assume both the sizes and costs are arbitrary. We make the following contributions:

1. We will reveal the intrinsic relationship between the static view selection problem and the caching problem. For any replacement policy (random or deterministic) under the independent reference model, we show that the optimal long-run average metric cannot be better than that of the static selection
algorithm. We systematically study the problems and present the essence of it, which is not found in any previous works to the best of the author’s knowledge.

2. We then present a 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 policy $A_0$ 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.

3. Because the intrinsic hardness of the problem when we introduce the arbitrary sizes, this optimal policy runs in non-polynomial time. Then we bring up $K$-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 $H_{K'}$-approximate to the integral optimum solution, where $K'$ is the total size of all objects minus $K$ and $H_{K'}$ is the harmonic number of $K'$.

The thesis is organized as follows: In Chapter 2, 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. We show that the optimal long-run average metric for any replacement policy (random or deterministic) under the independent reference model cannot be better than that of the static selection algorithm. This reveals the essence of the problem, which was not found in any previous works to the best of the author’s knowledge. Then we present optimal policies for the arbitrary sizes, arbitrary costs case in Chapter 2 and show that policy $A_0$ and $C_0$ are special cases of it. In the next chapter, we will present our approximate static selection algorithms and 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 $\sum_{i=1}^{N} |v_i| - K$. 

12
Chapter 2

Optimal Algorithms Under IRM

In Chapter 1 we have briefly introduced the static selection problem and the caching problem and browsed their similarities and differences. In this chapter we will further discuss the relationships of the two problems under the IRM model.

In Section 2.1 we firstly look at the static selection problem and show that minimizing the expected cost is equal to maximizing the benefit of the cached objects no matter whether there exists dependence relation among the views or not. In this section we will examine the problem without considering the dependence relation first since it provides a basis for the more complex case where dependence relations do exist. Let's call this restricted form of problem the naive view selection problem. We show that the naive static selection problem under IRM is a restricted Integer Covering problem and also the complement of the Knapsack Problem, then we review the well-known dynamic programming solution that can achieve optimum.

In Section 2.2 we will present optimal replacement policies for the caching problem. A solution can be seen as a sequential decision making process and hence can be modeled by a Markov Decision Process (MDP). However, the MDP described in [7] assumed uniform sizes, so any actions at any time evicts at most one view. We use a slightly different MDP to incorporate the arbitrary sizes, namely, to allow the system to admit or evict multiple objects. We will describe the MDP framework first, then show that the optimal long-run average reward for any replacement policy (random or deterministic) under IRM cannot be greater than the optimal benefit \( B_{\text{max}} \) the dynamic programming algorithm produces for the naive
view static selection problem. Later on we will show that this optimal value can indeed be achieved by some policies inducing a unichain with one closed set and some transient states. We also show that only such policies are average optimal, policies inducing irreducible chains or multichains are both non-optimal. Hence we will focus on policies inducing unichains and present one such policy, $D_0$. In addition, we show that policy $A_0$ and $C_0$ are both special cases of $D_0$. However $D_0$ is not a demand policy thus it's not suitable for web proxies, so we present another version of the policy $D_0'$ that is a demand policy. $D_0'$ is no worse than $D_0$.

### 2.1 The Static Selection Problem

In Chapter 1 we introduced the Static View Selection or Eviction Problem. For convenience, we state this problem here again:

**Definition (The Static View Selection or Eviction Problem)** Given a set $V$ of $N$ independent views $v_1, v_2, \ldots, v_n$, where each view $v_i$ has the following specified functions

- size $|v_i|$;
- evaluation cost $e_i$;
- retrieval cost if materialized $r_i$;
- probability estimation $\beta_i$ (subject to $\sum_{i=1}^{N} \beta_i = 1$)

find a subset $S$ of $V$ to materialize, such that the average time taken to evaluate a view is minimized, with the constraint that the total size of $S$ is no greater than an integer constant $K$.

The size $|v_i|$, evaluation cost $e_i$ and probability estimation $\beta_i$ of a view $v_i$ are usually obtained from the query frequencies observed in a running system. This is essentially assuming that future queries will follow some fixed and static distribution, which is analogous to the independent reference model. For some applications this model is reasonable, for example, the OLAP queries in data warehouses usually exhibit some stable patterns. Let the random variable $C_5$ be the cost of
answering a view request (or equivalently, a query) with cache content $S$. Our objective is to minimize the expected query answering time $E(C_S)$ for a query. To do this, we have the following important observation:

$$E(C_S) = \sum_{v_i \in S} \beta_i r_i + \sum_{v_i \notin S} \beta_i e_i$$  \hspace{1cm} (2.1)

Our objective is to minimize $E(C_S)$ in the above Equation subject to the condition that $\sum_{v_i \in S} |v_i| \leq K$. For most of the cases, the evaluation cost $e_i$ is larger than the retrieval cost if $r_i$ is materialized. This is a restricted integer covering problem. An integer covering problem is defined as follows:

**Definition Integer covering problem** Given an $n$ by $m$ matrix $A$, $n$ dimensional vector $b$, and $m$ dimensional vector $c$, the problem seeks to minimize $c^T x$ subject to $Ax \geq b$ over the non-negative integer vectors of length $m$.

We say that our problem is a restricted integer covering problem because the matrix $A$ in our case is just a vector. Hence we would expect to get simpler and better solutions than the full version of the problem when $A$ is a matrix. We will examine the optimal solution in this chapter and approximate solutions in Chapter 3.

The equation in Equation 2.1 can be written as:

$$E(C_S) = \sum_{i=1}^{N} \beta_i e_i - \sum_{v_i \in S} (e_i - r_i) \beta_i$$  \hspace{1cm} (2.2)

Because $N$ and $e_i$ are finite, the first half of Equation 2.2 $\sum_{i=1}^{N} \beta_i e_i$ can be deemed as a constant $\alpha$ which we call the *base evaluation cost*. It represents the evaluation cost for a query when nothing is materialized. We use $b_i$ to denote the benefit by caching $v_i$, $b_i = e_i - r_i$. Then the expected benefit of $S$, denoted as $E(B_S)$, represents the expected cost saved by caching the views in $S$, where $B_S$ is the random variable representing the benefit of answering a query using cache content $S$.

$$E(B_S) = \sum_{v_i \in S} b_i \beta_i = \sum_{v_i \in S} (e_i - r_i) \beta_i$$

Then Equation 2.2 can be written as:
Thus the problem of choosing a subset of views to minimize the expected cost can be transformed to the problem to maximize the expected benefit, while the total size constraint is satisfied. If we use $c_i$ to denote $(e_i - r_i)\beta_i$ the problem can be mathematically described as below:

$$\max \sum_{v_i \in S} c_i \quad \text{subject to } \sum_{v_i \in S} |v_i| \leq K$$

As we can see, this problem is exactly the Knapsack problem. It can be solved in pseudo-polynomial time using dynamic programming. Define a recursive function, $B(i, j)$ to be the maximum benefit that can be attained with the total size less than or equal to $j$ using items up to $i$. $B(i, j)$ is defined as follows:

$$B(0,j) = 0$$
$$B(i,0) = 0$$
$$B(i,j) = B(i-1,j) \text{ if } |v_i| > j$$
$$B(i,j) = \max \{B(i-1,j), c_i + B(i-1,j-|v_i|)\} \text{ if } |v_i| \leq j$$

The solution of optimal benefit can then be found by calculating $B(N, K)$. To do this efficiently we maintain a vector of size $K$ to store previous computations. This solution will therefore run in $O(NK)$ time. The different combinations of views whose total size doesn't exceed $K$ is at most $2^N$, therefore the total different value of $j$ is $2^N$. If $N$ is small enough to make $2^N < K$, it would require $2^N$ space instead of $K$. So the space complexity is $\min\{2^N, K\}$.

The value of $K$, to an extreme, is defined in terms of bytes so that it is an integer. But if using byte as the unit, it would be too slow to conduct the computation since the time complexity is in proportion to $K$. The unit cannot be too big either. For example, If we use 10MB as the unit but most of the view sizes are not multiples
of 10MB, then it would be inaccurate to use the above algorithm. Choosing the unit of the size should be done carefully, but in this thesis we will not put too much effort in it.

Now suppose we have got this maximum expected benefit $B_{\text{max}}$ under the cache capacity constraint,

$$B_{\text{max}} = \sup_S E(B_S) = \sup_S \sum_{V \in S} (e_i - r_i) \beta_i$$  \hspace{1cm} (2.5)

Then the minimum expected cost $C_{\text{min}}$ is just the complement of $B_{\text{max}}$ to the base evaluation cost $\alpha$:

$$C_{\text{min}} = \alpha - B_{\text{max}}$$

We call the cache content whose expected benefit is $B_{\text{max}}$ the optimal cache content $S^{OPT}$.

### 2.2 Finding The Optimal Policies For The Caching Problem

We design replacement policies so that the system knows how to make good decisions in the run. It can be seen as a sequential decision making and hence can be modeled by Markov Decision Process (MDP). However, the MDP described in [7] assumed uniform sizes, so the decision at any time evicts at most one view. We use a slightly different MDP to incorporate the arbitrary sizes.

#### 2.2.1 The MDP Framework

The query answering system maintains a dedicated piece of disk space which we call the cache to hold the materialized views. We assume the Independent Reference Model, where the queries for views form an infinite sequence with a static distribution. When a query arrives, the system would first check the cache. If the view is materialized, then its content is retrieved from the cache, and this is called a hit; otherwise it will be evaluated from scratch, which we call a fault. When a fault occurs, we must decide if this view is worth caching and some views in the cache
need to be evicted, and if yes, which ones. Note that we assume non-mandatory admissions, i.e., when a fault occurs, the requested view may not necessarily be admitted into the cache.

We will use the same notations as in the definition of The Caching Problem. For convenience we restate it below, but use the terms for views instead of files:

**Definition (The View Caching Problem)** Given a set $V$ of $N$ views $v_1, v_2, \ldots, v_N$, where each view $v_i$ has the following specified values

- size $|v_i|$
- evaluation cost from scratch $e_i$
- retrieval cost from the cache $r_i$
- probability of occurrence $\beta_i$ (subject to $\sum_{i=1}^{N} \beta_i = 1$)

and an infinite sequence of view requests, maintain the cache to satisfy the requests while minimizing the total (or average) query answering cost.

We assume the number of views $N$, the evaluation cost from scratch $e_i$, and the retrieval cost from the cache $r_i$ are all finite. $e_i$ is usually larger than $r_i$.

**Decision Epochs**

Decision epochs are defined as the instants when requests are posed. We use $T$ to denote the set of decision epochs. In our problem we assume infinite requests as

$$T = \{1, 2, \ldots\}$$

Elements of $T$ will be denoted by $t$ and usually referred to as "time $t$".

**State Sets**

At each decision epoch, the system occupies a state. In our work we define the state to be composed of the cache content information and the request.

We use $S_t$ to denote the cache content at time $t$. $S_t$ is a subset of $V$ with the constraint that its total size cannot exceed $K$: $\sum_{v_i \in S_t} |v_i| \leq K$. All the possible cache
contents form a set $S$, which is the state space of all cache contents satisfying the capacity constraint.

We use $Q_t$ to denote the request at time $t$. Request space $Q$ is made of all views $V$, so $Q = V$.

The MDP's state $\Theta_t$ at time $t$ is defined by the pair $(S_t, Q_t)$. We call the set of possible system states the state space and denote it $\Theta$. The state space $\Theta$ is then given by:

$$\Theta = \{S \times V\}$$

The number of elements in $S$, denoted as $|S|$, is at most $2^N$, and the number of elements in $V$ is $N$, so the total number of states $|\Theta|$ is at most $2^N N$. Though it is exponential in $N$, it is still finite. So our MDP is a finite state MDP.

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

**Definition (Group)** A group of states is a set of states sharing the same cache content.

From the definition of the MDP we know that each group contains $N$ states corresponding to the $N$ different view requests. Let $G_i$ denote the set of all states $\theta$ with cache content $s$:

$$G_i = \{\theta : \theta = (s, v_i), i = 1, 2, \ldots, N\}$$

**Action Sets**

When the request $Q_t$ ($Q_t$ is just one view, not a set) is made, the content of the cache is $S_t$. An action has to be made to decide the system behavior: whether to admit this view and evict some other views. We classify the situation into several cases:

1. $Q_t$ is materialized (which implies $e_t \geq r_t$);

2. $Q_t$ causes a fault and there is enough space to admit $Q_t$.
3. \( Q_t \) causes a fault and there is not enough space to admit it.

When case 1 happens, the request would be answered by retrieving the materialized view from the disk cache. When case 2 happens, all caching policies would just admit \( Q_t(= v_t) \) if \( e_t - r_t > 0 \). This strategy is always better than the one not to admit it, assuming the cost to evict the views is negligible. It is the different decisions made on case 3 that differentiate the policies. So when mentioning "actions", we are mainly talking about the actions taken on case 3.

For case 3, the system may decide to admit and evict some views. We denote the admitted view set at time \( t \) \( X_t \) and the evicted view set \( Y_t \). Unlike [7], both \( X_t \) and \( Y_t \) may contain multiple items including \( Q_t \) in our work. If at any time, \( X_t \) has only one view \( Q_t \), the policy is called a demanding policy [13], meaning that a view is only cached when requested. When \( X_t \) is \( \emptyset \), it means that the request is not admitted, and the \( Y_t \) should be \( \emptyset \) as well since we don't evict views when no view is to be admitted. \( X_t \) and \( Y_t \) cannot contain the same views at the same time. Thus given the state \( \Theta_t = (S_t, Q_t) \) the resulting state \( S_{t+1} \) can be written as

\[
\Theta_{t+1} = (S_{t+1}, Q_{t+1}) \quad \text{(2.6)}
\]

\[
= \begin{cases} 
(S_t, Q_{t+1}), & \text{if } Q_t \in S_t \\
(S_t + Q_t, Q_{t+1}), & \text{if } Q_t \notin S_t, |Q_t| \leq K - |S_t| \\
(S_t + X_t - Y_t, Q_{t+1}), X_t \cap Y_t = \emptyset, Y_t \subseteq S_t & \text{if } Q_t \notin S_t, |Q_t| > K - |S_t| 
\end{cases}
\]

We denote the actions available at time \( t \) \( A_t \) which is composed of \( X_t \) and \( Y_t \). The action space for state \( \theta \in \Theta \) is denoted as \( A_\theta \) and represents all possible actions for state \( \theta \). The action space for all states \( A \) is the superset of all \( A_\theta \) for all states \( \theta \) in the system.

In our MDP system \( X_t \) or \( Y_t \) can be any combinations of the elements in \( V \) with the constraint that their sizes cannot exceed \( K \), so the number of elements in any of them cannot exceed \( 2^N \). Thus the number of elements in \( A_\theta \) for any state \( \theta \) is at most \( 2^N 2^N = 2^{2N} \) which is also finite.
Rewards and Transition Probabilities

For any MDP system, as a result of choosing action $a \in A_\theta$ in state $\theta \in \Theta$ at decision epoch $t$,

- The system state at the next epoch is determined by a probability distribution
- The decision maker receives a reward $w_t(\theta, a)$ or a cost $c_t(\theta, a)$

We have defined the notion of group in Section 2.2.1. It is easily seen that for any state, when an action is made, the system would transfer to the $N$ states in some (not necessarily a different one) group with the probability distribution $\phi(\cdot) = \beta_1, \beta_2, \ldots, \beta_N$ defined in the Independent Reference Model respectively. So the transition probability from state $\theta_i$ to state $\theta_j$ for action $a$ at time $t$ is

$$p_t(\theta_j | \theta_i, a) = \beta_i$$

where the state $\theta_i = (S_{i+1}, v_i)$ is composed of cache content $S_{i+1}$ and request $v_i$.

Suppose at time $t$ the system is in state $\theta$ and action $a$ is taken, and $\theta = (S_t, Q_t)$, we define the one step reward $w_t(\theta, a)$ and one step cost $c_t(\theta, a)$ to be:

$$w_t(\theta, a) = \begin{cases} e_i - r_i & \text{if } Q_t \in S_t, Q_t = v_i \\ 0 & \text{if } Q_t \notin C_t \end{cases}$$

$$c_t(\theta, a) = \begin{cases} r_i & \text{if } Q_t \in S_t, Q_t = v_i \\ e_i & \text{if } Q_t \notin C_t \end{cases}$$

We can see that the reward or cost is only relevant to the state but not the action, then we can omit the parameter $a$ and write them as $w_t(\theta)$ and $c_t(\theta)$

The request $Q_t$ either causes a hit or a fault. If it is 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 $e_i$. Notice that our definition of the one step reward here is just the benefit $b_i$ of view $v_i$ we defined in Section 2.1.
Decision Rules

Decision rules range in generality from deterministic Markovian to randomized history dependent, depending on how they incorporate past information and how they select actions. A deterministic Markovian rule is a function \( d_t : \Theta \rightarrow A_\theta \), which specifies the action choice when the system occupies state \( \theta \) at decision epoch \( t \). For each \( \theta \in \Theta, d_t(\theta) \in A_\theta \). It is said to be Markovian (memoryless) because the decision only depends on the current state \( \theta \), and deterministic because it chooses an action with certainty. Furthermore, if the decision does not depend on time \( t \), then the rule is called stationary.

We call a deterministic decision rule history dependent if it depends on the past history of the system. That is, \( d_t \) is a function of the history \( h_t = (\theta_1, a_1, \ldots, \theta_t, a_t, \theta_t) \), where \( \theta_u \) and \( a_u \) denote the state and action at time \( u \).

Since we only consider deterministic algorithms in this thesis, we will only introduce the definitions of deterministic policies. We use the following notations to denote the variant set of policies:

- \( \Pi^{HD} \): history dependent and deterministic
- \( \Pi^{MD} \): Markovian and deterministic
- \( \Pi^{SD} \): Stationary Markovian and deterministic

The relationship among them are:

\[ \Pi^{SD} \subseteq \Pi^{MD} \subseteq \Pi^{HD} \]

2.2.2 The Objective Value of a Policy

The MDP we propose is discrete with infinite-horizon. Let policy \( \pi = \{d_1, d_2, \ldots\} \) denote a randomized history-dependent policy. For each \( t \), \( d_t : H_t \rightarrow A \), where \( H_t = \Theta \times A \times \Theta \times \cdots \times \Theta \) is the set of all histories of states and actions prior to decision epoch \( t \). For each history realized up to time \( T \), there corresponds a sequence of rewards

\[ \{r_1(\theta_1, a_1), r_2(\theta_2, a_2), \ldots, r_{T-1}(\theta_{T-1}, a_{T-1}), r_T(\theta_T, a_T)\} \]
Thus the random variables of states and actions induces a bivariate discrete-time reward process defined as \( \{ \Theta_t, w_t(\Theta_t, A_t) : t = 1, 2, \ldots \} \). \( \Theta_t \) represents the state of the system at time \( t \), and \( r(\Theta_t, A_t) \) represents the one-step reward received when using action \( A_t \) in state \( \Theta_t \), and its value follows the result we got in Section “Rewards and Transition Probabilities”. When \( t = 0 \), \( r(S_0, A_0) \) is denoted as \( r(S_0) \), representing the benefit obtained from the null action from a null state to the first state, and its value is the benefit made by \( R_0 \).

In the view cache management problem, we are mostly interested in the following values objectives:

1. The *expected total reward* \( tw^\pi(s) \) of policy \( \pi \in \Pi^{HR} \), or equivalently, the *expected total cost* \( tc^\pi(s) \)

\[
\begin{align*}
\quad tw^\pi(s) &= \lim_{T \to \infty} E_0^T \{ \sum_{t=1}^{T} w_t(\Theta_t, A_t) \} \\
\quad tc^\pi(\theta) &= \lim_{T \to \infty} E_0^T \{ \sum_{t=1}^{T} c_t(\Theta_t, A_t) \}
\end{align*}
\]

In our MDP, the one step reward \( w_t(\Theta_t, A_t) \) and one step cost \( c_t(\Theta_t, A_t) \) is irrelevant to whatever the decisions made, so we can denote the one step reward as \( w(\Theta_t) \) and one step cost as \( c(\Theta_t) \). So the above formulas can be written as

\[
\begin{align*}
\quad tw^\pi(\theta) &= \lim_{T \to \infty} E_0^T \{ \sum_{t=1}^{T} w(\Theta_t) \} \\
\quad tc^\pi(\theta) &= \lim_{T \to \infty} E_0^T \{ \sum_{t=1}^{T} c(\Theta_t) \}
\end{align*}
\]

2. The *average reward* of policy \( \pi \in \Pi^{HR} \): \( ar^\pi(\theta) \), or equivalently, the *average cost* \( ac^\pi(\theta) \)

\[
\begin{align*}
\quad aw^\pi(\theta) &= \lim_{T \to \infty} \frac{1}{T} E_0^T \{ \sum_{t=1}^{T} w(\Theta_t) \} \quad (2.9) \\
\quad ac^\pi(\theta) &= \lim_{T \to \infty} \frac{1}{T} E_0^T \{ \sum_{t=1}^{T} c(\Theta_t) \} \quad (2.10)
\end{align*}
\]
In the above equations, the parameter \( \theta \) is the starting state at time 1. In many applications the expected total cost diverges for some or all policies, but for our problem, the limit exists and is equal to \( \infty \) for all policies.

**Theorem 2.2.1**

\[ tc^\pi(\theta) = \infty \text{ for any policy } \pi \text{ and any starting state } \theta \]

**Proof** Define

\[ tc^\pi_\theta(\theta) = E_\theta^\pi \left\{ \sum_{t=1}^{\infty} r^+(\Theta_t, A_t) \right\} \]

and

\[ tc^\pi(\theta) = E_\theta^\pi \left\{ \sum_{t=1}^{\infty} r^-(\Theta_t, A_t) \right\} \]

where \( r^-(\Theta_t, A_t) = \max \{-w_t(\Theta_t, A_t), 0\} \) and \( r^+(\Theta_t, A_t) = \max\{w_t(\Theta_t, A_t), 0\} \). The theory of MDP [29] prescribes that if for any policy \( \pi \), for each state \( \theta \in \Theta \), either \( tc^\pi_\theta(\theta) \) or \( tc^\pi(\theta) \) is finite, then \( tc^\pi(\theta) = \lim_{M \to \infty} E_\theta^\pi \left\{ \sum_{t=1}^{M} w_t(\Theta_t, A_t) \right\} \) exists and satisfies \( tc^\pi(\theta) = tc^\pi_\theta(\theta) - tc^\pi(\theta) \).

In our case, the cost at each step is either \( e_i \) or \( r_i \) which are positive values, so

\[ tc^\pi_\theta(\theta) = 0 \]

and

\[ tc^\pi(\theta) = E_\theta^\pi \left\{ \sum_{t=1}^{\infty} r^+(\Theta_t, A_t) \right\} \geq \sum_{t=1}^{\infty} r_i, \text{ where } r_i \text{ is the request at time } t \]

\[ = \infty \]

thus the limit converges and the value is

\[ tc^\pi_\theta(\theta) - tc^\pi(\theta) = \infty - 0 = \infty \]

Since the expected total reward is \( \infty \) for all policies, it cannot distinguish the
policies. Therefore we will mainly focus on the average reward or the average cost. The average cost is in fact the expected cost to answer a query.

For either the total criteria or the average criteria over infinite horizons, maximizing the benefit is equivalent to minimizing the cost. For simplicity we will use the average reward criterion in this section.

2.2.3 Comparing the Objective Values

Both the problem of seeking optimal policies under the MDP framework and the problem of static selection assumes the same probability model – the Independent Reference Model. For either the total criteria or the average criteria over infinite horizons, we have known that maximizing the benefit is equivalent to minimizing the cost. As stated before we will examine only the expected average reward. Now let’s look at the objectives of the two problems:

1. The average reward objective for the dynamic policies

Recall the definition of average reward $aw^\pi(\theta)$ as follows

$$aw^\pi(\theta) = \lim_{M \to \infty} \frac{1}{M} E_0^\pi \{ \sum_{t=0}^M w_t(\Theta_t) \}$$

where $\theta$ is the starting state, and $\Theta_t$ is the random variable representing the system state at time $t$. We want to seek a method for computing the best reward

$$aw^{OPT}(\theta) = \sup_{\pi \in \Pi_{OPT}} aw^\pi(\theta)$$

and find a policy $\pi^{OPT}$ such that

$$aw^{\pi^{OPT}} = aw^{OPT}(\theta)$$

2. The expected benefit objective for the static selection solutions

Recall the definition of the benefit a cache content $S$ gets:

$$E(B_S) = \sum_{v_i \in S} b_i \beta_i = \sum_{v_i \in S} (e_i - r_i) \beta_i$$

25
We want to find the best subset $S^{\text{OPT}}$ that produces the maximum benefit $B_{\text{max}}$ such that

$$B_{\text{max}} = \sup_{S \subseteq V, |S| \leq K} E(B_S)$$

$$E(B_{S^{\text{OPT}}}) = B_{\text{max}}$$

where $|S|$ is the total size of the views in $S$.

The average reward $aw^\pi(\theta)$ of an MDP policy is the expected benefit of the gradually adapted states over the horizon for a "dynamic" policy $\pi$, while the expected benefit $B_S$ that the static selection algorithm seeks to maximize is for a static cache content $S$ to answer one query. The relation between the optimal average reward $tw^\pi(\theta)$ and $B_{\text{max}}$ is given in the following theorem

**Theorem 2.2.2** $aw^{OPT}(\theta) \leq B_{\text{max}}$ for any starting state $\theta$.

**Proof** From the definition of the average reward defined in Equation 2.9 we know

$$aw^{OPT}(\theta) = \sup_{\pi \in \Pi^R} \lim_{M \to \infty} \frac{1}{M} \sum_{t=0}^{M} E_\theta^\pi \{ w_t(\Theta_t) \}$$

$$= \sup_{\pi \in \Pi^R} \lim_{M \to \infty} \frac{1}{M} \sum_{t=0}^{M} E_\theta^\pi \{ w_t(\Theta_t) \}$$

From (Equation 2.5) in Section 2.1 we know that $B_{\text{max}}$ is the greatest expected reward to answer one query over all possible cache content $S$ under IRM, so we can assert that

$$B_{\text{max}} \geq E_\theta^\pi \{ w_t(\Theta_t) \}$$

for any starting state $\theta$ and any policy $\pi$ no matter what the distribution for the states $\Theta_t$ at time $t$ is. Thus we have

$$aw^\pi(\theta) \leq \lim_{M \to \infty} \frac{1}{M} \sum_{t=0}^{M} B_{\text{max}}$$

$$\leq B_{\text{max}}$$

$\blacksquare$

26
This theorem shows that under the Independent Reference Model, the average reward or even the total expected reward any policy can achieve cannot exceed that of the best view selection algorithm. The theorem also holds for the case where dependence relationship exist among the views. Our result is in contrast with the result in Dynamat [25] 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. The SPF algorithm they used is similar to our Greedy algorithm introduced in Chapter 3 in which we will show it is K-competitive. The authors of Dynamat examined the results by experiments. They used the parameter Detailed Cost Savings Ratio (DCSR) as the criterion:

$$DCSR = \frac{\sum b_i}{\sum e_i}$$

where $b_i$ and $e_i$ are similar to ours:

$$b_i = \begin{cases} 
0, & \text{if } q_i \text{ cannot be answered by } S \\
e_i, & \text{if there is an exact match for } q_i \\
e_i - e_f, & \text{if } f \text{ from } S \text{ is used to answer } q_i \end{cases} \quad (2.11)$$

From its literal meaning the DCSR is the benefit in our case. 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.

Static selection requires that all the information about the probability distribution, the view sizes, the evaluation costs and retrieval costs is known. It may look like a strong requirement, but the idea is applicable since while collecting the information of probability distribution in the probation period, other information such as sizes, evaluation costs can also be collected. In fact, both doing a static selection
or seeking a "dynamic" policy would require such information. In seeking a "dynamic" policy, we would have to know all the states, which essentially requires the same information. This result suggests that if we assume the independent reference model and if the applications can actively pick the objects, which is just the case for view selection problem, we can just use the static selection algorithms to get the optimal average cost.

Nonetheless, it is still valuable to study the dynamic policies although they are no better than the static selection under the Independent Reference Model. We believe the probability distribution is not fixed all time in practice and the materialized views should be managed as time goes by. In that case our knowledge in designing optimal policies under the Independent Reference Model will help us find good online policies. So in the next section we will be seeking optimal policies under IRM.

2.2.4 The Way to Calculate The Average Reward

Recall that the average reward \( aw^\pi(\theta) \) for starting state \( \theta \) was defined as follows

\[
aw^\pi(\theta) = \lim_{M \to \infty} \frac{1}{M} \mathbb{E}_{\pi} \left\{ \sum_{t=0}^{M} w_t(\Theta_t) \right\}
\]  

(2.12)

In some MDP systems, \( aw^\pi(\theta) \) need not exist for some policies, i.e., it may diverge. Fortunately researchers have found that when the state space and action space are finite or countable, the limit always exists under some stationary randomized policy \( \pi \) [30]. The question now is: How large will this limit be? In Theorem 2.2.2 we have seen that the average reward for any policy and any starting state is no better than the optimal expected benefit \( B_{\text{max}} \) to answer one query using the static selection. Can the optimal average reward of the MDP policies be equal to \( B_{\text{max}} \)? If yes, what kind of policies can achieve the optimum? To know that we will have to be able to calculate the average reward for a specified policy in the first place. Only after that can we compare the policies and find out the optimal ones.

However it is not easy to calculate the average reward for a specific policy \( \pi \) starting from the definition in Equation 2.12. A policy \( \pi \) induces an infinite
sequence $\omega$ of states and actions when the horizon is infinite:

$$\omega = (\theta_1, a_1, \theta_2, a_2, \ldots)$$

We call $\omega$ the sample path. All possible sample paths form the space of sample paths $\Omega$:

$$\Omega = S \times A \times S \times A \times \ldots$$

Then the expectation of the reward over the horizon is:

$$E_{\pi} \left( \sum_{t=0}^{M} w_t(\Theta_t) \right) = \sum_{\omega \in \Omega} (\sum_{t=0}^{M} w_t(\Theta_t))Pr(\omega) \quad (2.13)$$

where $\omega = (\Theta_1, A_1, \Theta_2, A_2, \ldots)$ is one sample path resulting in the states $\{\Theta_1, \Theta_2, \ldots\}$ by the action taken at time $t$, $A_t$, and $Pr(\omega)$ is the probability of the sample path $\omega$.

Equation 2.13 does not provide an efficient procedure for computing such expectations and ignores the dynamic aspects of the problem. However if $\{\Theta_t : t \in T\}$ is a Markov chain\(^1\), the expectation can be calculated by standard matrix methods in the MDP theory which we’re going to introduce.

Suppose $P_\pi$ is the one-step transition matrix of the induced Markov chain by this policy $\pi$, with $P_\pi^{(1)}(\theta_j|\theta_i)$ as its element at row $i$ and column $j$ representing the probability of one step state transfer from $\theta_i$ to $\theta_j$ by the decision of policy $\pi$ on state $\theta_i$, $\theta_i, \theta_j \in \Theta$. Let $P_\pi^*$ be the Cesaro limit:

$$P_\pi^* = \lim_{M \to \infty} \frac{1}{M} \sum_{i=0}^{M-1} P_\pi^{(i)}$$

(2.14)

where $P_\pi^{(i)}$ is the $i$ step transition matrix. Doob [16] and Chung [12] proved that the limit in Equation 2.14 always exists though the limit $\lim_{M \to \infty} P^{(M)}$ sometimes doesn’t. In addition, if the states in the chain are finite, $P_\pi^*$ must be stochastic, i.e., the value of the elements in each row sum to 1. Then by the MDP theory [29], the

\(^1\)Meaning the decision rules are Markovian, i.e., the action taken depends only on the current state.
average reward $aw^\pi(\theta)$ can be calculated using the following formula:

$$aw^\pi(\theta) = \lim_{M \to \infty} \frac{1}{M} E_{\alpha}^\pi \left( \sum_{t=0}^{M} w_t(\theta_t) \right) = P^\pi_\infty w^\pi(\theta)$$

(2.15)

where $w^\pi(\theta)$ is the one step reward for state $\theta$ under the stationary policy $\pi$. The proof of this formula can be found in Proposition (8.1.1) in [29].

Since $w^\pi(\theta)$ is known according to the definition in Equation 2.7, the only work left to get the average reward for a policy $\pi$ is to obtain $P^\pi_\infty$. $P^\pi_\infty$ has different looks when the induced chain has different structures, namely, irreducible, multichain, and unichain. We will be examining them one by one shortly. This will not only let us understand the MDP system better, but also help us design the optimal policies.

According to the Markov chain theory, a Markov chain can be classified into three categories: Irreducible, Unichain and Multichain. To understand these definitions, we need to introduce the notion of communicating class as defined in Markov chain theories:

**Definition Accessibility** A state $\theta_j$ is said to be accessible from a different state $\theta_i$ if, given that we are in state $\theta_i$, there is a non-zero probability that at some time in the future, we will be in state $\theta_j$. Formally, state $\theta_j$ is accessible from state $\theta_i$ if there exists an integer $n \geq 0$ such that

$$Pr\{\Theta_n = \theta_j | \Theta_0 = \theta_i\} > 0$$

Allowing $n$ to be zero means that every state is defined to be accessible from itself. The expression can also be written as

$$p^{(n)}(\theta_j | \theta_i) > 0$$

**Definition Communicating Class** A state $\theta_j$ is said to communicate with state $\theta_i$ if it is true that both $\theta_i$ is accessible from $\theta_j$ and that $\theta_j$ is accessible from $\theta_i$. A set of states $\Theta$ is a communicating class if every pair of states in $\Theta$ communicates with each other, and no state in $\Theta$ communicates with any state not in $\Theta$. It can be shown that communication in this sense is an equivalence relation.
Definition Closed Communicating Class/Closed Irreducible Set A closed class \( \Theta \) is a communicating class with the probability of leaving the class being zero, namely that if \( \theta_i \) is in \( \Theta \) but \( \theta_j \) is not, then \( \theta_j \) is not accessible from \( \theta_i \).

Definition Transient State A state \( \theta_i \) is said to be transient if, given that we start in state \( \theta_i \), there is a non-zero probability that we will never return to \( \theta_i \).

So we can classify the Markov chains as follows:

Irreducible A Markov chain is said to be irreducible if its state space is a communicating class; this means that, in an irreducible Markov chain, it is possible to get to any state from any state.

Unichain A Markov chain is said to be unichain if its state space is made up by one closed communicating class and some transient states.

Multichain A Markov chain is said to be multichain if its state space is made up by multiple closed communicating class and (possibly) some transient states.

One thing we need to mention is that not all MDP systems have policies inducing all three kinds of chains. It depends on the MDP's classification concerning the ergodic structure. But our MDP, being a general (or multichain) one, has policies that can induce all the chains defined above.

Before examining the limiting matrix for all three kinds of chains, we need to introduce some knowledge in Markov chains and linear algebra that will be used later. We can partition a chain into disjoint closed communicating classes \( D_l, l = 1, 2, \ldots, L, L \leq |\Theta| \), and possibly a set of transient states \( T \). After relabeling the states if necessary, we can express any transition matrix \( P \) as below if the policy \( \pi \) induces disjoint closed irreducible sets \( D_l, l = 1, 2, \ldots, L \) and possibly a set of transient states \( T \).

\[
P_\pi = \begin{pmatrix}
P_1 & 0 & 0 & \cdots & 0 \\
0 & P_2 & 0 & \cdots & 0 \\
& \ddots & \ddots & \ddots & \ddots \\
0 & \cdots & \cdots & \cdots & P_L \\
Q_1 & Q_2 & \cdots & Q_L & Q_{L+1}
\end{pmatrix}
\] (2.16)
where $P_1$ correspond to transitions between states in $D_1$ under policy $\pi$, $Q_1$ to transitions from states in $T$ to $D_1$, 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 [29]:

\[
P^*_\pi = \begin{pmatrix}
P^*_1 & 0 & 0 & \cdots & 0 \\
0 & P^*_2 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & & & P^*_L & 0 \\
Q^*_1 & Q^*_2 & \cdots & Q^*_L & Q^*_{L+1}
\end{pmatrix}
\]

where $P^*_i$ is the limiting matrix of $P_i$. Because every two states in a closed communicating class are commutable, the elements in $P^*_i$ must be positive. And since $P^*_i$ is stochastic, every $P^*_i$ must be stochastic.

Another fact we'd like to introduce for our MDP system is that the states within one group cannot belong to different disjoint closed communicating classes under any policy. This is because at any state $\theta = (s,v)$, when an action is taken, the system would transfer to the $N$ states of a group (not necessarily different from the current one) with probability $\beta_1$ to $\beta_N$. It is impossible for state $\theta$ to transfer to only part of a group. Hence if one state within a group belongs to a closed communicating class, then all the states in this group must also belong to the same closed communicating class.

Now we look at the three kinds of Markov chains one by one:

**Irreducible**

Suppose a stationary policy $\pi$ induces an irreducible Markov chain. Because the chain is just one communicating class, the Cesaro limit matrix $P^*_\pi$ for such a chain is stochastic and has the following property:

\[
\beta^*_\pi(s_j|s_i) > 0
\]

The chain consists of $|S|$ 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
$v_1$ to $v_N$. For each group, the limiting distribution for the $N$ states in it for any starting state must be linearly dependent with the vector $(\beta_1, \beta_2, \ldots, \beta_N)$, since 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 $\beta_1$ to $\beta_N$. So the $i^{th}$ row of the limiting matrix $P^*$ which representing the transition probability from state $\theta_i$ to all the states is in the following form:

$$[(a_{i1} \beta_1, a_{i1} \beta_2, \ldots, a_{i1} \beta_N), (a_{i2} \beta_1, a_{i2} \beta_2, \ldots, a_{i2} \beta_N), \ldots, (a_{i|S|} \beta_1, a_{i|S|} \beta_2, \ldots, a_{i|S|} \beta_N)]$$

subject to

$$\sum_{k=1}^{|S|} a_{ik} = 1, a_{ik} > 0$$

where $a_{ik}$ is the coefficient for row $i$ and the $k^{th}$ group.

The one-step reward $w_0(\theta, a)$ is independent of to time $t$ and action $a$ in our system so we simply denote it to be $w(\theta)$. Let's also order the states $\theta \in \Theta$ in the vector $w(\theta)$ first by the groups then by the requests from $v_1$ to $v_N$, just like what we did in the limiting matrix $P^*$, so that the resulting vector is in the form:

$$[(w(\theta_1), w(\theta_2), \ldots, w(\theta_N)), (w(\theta_{N+1}), w(\theta_{N+2}), \ldots, w(\theta_{N+N})), \ldots, (w(\theta_{|S|-1} \times N+1), w(\theta_{|S|-1} \times N+2), \ldots, w(\theta_{|S| \times N}))]$$

(2.19)

where $w(\theta_{(k-1) \times N+j})$ means the reward for 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 $\theta_i$ can be calculated as

$$aw^\pi(\theta_i) = \sum_{i=N,j=|S|} P^*_n \times w(\theta) = \sum_{k=1}^{N} \sum_{j=1}^{N} P^*_n(\theta_{(k-1) \times N+j} | \theta_i) w(\theta_{(k-1) \times N+j})$$

$$= \sum_{k=1}^{|S|} \left( a_{ik} \sum_{j=1}^{N} \beta_j w(\theta_{(k-1) \times N+j}) \right)$$
where $S_k$ is the cache content of group $k$ and $B_{S_k}$ is the expected benefit for cache content $S_k$. Because $a_{ik} > 0$ and there must be some $S_k$ such that $B_{S_k} < B_{\text{max}}$, so

$$aw^r(\theta_i) = \sum_{k=1}^{|S|} a_{ik}B_{S_k} < B_{\text{max}}$$

So the policy $\pi$ inducing an irreducible Markov chain must have an average reward smaller than $B_{\text{max}}$.

**Unichain**

If a policy induces a unichain $P_\pi$, i.e., the chain consists of one closed communicating class $D$ and a (possibly empty) set of transient states $T$, then $P_\pi^*$ has equal rows and

$$p_\pi^*(s_j|s_i) = \begin{cases} q_j & s_j \in D \\ 0 & s_j \in T \end{cases} \quad (2.20)$$

where $q$ satisfies $qP_\pi = P_\pi$ subject to $\sum_{s_j \in D} q_j = 1$ This result was known in MDP theory and can be found in [29]. Note that the closed communicating class may contain more than one group, but certainly not all the groups, otherwise it would become the Irreducible case. Now the problem remaining is to ascertain $q_j$'s values.

We first write the one step transition matrix into the canonical form as in Equation 2.16:

$$P_\pi = \begin{pmatrix} P_D & 0 \\ Q_1 & Q_2 \end{pmatrix}$$

where $P_D$ corresponds to transitions between states in the closed irreducible set $D$ under policy $\pi$, $Q_1$ to transitions from states in the transitive set $T$ to $D$, and $Q_2$ to transitions between states within $T$. We know that the limiting matrix $P_\pi^*$ must be the following form as shown in the Equation 2.17:

$$P_\pi^* = \begin{pmatrix} P_D^* & 0 \\ Q_1^* & Q_2^* \end{pmatrix}$$
Since we know that $P_D^*$ must also be in the form of Equation 2.20, we shall get the result that

- $P_D^*$ has same rows and the $j^{th}$ element in the row is $q_j$
- The rows in $Q_D^*$ are identical to the rows in $P_D^*$
- $Q_D^* = 0$

So in order to determine $q_j, s_j \in D$, it's enough to find out what $P_D^*$ is. Suppose the closed communicating class $D$ contains $b$ groups. Then the rows in $P_D^*$ must take a similar form as in Equation 2.18 for the same reason:

$$[(a_1\beta_1, a_1\beta_2, \ldots, a_1\beta_N), (a_2\beta_1, a_2\beta_2, \ldots, a_2\beta_N), \ldots, (a_b\beta_1, a_b\beta_2, \ldots, a_b\beta_N)]$$

subject to

$$\sum_{k=1}^{b} a_k = 1, a_k > 0$$

where $a_k$ is the coefficient for the $k^{th}$ group in $D$.

As in the last section, we write the one step reward vector in its canonical form:

$$[(w(\theta_1), w(\theta_2), \ldots, w(\theta_N)), (w(\theta_{N+1}), w(\theta_{N+2}), \ldots, w(\theta_{2N})), \ldots, (w(\theta_{(b-1)\times N+1}), w(\theta_{(b-1)\times N+2}), \ldots, w(\theta_{b\times N}))]$$

So the average reward for any starting state $\theta$ in the system has the same value and can be calculated as

$$aw^\pi(\theta) = P_D^* w(\theta)$$

$$= \sum_{k=1, j=1}^{b, N} p_D^* (\theta_{(k-1)\times N+j} | \theta) w(\theta_{(k-1)\times N+j})$$

$$= \sum_{k=1}^{b} (a_k \sum_{j=1}^{N} \beta_j w(\theta_{(k-1)\times N+j}))$$

$$= \sum_{k=1}^{b} a_k B_s_k$$

35
Specifically, if $D$ only contains one group $G_S$ where $S$ is a specific cache content, then $a_1 = 1$ and the rows in the limiting matrix would be:

$$[\beta_1, \beta_2, \ldots, \beta_N]$$

and the canonical one step reward vector would be:

$$[w(\theta_1), w(\theta_2), \ldots, w(\theta_N)]$$

so that

$$aw^\pi(\theta) = B_S$$

for every $\theta \in \Theta$.

This shows that the resulting average reward is only relevant to the groups in the closed communicating class $D$. If $D$ only contains one group $G_{S_{OPT}}$ where $S_{OPT}$ is the cache content that has the best benefit $B_{max}$, then the average reward for every starting state $\theta$ can achieve optimum.

Multichain

Suppose policy $\pi$ partitions the induced chain into disjoint closed communicating classes $D_l, l = 1, 2, \ldots, L, L \leq |\Theta|$, and possibly a set of transient states $T$. Write the one step transition matrix and one step reward vector in their canonical forms as in Equation 2.16 and Equation 2.19, then the average rewards for the starting state $\theta_l$ in $D_l, l \in [1, L]$ must be

$$\sum_{G_{S_k} \in D_l} a_{\theta_k} B_{S_k}$$

and smaller than $B_{max}$. Only in a closed communicating class $D_l = G_{S_{OPT}}$ can the average reward be $B_{max}$. We don’t even need to look at the limiting distribution for the transient states to conclude 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 $\pi$. The discussion lead to the following theorem:
**Theorem 2.2.3** The optimal average reward \( aw^{OPT}(\theta) \) a stationary policy can achieve is \( B_{\text{max}} \) for any starting state \( \theta \), 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 Section 2.2.4

Note that we only discussed the average reward. The same result applies to the average cost.

### 2.2.5 Finding The Optimal Policies

Like any other finite state MDPs, the optimal stationary replacement policy can be attained by value iteration (dynamic programming) as in Figure 2.1.

1. Define \( ar^{(0)}(\theta) = w(\theta) \), set the iteration index \( n = 0 \)
2. For each \( \theta \in \Theta \), compute \( ar^{(n+1)}(\theta) \) by
   \[
   ar^{(n+1)}(\theta) = \max_{a \in A} \{ w(\theta, a) + \sum_{\theta_j \in \Theta} p(\theta_j | \theta, a) ar^{(n)}(\theta_j) \} 
   \]
3. If \( ar^{(n+1)}(\theta) = ar^{(n)}(\theta) \), then go to step 4.
   Otherwise increment \( n \) by 1 and return to step 2.
4. For each \( \theta \in \Theta \), choose the decision
   \[
   d(\theta) = \arg \max_{a \in A} \{ w(\theta, a) + \sum_{\theta_j \in \Theta} \} 
   \]

**Figure 2.1:** The Value Iteration Algorithm

The value iteration process starts from \( n = 1 \) for all states, then \( n = 2, 3, \ldots \) until the values don't change any more. The policy that satisfies this equation is the optimal stationary policy, and from Section 2.2.4 we know it must induce a unichain Markov chain. When the execution of the value iteration algorithm is done, it would find out the optimal actions in each state, so that the system knows which action it should take when seeing various states in processing the query sequence.

However, the value iteration algorithm requires the information about all states
before it can answer the 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 views in cache and the requested one. This is because that when the sizes are uniform, stack algorithms like \( A_0, B_0 \) and \( C_0 \) can progressively converge to the optimum cache content by sorting the views in terms of the benefit density \( \beta_i(e_i - r_i)/|v_i| \) and evicting the smallest one when \( |v_i| \) is uniform. Coffman and Denning [13] gave the definition of stack algorithms as follows:

**Definition Stack Algorithms** An replacement policy \( A \) is called a stack algorithm if its cache content satisfies the inclusion property: For any query sequence \( \sigma = \{Q_1, Q_2, \ldots\} \) and any cache size \( k \)

\[
S_t(k, \sigma) \subseteq S_t(k + 1, \sigma)
\]

where \( S_t(k, \sigma) \) is the cache content for a cache with capacity \( k \) and query sequence \( \sigma \) at time \( t \).

In the case that \( |v_i| \) is uniform, the stack for the replacement policies are the sorted order of views (in descending order) on the benefit density \( \beta_i(e_i - r_i)/|v_i| \) where \( |v_i| = 1 \) for all \( i \). By evicting the views with smallest benefit density, the cache would gradually converge to the first \( K \) views, which is exactly the optimal cache content when the sizes are uniform. The action to evict the view with smallest \( \beta_i(e_i - r_i)/|v_i| \) when \( |v_i| = 1 \) is the optimal decision for every state, since the process of convergence is monotone, i.e., if a view is among the first \( K \) on the stack, then once it is requested, it would be admitted and never be evicted again.

However when the sizes \( |v_i| \) are arbitrary the case is different. As we have seen in Section 2.2.4 the optimal policies would converge to group \( G_{opt} \), but the optimal algorithms for this case doesn’t have the inclusion property. As a simple example, the cache content \( \{v_1, v_2\} \) is optimal for cache capacity \( k \), but the optimal cache content for cache capacity \( k + 1 \) might be a totally different set \( \{v_3, v_4\} \). So it is impossible to get the optimal policies using the principles of stacks, i.e., greedily evict the view with smallest density. To obtain the optimal policies, one has to use dynamic programming.

Now let’s analyze the complexities for the value iteration algorithm. In each
iteration, the algorithm needs to calculate the objective value for all states and all
actions. The number of states in the system is $|S|/N$, where $|S|$ is $O(2^N)$. The
number of possible actions for a state $\theta$ is $A_\theta$, which is also $|S| = O(2^N)$ since a
state can transfer to any other states in the system by our MDP definition. So the
total time complexity is:

$$I \times |S| \times |N| = O(I \cdot 2^{2N})$$

where $I$ is the number of iterations.

The space complexity of value iteration is $|S|/N$ and $O(2^N)N$.

The value iteration algorithm, though it can obtain the optimal stationary policy
that has maximum average reward, it doesn’t help us thoroughly understand
what the decisions are, and it is excessively expensive. Note that all these computa-
tions have to be done before answering the queries and this would make the value
iteration algorithm not very meaningful. Suppose that all the information is known
prior to the real queries, then why wouldn’t we just calculate what the optimal
cache content is and make the system jump into the states with that cache content?
If this is true then we’re essentially dealing with the static selection problem.

However we know that designing dynamic polices are still worth studying. To
design a dynamic policy, we can integrate the calculation in the process of query
answering. Now we will introduce a policy called $D_0$ in Figure 2.2. This policy,
like the dynamic programming algorithm for the Knapsack problem, keeps a vector
$B^{(n)}(j)$ of size $K$ at any time to hold the best selection so far in iteration $n$.

The algorithm reaches the optimal cache content when all the $N$ views 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. Though this policy
needs to remember the views seen in the past and hence is a history dependent
policy, it doesn’t incur more time or space complexity. The space consumption is
$K$ and with a little modification it can become $\min\{2^N, K\}$. For each iteration of
a new view, the algorithm incurs a cost of $O(K)$, so the total time complexity is
$O(N \min\{2^N, K\})$, much smaller than the value iteration.

The average reward of policy $D_0$ is also $B_{\text{max}}$. In fact the stochastic process
induced by this policy has the same limiting matrix $P_n^+$ as in Equation 2.20, since
1. Set $B^{(0)}(j) = 0$ for every $j \in [1, K]$

2. If a new view is requested, set $n = n + 1$, update for each $j$
   
   $B^{(n)}(j) = B^{(n-1)}(j)$ if $|v_i| > j$

   $B^{(n)}(j) = \max\{B^{(n-1)}(j), c_i + B^{(n-1)}(j - |v_i|)\}$ if $|v_i| \leq j$

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.

**Figure 2.2: The $D_0$ Algorithm**

The system would remain in the optimal groups whenever it reaches them, making the limiting probability of the transient states 0.

This policy, however, has a drawback that it is not a demand policy and may load views that are not the view request. These views might be evicted without being referenced when another new view is requested. Therefore we may want to use the demand version of the policy.

1. Set $B^{(0)}(j) = 0$ for every $j \in [1, K]$

2. If a new view is requested, set $n = n + 1$, update for each $j$

   $B^{(n)}(j) = B^{(n-1)}(j)$ if $|v_i| > j$

   $B^{(n)}(j) = \max\{B^{(n-1)}(j), c_i + B^{(n-1)}(j - |v_i|)\}$ if $|v_i| \leq j$

3. If $B^{(n)}(K) \neq B^{(n-1)}(K)$
   
   If the current request $Q_t \in B^{(n)}(K) - S_t$
   
   While the space is not enough

   Evict the view with smallest $\beta_i(e_i - n_i)/|v_i|$ from $S_t - B^{(n)}(K)$

4. Loop back to step 2 when seeing the next request.

**Figure 2.3: The $D^*_0$ Algorithm**

The $D^*_0$ policy in Figure 2.3 remembers the state of $D_0$ but defers the admis-
sions and evictions of views until they are requested. As proved in [13], the demand
version of a policy is no worse than itself in the total reward criteria. Furthermore,
it has the same optimal average reward as in $D_0$. However the convergence rate
is slower than $D_0$: the time to reach the optimal cache content is bounded to be
$2\sum_{i=1}^{N} 1/\beta_i$, which is at most two times of that of $D_0$. 
Chapter 3

Approximation Algorithms

In Chapter 2 we discussed the optimal static selection algorithm and “dynamic” policies. We have seen that when both the view sizes and costs are arbitrary, the problem is a restricted Integer Covering Problem and is NP-complete, so the time complexity to get the optimal solutions cannot be polynomial. Therefore it is necessary to study the polynomial time approximate or competitive algorithms, and these become the main topic in this chapter.

Knowing that the dynamic policies are dominated by the static selection under IRM from the last chapter, we will start from the static selection problem in Section 3.1. We will first show an polynomial algorithm called “Greedy” and prove it to be an $K$-approximate algorithm with respect to the optimal fractional solution. Then we show that it is in fact the “best” approximate algorithm because $K$ is the best approximation ratio for all polynomial algorithms. We also look back at the 2-approximate algorithm for knapsack and reveal that it is in fact also $K$-approximate for the view selection. After that we present a $H_K'$-approximate algorithm with respect to the optimal integral solution called “Extended Greedy”, where $K' = \sum_{i=1}^{N} |v_i| - K$. Sometimes the Extended Greedy algorithm can outperform both Greedy and knapsack’s 2-approximate algorithm, which will be shown in an example follows. At the end of the section we present a mediated algorithm that is superior to the above three algorithms.

In Section 3.2 we will discuss polynomial competitive policies under IRM. We will present a polynomial time policy that is $K$-competitive to the fractional opti-
mal average reward. The policy is analogous to the one used in Watchman [31], but in that paper the authors didn't give the analysis of the competitive bound. The authors argued that their policy was optimal with some additional constraints, however the constraints in fact removed the hardness of the problem and degrades it into the uniform sizes case. In contrast, we prove the policy to be \( K \)-competitive and \( K \) is the lower bound of all polynomial policies.

### 3.1 Approximate Static Selection Algorithms

In Section 2.1 we have seen that the static selection problem is actually a knapsack problem. The knapsack problem has a polynomial 2-approximation algorithm, and it was shown in [24] that a polynomial algorithm for the knapsack problem cannot achieve an approximation ratio smaller than 2. But here comes the distinction of maximizing the benefit and minimizing the cost. Does the approximation ratio on the benefit imply the same ratio on the cost? If it doesn't give the same ratio, does it guarantee any ratio at all? In the view selection problem for OLAP queries where the dependence relationships among views form data cubes, it was proved by Karloff and Mihail [23] that the problem is basically inapproximable for general partial order relationships under IRM, even though the approximate algorithms to get competitive benefit ratios do exist [21][32]. So we may be wondering if the same situation happens when there is no dependence. Fortunately we have found that this is not the case. In the following sections we will give a \( K \) approximate selection algorithm.

#### 3.1.1 A Greedy Algorithm With Bound \( K \)

We introduced the static selection problem in Section 2.1 and defined the objective value, the expected cost to answer one query, in a mathematical way in Expression (Equation 2.1). For convenience of finding the approximate algorithm, we now restate the problem using the integer programming formulas here:

**Definition** (The Static View Selection: An Integer Programming (IP) Formulation) Given a set \( V \) of \( N \) independent views \( v_1, v_2, \ldots, v_n \), where each view \( v_i \) has the following specified functions
• size $|v_i|$;
• evaluation cost $e_i$;
• retrieval cost if materialized $r_i$;
• probability estimation $p_i$ (subject to $\sum_{i=1}^{n} p_i = 1$)

The problem prescribes:

$$\text{Minimize } \sum_{i=1}^{N} (p_i e_i x_i + p_i r_i (1 - x_i))$$

Subject to $\sum_{i=1}^{N} |v_i| x_i \geq K'$,

where $K' = \sum_{i=1}^{N} |v_i| - K$ and $K$ is the cache size

$x_i = 0, 1; x_i = 0: v_i$ is cached; $x_i = 1: v_i$ is evicted

The objective value $\sum_{i=1}^{N} (p_i e_i x_i + p_i r_i (1 - x_i))$ in the above formulation can be simplified as:

$$\sum_{i=1}^{N} (p_i e_i x_i + p_i r_i (1 - x_i)) = \sum_{i=1}^{N} p_i (e_i - r_i) x_i + \sum_{i=1}^{N} p_i r_i$$

$$= \sum_{i=1}^{N} p_i (e_i - r_i) x_i + \delta$$

where $\delta = \sum_{i=1}^{N} p_i r_i$ is a constant

(3.1)

Thus the problem is simplified as the following if we use $c_i$ to represent $p_i (e_i - r_i)$ for each view:

$$\text{Minimize } \sum_{i=1}^{N} c_i x_i$$

Subject to $\sum_{i=1}^{N} |v_i| x_i \geq K'$,  

(3.2)
Note that our formulation here is different from that in Equation 2.3. In Equation 2.3 we represented the objective value as the benefit of the cache content taken off a constant $\alpha$, while here the objective value is some cost added to some other constant $\delta$. Let $\Gamma_\Phi$ denote the objective value $\sum_{v_i \notin S^\Phi} c_i x_i$ where $S^\Phi$ is the selected views by algorithm $\Phi$, if it achieves approximation ratio $\sigma$, which means $\Gamma_\Phi \leq \sigma \Gamma^*$ where $\Gamma^*$ is the optimal $\Gamma$, then the expected cost $E_\Phi(C) = \delta + \Gamma_\Phi$ for algorithm $\Phi$ must have the same ratio $\sigma$ because:

$$E(C) = \delta + \Gamma_\Phi$$
$$\leq \delta + \sigma \Gamma^*$$
$$\leq \sigma (\delta + \Gamma^*)$$
$$\leq \sigma E^*(C)$$

where $E^*(C)$ is the optimal expected cost to answer one query.

In order for the readers to understand the analysis of the approximate ratio, we will first introduce the linear programming relaxation and the fractional optimal solution of the problem.

**Linear Programming Relaxation and The Fractional Optimal Solution**

As we know, a standard approach for any integer program to get any insight into the structure of the problem and also to get a first estimate of the solution value is to relax the integrality constraint to a linear one. The linear programming relaxation (LP relaxation) of this problem can be defined as:

**Definition** *(The Static View Selection / Eviction Problem: An Linear Programming (LP) Formulation)*

Minimize $\sum_{i=1}^{N} c_i x_i$

Subject to $\sum_{i=1}^{N} |v_i| x_i \geq K^*$,  

$$x_i = 0, 1$$
\[ 0 \leq x_i \leq 1 \]

In this formulation \( x_i = 0 \) or \( x_i = 1 \) has the same meaning as in the IP formulation, but when \( 0 < x_i < 1 \) it means that only part of the view is chosen.

The solution of the LP problem can be computed in a very simple way as the global optimum can be obtained by making a series of greedy (local optimal) choices. Sort the views in terms of \( \frac{c_i}{|v_i|} \) in non-decreasing order. Without loss of generality, assume

\[
\frac{c_1}{|v_1|} \leq \frac{c_2}{|v_2|} \leq \cdots \leq \frac{c_N}{|v_N|}
\]

The greedy algorithm assumes all views are materialized initially, i.e., \( x_i = 0 \) for all views, then greedily evict the view with smallest \( \frac{c_i}{|v_i|} \) in each iteration, by evicting we mean that these evicted items’ \( x_i \) were set to be 1. If evicting a view would cause the capacity constraint to be satisfied, let us say in iteration \( \zeta \), then the execution is stopped and the residual space \( (K' - \sum_{j=1}^{\zeta-1} |v_j|) \) that must be “filled” by the evicted views is then filled by an appropriate fractional part of view \( v_\zeta \). \( v_\zeta \) is referred to as the split view. The optimal solution vector for the LP problem is defined in the following theorem:

**Theorem 3.1.1** An optimal solution vector \( x^{LP} = (x_1^{LP}, x_2^{LP}, \ldots, x_N^{LP}) \) is defined as:

\[
\begin{align*}
    \chi_j^{LP} &= 1, j = 1, \ldots, \zeta \\
    \chi_\zeta^{LP} &= \frac{(K' - \sum_{j=1}^{\zeta-1} |v_j|)}{|v_\zeta|} \\
    x_j^{LP} &= 0, j = \zeta + 1, \ldots, N
\end{align*}
\]

The corresponding solution value is

\[
\Gamma_{LP} = \sum_{i=1}^{\zeta-1} c_i + \frac{(K' - \sum_{j=1}^{\zeta-1} |v_j|)}{|v_\zeta|} c_\zeta
\]

The optimal solution for finding smallest \( \Gamma \) is just the complement of the optimal solution of finding the largest benefit caused by views not evicted, which is

46
the solution to a knapsack problem. The optimality proof of the knapsack problem was shown in [24], so we don’t repeat it here.

The Greedy K-Approximate Algorithm

Inspired by the optimal solution for the LP relaxation of the view selection/eviction problem, we now present an algorithm for the IP version of the problem in Figure 3.1.

1. Sort views in non-increasing order of \( \frac{c_j}{|v_j|} \).
   Assume the resulting order is marked by \( j, j = 1, 2, \ldots, N \)
2. Assume all views are cached, and evict the views ranked from 1 to \( \zeta \)

![Figure 3.1: Greedy Algorithm](image)

**Theorem 3.1.2** The greedy algorithm in Figure 3.1, denoted as \( G \), is \( K \)-approximate, i.e.,

\[
E_G(C) \leq KE_{LP}^*(C)
\]

where \( E_G(C) \) is the expected cost to answer a query by algorithm \( G \) and \( E_{LP}^*(C) \) is the fractional optimal expected cost.

**Proof**

\[
\Gamma_G = \sum_{i=1}^{\zeta-1} c_i + c_{\zeta}
\]

\[
\leq \sum_{i=1}^{\zeta-1} c_i + K \frac{(K' - \sum_{j=1}^{\zeta-1} |v_j|)}{|v_\zeta|} c_{\zeta}
\]

\[
\leq K(\sum_{i=1}^{\zeta-1} c_i + \frac{(K' - \sum_{j=1}^{\zeta-1} |v_j|)}{|v_\zeta|} c_{\zeta})
\]

\[
= K\Gamma_{LP}^*
\]

Together with the result in Equation 3.3, we can conclude that

\[
E_G(C) \leq KE_{LP}^*(C)
\]
3.1.2 The Lower Bound of The Approximate Ratios

We have showed that the greedy algorithm in Figure 3.1 has a competitive ratio \( K \). \( K \) might be a very large number and we wonder if this bound can be improved. In order to do that, we need to first introduce the integrality gap defined in [33].

Given a LP relaxation of any minimization problem, let \( \chi^*_L(I) \) 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 to be

\[
\sup_i \frac{\chi^*(I)}{\chi^*_L(I)}
\]

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. 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 3.1.3** The integrality gap on the expected cost for the view selection/eviction problem, defined as

\[
\sup_i \frac{E^*(C)(I)}{E^*_L(C)(I)}
\]

is at least \( K \).

**Proof** Consider the following instance:

| view | size \( |v_i| \) | cost \( c_i \) | \( c_i/|v_i| \) |
|------|----------------|-------------|--------------|
| \( v_1 \) | \( K \) | \( K \) | 1 |
| \( v_2 \) | 1 | \( K + 1 \) | \( K + 1 \) |

*Table 3.1: An instance where the integrality gap is \( K \)*

In this instance the system only has two views and both of their sizes are no greater than the cache size \( K \). Now we can calculate its optimal fractional solution \( E^*_L(C) \) and optimal integral solution \( E^*(C) \).
The optimal solution for the LP relaxation is to just evict part of \( v_1 \) and the result is:

\[ E^*_{LP}(C) = 1 \]

The optimal solution for the original IP problem is to evict the whole \( v_1 \) and the result is:

\[ E^*(C) = K \]

And the ratio of the optimal solution for the original IP problem to the optimal fractional solution for this instance is:

\[ \frac{E^*(C)}{E^*_{LP}(C)} = K \]

So the integrality gap \( \sup_i \frac{E^*(C)}{E^*_{LP}(C)} \) is at least \( K \). □

This theorem says that the best approximation ratio for the view selection/eviction problem is \( K \). It means no polynomial algorithm can achieve a better ratio. The direct consequence of this theorem is the following corollary.

**Corollary 3.1.4** The Greedy algorithm in Figure 3.1 can achieve the best approximate ratio for the view selection/eviction problem.

**Proof** We have shown in Theorem 3.1.2 that the algorithm is \( K \)-approximate. From Theorem 3.1.3 we know \( K \) is the best ratio. Thus the corollary holds. □

### 3.1.3 The KnapSack Revisited

Now let's revisit the well known 2-approximate KnapSack solution in Figure 3.2. The solution using KnapSack's 2-approximate algorithm also sorts the views in terms of \( c_i/|v_i| \), but unlike the Greedy Algorithm in Figure 3.1, it solves the problem by selecting the views to materialize, not to evict. Still, we use \( c_i \) to denote \( p_i(e_i - r_i) \).

Like Greedy, KnapSack's 2-approximate algorithm also evicts all the views in \( \{v_1, v_2, ..., v_{\xi-1}\} \). Based on that, it then picks the better of \( \{v_1, v_2, ..., v_{\xi-1}\} \) and \( \{v_\xi\} \). Since we know the Greedy algorithm which evicts all the views in
1. Sort views in non-increasing order of $\frac{s_j}{|v|}$.
   Assume the resulting order is marked by $j, j = 1, 2, \ldots, N$
2. Greedily add views until we hit the split view $v_S$. ($\sum_{j=1}^{S-1} |v_j| > K)$
3. Pick the better of $\{v_1, v_2, \ldots, v_{S-1}\}$ and $\{v_S\}$.

**Figure 3.2: Knapsack's Polynomial 2-Approximation Algorithm**

$\{v_1, v_2, \ldots, v_S\}$, Knapsack's 2-approximate algorithm must be better than the Greedy. This means that Knapsack's 2-approximate algorithm not only has the best approximation ratio $K$, but also achieves at least half of the optimal benefit and must outperform Greedy. This result is in contrast with the data cube case, where the approximate algorithm for the optimal benefit exists but the minimum expected cost is inapproximable.

### 3.1.4 An Extended Greedy Algorithm

In Section 3.1.2 we have proved that the Greedy algorithm in Figure 3.1 is the best approximation algorithm in the sense that it achieves the best approximation ratio $K$. But that doesn't mean it performs well under any circumstances. In this section we present another approximate algorithm, called Extended Greedy, that might outperform the Greedy algorithm in Figure 3.1 in some cases.

We have known that the problem can be seen as evicting a subset of the views $V$ conforming to Equation 2.1. Let's call this subset $\tilde{S}$. The Extended Greedy algorithm iteratively picks the least cost-expensive view into $\tilde{S}$ until the size $|\tilde{S}|$ is no less than $K'$, where $K' = \sum_{v \in V} |v|$. Suppose the view picked in iteration $j$ is denoted as $v_j$ and the set selected by the end of iteration $j$ is $\tilde{S}_j$, the cost-expensiveness for $v_j$ is defined as follows:

**Definition** *(Cost-Expensiveness)* The cost-expensiveness for $v_j$ in iteration $j$ is $\frac{q_j}{\min\{|v_j|, K' - |\tilde{S}_{j-1}|\}}$, where $\tilde{S}_{j-1}$ is the current subset to evict when iteration $j-1$ ends. $\tilde{S}_0 = \emptyset$.

The Extended Greedy algorithm is defined formally in Figure 3.3.
1. \( j = 0; S_j = \emptyset \)
2. While \( |S_j| \leq K' \)
   \( j = j + 1 \);
   Find the view \( v_j \) with the smallest cost-expensiveness, where \( j \) is the index of current iteration. (Break ties arbitrarily)
   \( S_j = S_{j-1} \cup v_j \)
3. Materialize the views in \( V - S_j \).

Figure 3.3: The Extended Greedy Algorithm

In any iteration \( j \), the selected view \( v_j \) covers the space of size \( \min\{|v_j|, K' - |S_{j-1}|\} \) with cost \( c_i \), where \( |S_{j-1}| \) is the total size of the views in \( S_{j-1} \). Let the cost per unit space, or cost density incurred by these views be: \( \rho_1, \rho_2, \ldots, \rho_{K'} \). The following lemma holds:

**Lemma 3.1.5**

\[
\rho_j \leq \frac{E^*(C)}{K'-j+1}, j = 1, 2, \ldots, K'
\]

where \( E^*(C) \) is the optimal objective value—the minimum expected cost—for the IP view selection/eviction problem.

**Proof** Let the subset to evict selected by the optimal IP solution be \( S^* \). In any iteration \( j \) when the Extended Greedy algorithm in Figure 3.3 is picking a view to evict, there must be some views belonging to \( S^* \) that have not been picked yet, otherwise the extended greedy algorithm would have covered all the space \( K' \) and terminated. Denote these views set \( O_j \), and \( O_j = S^* - S_{j-1} \). Views in \( O_j \) can fill the left uncovered space at a cost \( \sum_{i \in O} c_i \), which is no greater than the optimum solution's total cost \( S^* \).

Suppose in iteration \( j \), the selected view \( v_j \) covers the space indexed from \( |S_{j-1}| + 1 \) to \( |S_j| = |S_{j-1}| + |v_j| \). The cost densities incurred by it are denoted as \( \rho_{|S_{j-1}|+1}, \rho_{|S_{j-1}|+1}, \ldots, \rho_{|S_j|} \). As they belong to one view, these values are the same. Without losing generality, suppose the order in which the views are chosen is the same as their original numbering, i.e., the first view chosen...
\[ \rho_{\mathcal{S}_{j-1}+1} = \rho_{\mathcal{S}_{j-1}+2} = \ldots = \rho_{\mathcal{S}_j} \]
\[ = \frac{c_j}{\min\{v_j, K' - |\mathcal{S}_{j-1}|\}} \]
\[ \leq \text{The smallest } \frac{c_i}{\min\{v_i, K' - |\mathcal{S}_{j-1}|\}}, \text{ where } v_i \in O_j \]
\[ \leq \frac{\sum_{v_i \in O_j} c_i}{\sum_{v_i \in O_j} |v_i|} \]
\[ \leq \frac{E^*(C)}{\sum_{v_i \in O_j} |v_i|} \]
\[ \leq \frac{E^*(C)}{K' - |\mathcal{S}_{j-1}|} \]
\[ \leq \frac{E^*(C)}{K' - |\mathcal{S}_{j-1}| - |v_j| + 1} \]

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

\[ \rho_{\mathcal{S}_{j-1}+1} \leq \frac{E^*(C)}{K' - |\mathcal{S}_{j-1}|} \]
\[ \rho_{\mathcal{S}_{j-1}+2} \leq \frac{E^*(C)}{K' - |\mathcal{S}_{j-1}| - 1} \]
\[ \ldots \]
\[ \rho_{\mathcal{S}_{j-1}+|v_j|} \leq \frac{E^*(C)}{K' - |\mathcal{S}_{j-1}| - |v_j| + 1} \]

Now let's start from the first view:

\[ \rho_1 \leq \frac{E^*(C)}{K'} \]
\[ \rho_2 \leq \frac{E^*(C)}{K' - 1} \]
\[ \ldots \]
\[ \rho_{K'} \leq E^*(C) \]

52
The lemma is proved.

The lemma leads naturally to the following theorem:

**Theorem 3.1.6** The Extended Greedy algorithm is $H_{K'}$-approximate with respect to the optimal IP solution, where $H_{K'}$ is the harmonic number of $K'$.

**Proof** Denote the cost achieved by the Extended Greedy algorithm $E_{EG}(C)$. Starting from Lemma 3.1.5 we obtain:

\[
\rho_j \leq \frac{E^*(C)}{K' - j + 1}, \, j = 1, 2, \ldots, K'
\]

\[
\sum_{j=1}^{K'} \rho_j \leq \sum_{j=1}^{K'} \frac{E^*(C)}{K' - j + 1}
\]

\[
E_{EG}(C) \leq \sum_{j=1}^{K'} \frac{E^*(C)}{K' - j + 1}
\]

\[
E_{EG}(C) \leq H_{K'}
\]

Note that the approximation ratio of the Extended Greedy $H_{K'}$ is with respect to the solution of the IP problem. It doesn’t hold with respect to the solution of the LP relaxation because Lemma 3.1.5 doesn’t hold if $E^*(C)$ is changed to $E^*_{LP}(C)$. However this doesn’t guarantee that Extended Greedy is worse than Greedy. It can outperform both the Greedy and the Knapsack’s 2-approximate algorithms in some cases. Here is an example of such an instance:

**Example** In this example the system has four views and their features are shown in Table 3.2.

Suppose the cache size is 45, the execution of the three algorithms are respectively shown in Table 3.3

### 3.1.5 A Mediated Greedy Algorithm

In the last section we have seen that though the Greedy and the 2-approximate knapsack algorithm can achieve the best approximation ratio, the Extended Greedy
| view | cost $c_i$ | size $|v_i|$ | $c_i/|v_i|$ |
|------|------------|----------|---------|
| $v_1$ | 20         | 40       | 0.5     |
| $v_2$ | 40         | 40       | 1       |
| $v_3$ | 20         | 10       | 2       |
| $v_4$ | 15         | 5        | 3       |

**Table 3.2:** An instance where Extended Greedy is the best

**Table 3.3:** The result of the three algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Evicted View</th>
<th>Resulting Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>KS 2-app</td>
<td>$v_1, v_3, v_4$</td>
<td>55</td>
</tr>
<tr>
<td>Greedy</td>
<td>$v_1, v_2$</td>
<td>60</td>
</tr>
<tr>
<td>Extended Greedy</td>
<td>$v_1, v_3$</td>
<td>40</td>
</tr>
</tbody>
</table>

can outperform them in some cases. Is there a way to find an algorithm that is superior to all three of them? To find out the answer we need to analyze the behavior of the three algorithms.

We first look at the Extended Greedy in Figure 3.3. Just like what we did in the Greedy algorithm and the 2-approximate knapsack algorithm, let’s sort the views in terms of the cost density $\frac{c_i}{|v_i|}$ in non-increasing order. Without losing generality, suppose the result views’ indexes are conforming to their original indexes, i.e.,

$$\frac{c_1}{|v_1|} \leq \frac{c_2}{|v_2|} \leq \cdots \leq \frac{c_N}{|v_N|}$$

The split view is denoted as $v_\xi$.

The following lemma holds

**Theorem 3.1.7** The Extended Greedy algorithm must evict the first $\xi - 1$ views and their order of being chosen conforms to the order of views on cost-expensiveness.

**Proof** We assume that $\xi > 1$ because when $\xi = 1$ the theorem is trivially true. Then we prove the theorem by strong mathematical induction on $j, 1 \leq j \leq \xi - 1$, where $j$ is the iteration index.

**The Basis Step** We show that in the first iteration, $v_1$, who has the smallest density, must be chosen to be evicted.
In the first iteration the selected subset is $S_0 = \emptyset$, thus the algorithm picks the view with smallest $\frac{c_i}{\min\{|v|, K'\}}$. Suppose the algorithm didn’t choose $v_1$ but chose some other view $v_i$. It means the algorithm chose some other view $v_i$ other than $v_1$ and

$$\frac{c_i}{\min\{|v_i|, K'\}} < \frac{c_1}{|v_1|}$$

There are two cases:

1. $|v_i| \leq K'$
2. $|v_i| > K'$

Case 1 means $\frac{c_i}{|v_i|} \leq \frac{c_1}{|v_1|}$. It contradicts with the fact that $v_1$ has the smallest density.

In case 2, the cost-expensiveness for $v_1$ is $c_i/K'$ and

$$\frac{c_i}{K'} > \frac{c_i}{|v_i|} \geq \frac{c_1}{\min\{|v_1|, K'\}}$$

where $\frac{c_1}{\min\{|v_1|, K'\}}$ is the cost-expensiveness of $v_1$. The last step holds because we have assumed that $v_1$ is not the split view so $|v_1| \leq K'$. Thus it contradicts with the hypothesis that the algorithm chose $v_1$ instead of $v_i$.

So the algorithm must choose $v_1$ in the first iteration.

The Induction Step Suppose in iteration $j, 1 \leq j \leq \zeta - 1$, the algorithm didn’t choose $v_j$ but chose some other view $v$. It means

$$\frac{c_i}{\min\{|v|, K' - |S_{j-1}|\}} < \frac{c_j}{\min\{|v_j|, K' - |S_{j-1}|\}} \quad (3.5)$$

Because $1 \leq j \leq \zeta - 1$, the size of view $v_j$ must be no greater than the left uncovered space, i.e., $|v_j| \leq |S_{j-1}|$. Thus Equation 3.5 can be simplified to

$$\frac{c_i}{\min\{|v|, K' - |S_{j-1}|\}} < \frac{c_j}{|v_j|}$$

Now there are two cases:

1. $|v_j| \leq K' - |S_{j-1}|$
2. $|v_i| > K' - |S_{j-1}|$

Case 1 means $\frac{c_i}{|v_i|} \leq \frac{c_j}{|v_j|}$. Then $v_i$ must have been chosen in previous iterations, which contradicts with the hypothesis.

In case 2, the cost-expensiveness for $v_i$ is $\frac{c_i}{K' - |S_{j-1}|}$ and the hypothesis says

$$\frac{c_i}{K' - |S_{j-1}|} > \frac{c_j}{|v_j|}$$

We have know that $\frac{c_i}{|v_i|}$ is the cost-expensiveness of $v_j$. Thus it contradicts with the hypothesis that the algorithm chose $v_i$ instead of $v_j$.

So the algorithm must choose $v_j$ in iteration $j, 1 \leq j \leq \xi - 1$.

Now let’s look back at the Greedy algorithm and knapsack’s 2-approximate algorithm. We found that both algorithms evict the views from $v_1$ to $v_{\xi-1}$, just like the Extended Greedy. The difference is how they deal with the views from $v_\xi$ to $v_N$. Thus we can mediate the three algorithms so that the new algorithm not only achieves the best approximate ratio, namely, $K$-approximate on the expected cost and 2-approximate on the benefit, but also performs no worse than the Extended Greedy. In the following algorithm we use the denotation $C(\Phi)$ as the cost incurred by views in the set $\Phi$, and $C(\Phi) = \sum_{\phi \in \Phi} c_i$.

The algorithm is based on the Extended Greedy but it first identifies those views that must be evicted and only iteratively picks views in $V_{\text{high}} \cup v_\xi$. $V_{\text{high}}$ and $V_{\text{low}}$ are defined in Figure 3.4. In addition, it keeps a bound of $\min\{C(V_{\text{high}}), c_\xi\}$ when choosing the views. If the cost incurred by the chosen views exceeds the bound, it stops and pick the better of $V_{\text{high}}$ or $v_\xi$ to evict. Thus the algorithm inherits all the approximation ratios.

The run time complexity is still $O(N^2)$ but we could expect the real running time be smaller than that of the Extended Greedy, because step 1 and 2 in the Mediated Greedy Algorithm can be computed using the partial quick sort because we don’t care about the ordering within $V_{\text{low}}$. If the number of views in $V_{\text{low}}$ is a large number, a lot of of operations would be saved.
1. Use sorting algorithms to identify the split view \( v_\circ \),
the set of views whose cost-expensiveness are smaller than \( v_\circ \), denoted as \( V_{\text{low}} \),
the set of those higher than \( v_\circ \), denoted as \( V_{\text{high}} \),
and the total size of \( V_{\text{low}} \), denoted as \( |V_{\text{low}}| \).

2. During the sorting process, get the cost of \( V_{\text{high}} \), denoted as \( C(V_{\text{high}}) \).

3. \( S = \emptyset \).

4. While ( \( |S| \leq K' - |V_{\text{low}}| \) and \( C(S) < \min\{C(V_{\text{high}}), c_\circ\} \) )
   In \( V_{\text{high}} \cup \{v_\circ\} \), find the view \( v_j \) with the smallest cost-expensiveness.
   (Break ties arbitrarily)
   \( S = S \cup \{v_j\} \)

3. If \( C(S) \geq \min\{C(V_{\text{high}}), c_\circ\} \)
   Materialize the better of \( V_{\text{high}} \) or \( v_\circ \).
   Else
   Materialize the views in \( V - S - V_{\text{low}} \).

---

### 3.2 Competitive Policies

In Section 2.2 we discussed the optimal "dynamic" policies. When both the view sizes and costs are arbitrary, in order to get the optimal average reward, the running time before reaching the optimal states must be non-polynomial. In Section 3.1 we discussed the approximate static selection algorithms. In this section we will discuss polynomial competitive policies. We will present a polynomial time policy that can achieve \( K \) times the fractional optimal average reward under IRM. Contrast our result with the \( K \)-competitive algorithms to the integral average reward in [35][14] and the so-called optimality argument in LCB-K [28] and WATCHMAN[31], it shows that precisely predicting the probability information does help in improving the competitive bound, but cannot never be better than \( K \) times of the optimal fractional solution.
3.2.1 The $K$-Competitive Policy

To obtain the optimal average reward, a policy must reach the optimal states within finite time $T$. The amortized running time for each action from time 1 to $T$ must be non-polynomial, since if it is polynomial, we could have solved the Knapsack problem in polynomial time. When the number of views is very high, it would take too long to make a decision. Thus we introduce competitive policies defined below:

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

The optimal average reward we've achieved in Chapter 2 -- $ac^{OPT}(\theta)$ -- is actually an integral one, i.e., no views are chosen to be evicted or materialized partially. It can be easily inferred that if we allow the MDP system to admit or evict part of a view in its actions, the optimal average reward the policies can achieve is no worse than $ac^{OPT}(\theta)$ which is equal to $B_{max}$. $B_{max}$ is the integral maximum benefit the static selection can obtain. Following the convention in Section 3.1, we use $ar^{OPT}_c(\theta)$ to denote the fractional optimal average reward.

Now we will present a $K$ competitive policy in terms of the average reward. 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 2.2, we only consider the situation of faults.

![Figure 3.5: The $K$-competitive Policy](image)

Whenever a fault occurs on request $Q_t$ at time $t$ and there's not enough room for it, while the request $Q_t$ doesn't fit into the cache, evict the view with the smallest $\frac{C_i}{|V_i|}$ from $S_{t-1} \cup \{Q_t\}$. 

58
Theorem 3.2.1 The policy in Figure 3.5, denoted as \( \pi \), is \( K \)-competitive to the optimal fractional solution of the static selection problem, i.e.,

\[
ac^\pi(\theta) \leq K \cdot E_{LP}^{OPT}(C) \text{ for all starting state } \theta
\]

Where \( K \) is the size of the cache, and \( E_{LP}^{OPT}(C) \) is the expected cost for the optimal fractional solution to the static view selection problem.

Proof Without losing generality, suppose the views' indexes satisfy the following sequence:

\[
\frac{c_1}{|v_1|} \leq \frac{c_2}{|v_2|} \leq \ldots \leq \frac{c_N}{|v_N|}
\]

The split view is denoted as \( v_S \).

Under policy \( \pi \), the states converge to the group \( G_S \) whose \( N \) states share the same cache content \( S = \{v_{S+1}, v_{S+2}, \ldots, v_N\} \) for any starting state within finite time. At the first time the system reaches one state in this group, the system would remain in the group because the views in the cache (\( S \)) will never be evicted again according to the algorithm definition. 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 2.2, we know the average cost \( ac^\pi(\theta) \) for policy \( \pi \) is

\[
ac^\pi(\theta) = P_s^* c_H(\theta) = B_S
\]

if the closed communicating class only contains one group \( G_S \).

So

\[
ac^\pi(\theta) = B_S = \sum_{i=S+1}^{N} \beta_i(e_i - r_i) \leq K \cdot E_{LP}^{OPT}(C)
\]

where \( E_{LP}^{OPT}(C) \) is the expected cost for the optimal fractional solution to the static view selection problem. \( \blacksquare \)
This theorem leads to the following corollary:

**Corollary 3.2.2** The policy $\pi$ in Figure 3.5 is $K$-competitive to the optimal MDP policy, i.e.,

$$ac^\pi(\theta) \leq K \cdot ac^{OPT}(\theta)$$

for all starting state $\theta$

where $ac^{OPT}(\theta)$ is the optimal average cost for any starting state $\theta$.

**Proof** In Chapter 2 we have known that the optimal average cost $ac^{OPT}(\theta)$ is equal to $C_{\text{min}}$, the optimal expected cost of the static selection. Thus the following steps hold:

$$ac^\pi(\theta) \leq K \cdot E_{LP}^{OPT}(C)$$

$$\leq K \cdot C_{\text{min}}$$

$$\leq K \cdot ac^{OPT}(\theta)$$

In fact, for any MDP policy, the lower bound of the competitive ratio with respect to the optimal fractional solution of the static selection problem is $K$. This fact is given by the following corollary:

**Corollary 3.2.3** Any policy $\pi$ cannot achieve better competitive ratios smaller than $K$ with respect to the optimal fractional solution of the static selection problem, i.e.,

$$ac^\pi(\theta) \leq K \cdot E_{LP}^{OPT}(C)$$

for any starting state $\theta$

where $\pi$ is an arbitrary policy and $\theta$ is any starting state for $\pi$.

**Proof**

$$ac^\pi(\theta) \geq ac^{OPT}(\theta)$$

$$= C_{\text{min}}$$

$$\geq K \cdot E_{LP}^{OPT}(C) \cdots \text{From Theorem 3.1.3}$$
3.2.2 Comment To Related Works

In WATCHMAN [31] Scheuermann described a similar algorithm. Their algorithm admits a view only if the admission can increase the density of all the views in cache when a fault occurs. He proved under a constraint model where the sizes of the cached retrieved views are relatively small when compared with the total cache size $K$ and it is always possible to utilize almost all space in the cache, the algorithm is optimal. This, however, does not imply the optimality for general cases since it wipes off the distinction of arbitrary sizes and thus eliminates the difficulty of the problem. This algorithm is in fact $K$-competitive just as what we have shown in this chapter.
Chapter 4

Conclusion and Future Work

As a conclusion of this thesis, we discussed the view selection/caching problem under IRM when the views are non-correlated to each other. We 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. We presented both optimal and competitive algorithms, and revealed the lower bound of the competitive ratio.

However some problems still remain in this area, so in this chapter we will discuss the problems remaining and the problems open for future study.

4.1 The Problem Remaining

The following problems are found within the topics discussed in the thesis:

4.1.1 Overtaking Optimal Policies

In this thesis we were focusing on the average reward/cost criteria in seeking an optimal MDP policy. This criteria is not very selective, since we may have multiple average optimal policies. In order to discriminate the policies further, researchers introduced overtaking optimal policies:

Definition Overtaking Optimal A policy $\pi^*$ is Overtaking Optimal if for each starting state in the system, the total reward/cost of $\pi^*$ is better than any other policy $\pi$ at any time.
Thus our task is to find and prove such optimal algorithms.

4.1.2 The Problem For the Naive Version Problem, Online Case

This thesis mainly discussed the problem under IRM, which assumes the fixed and static distribution is known. If we don't know any thing about the future, the case is called online. In that case we can only predict the future through the past. There is a fold of policies evicting the view with smallest utility value function $LocalityFactor_1 \times \frac{\text{com}}{\text{size}}$, where $LocalityFactor_1$ is a simulation of the probability distributions. GreedyDual-Size, LCB-K, LCVT all falls into this category. In the thesis we have shown that under IRM, any policy cannot have competitive ratio smaller than $K$ with respect to the optimal fractional policy (if we define one, otherwise we use the optimal fractional solution for static selection). So for the online case we may wonder if it is possible to have the performance if we could precisely predict the distribution at any time. The conjecture is that even if the distribution at time $t$ is known, there is still a bound of $K$ to the optimal fractional solution.

4.1.3 The Problem For The Dependence Case

The views in this thesis assumes no dependence relations among the views. We have shown by experiments that if we decompose the views and build the dependence tries among them, the query answering cost can be improved. However in the data cube case, it was shown that the view selection problem for general partial order is not approximable, i.e., the expected cost to answer a query doesn't have a meaningful approximation bound. We may be wondering if this applies to the trie case in our problem.

Meanwhile, the researches done for the view management problem in the data warehouse area are mainly static, i.e., they select the views in one lump sum without considering the work load change. Some work was done in managing the view dynamically, such as WATCHMAN and Dynamat, however the policies they use were not creative or didn't have strong theoretical results. Moreover, they didn't consider the dependence relationships among the views.

Therefore our task is to design some real dynamic systems in managing the views for both the IRM and online case, taking the dependence tries into consider-
ation. There're certainly a lot of things to explore in this area.
Bibliography


   → pages iii, 9, 10, 20, 38, 41

   URL citeseer.ist.psu.edu/cohen99caching.html. → pages 57

   URL citeseer.ist.psu.edu/cohenolcompetitive.html. → pages 8


[34] N. Young. The k-server dual and loose competitiveness for paging. Algorithmica, 11:525–541, 1994. → pages 8


67