Università
degli Studi
di Padova

DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA

Facoltà di Ingegneria
Dipartimento di Ingegneria dell'Informazione

Scuola di Dottorato di Ricerca in Ingegneria dell'Informazione – XXII Ciclo
Indirizzo: *Scienza e Tecnologia dell'Informazione e della Comunicazione*

Applications of Control Theory to Computer Systems Optimization

Direttore della Scuola
Ch.mo Prof. Matteo Bertocco

Supervisore
Ch.mo Prof. Gianfranco Bilardi

Dottorando
Francesco Versaci

Abstract

Computer systems complexity is growing rapidly, thus making the efficient use of their resources an increasingly challenging task. In many cases the optimization of the management in such systems has been developed with ad-hoc techniques and heuristics.

In this thesis, a more general and flexible approach is explored to resource management, based on the powerful framework of stochastic optimal control. This approach requires a careful modeling of the system of interest as a dynamical system with appropriate cost functions and stochastic descriptions of the inputs that are imposed by the external environment. Then, the question of an optimal management policy that minimizes the expected cost becomes mathematically well posed and can be systematically investigated. Two cases studies illustrating the approach are developed, as summarized below.

In Chapters 1–5, the classical replacement problem for memory hierarchies is cast within the framework of optimal control theory. Memory references are assumed to comply with the Least Recently Used Stack Model (LRUSM); arbitrary stack-distance distributions are considered.

An optimal policy is derived to minimize the miss rate for an infinite trace (a control over an infinite horizon). We call it a K-L policy where $K(C)$ and $L(C)$ are parameters, whose value is a function of the buffer (cache) size C, determined by the stack-distance distribution. Then, the concept of Least Profit Rate (LPR) policy is introduced and it is shown that, for the LRUSM model, LPR is an optimal policy over an infinite horizon which, in steady state, coincides with the K-L policy. The LPR satisfies the inclusion property whereby the content of a given buffer is also contained in all larger buffers. This property is known to enable the efficient computation of the number of misses for a given address trace, simultaneously for all buffer sizes. Furthermore, the LPR formulation leads to a linear time computation of the values $K(C)$ and $L(C)$ for all relevant values of C, improving on the cubic bound that naturally arises within the K-L derivation. Furthermore, the properties of LPR are exploited to derive an efficient algorithm to optimally partition a buffer concurrently accessed by multiple processes. Finally, the miss rate of LPR is compared with that of OPT, the well-known optimal off-line policy, to investigate the difference between an exact and a statistical knowledge of the future of the trace.

Obtaining a closed form characterization of the optimal replacement policy over a finite horizon has proved to be rather more difficult than over the infinite horizon. The problem has been solved for monotone stack-distance distributions. Separate arguments establish the optimality of the Least Re-

cently Used (LRU) policy for all nonincreasing distributions and of the Most Recently Used (MRU) policy for all nondecreasing distributions. Interestingly, LRU and MRU are special cases of LPR, within the LRUSM model, for nonincreasing and non decreasing distributions, respectively. The results have been obtained by introducing a significant variant of the standard Bellman's equation, potentially useful for other control problems.

In Chapters 6–7 it is studied the problem of processors allocation for the Galois System, a tool for automatically parallelizing, by means of speculative execution, algorithms that present *data amorphous parallelism*. The Galois System is modeled using graph-theoretic concepts and the optimization goal is identified in trying to maximize the parallelism, while keeping a constant conflict ratio. This is linked to a function for which we analytically derive some properties that are then used to design an algorithm that controls the number of processors in a quick and stable way. For this purpose an extension to the well known Turán's theorem is developed.

Sommario

La complessità dei sistemi informatici sta crescendo rapidamente, rendendo l'uso efficiente delle loro risorse un compito sempre piú proibitivo. In molti casi la gestione ottimizzata di questi sistemi è stata sviluppata con tecniche ad-hoc ed euristiche.

In questa tesi viene esplorato un approccio piú generale e flessibile alla gestione delle risorse, fondato sul potente quadro teorico del controllo ottimo stocastico. Questo approccio richiede un'attenta modellizzazione del sistema di interesse come sistema dinamico, con appropriate funzioni di costo e descrizioni stocastiche degli ingressi imposti dall'ambiente esterno. A questo punto la ricerca di una politica di gestione ottima che minimizzi il costo aspettato è matematicamente ben posta e può essere risolta sistematicamente. Vengono sviluppati due casi di studio, come riassunto di seguito.

Nei Capitoli 1–5 il classico problema di sostituzione (*replacement*) per le gerarchie di memoria viene formulato nel quadro della teoria del controllo ottimo. Si assume che i riferimenti di memoria rispettino il Least Recently Used Stack Model (LRUSM) e vengono considerate distribuzioni arbitrarie sullo stack.

Una politica ottima per minimizzare il tasso di miss (accessi fuori dal buffer) per una traccia di lunghezza infinita viene derivata e chiamata K-L, dove $K(C)$ ed $L(C)$ sono parametri il cui valore è un funzione, determinata dalla distribuzione di accessi allo stack, della taglia C del buffer. In seguito viene introdotto il concetto di politica a tasso di profitto minimo (Least Profit Rate – LPR) e si dimostra che, nell'LRUSM, LPR è una politica ottima su orizzonte infinito che, allo stato stazionario, coincide con la politica K-L. LPR soddisfa la proprietà di inclusione: i contenuti di buffer di dimensione minore sono inclusi in quelli maggiori. Questa proprietà consente di calcolare efficientemente il numero di miss per una data traccia in contemporanea per tutte le taglie del buffer. Inoltre le proprietà della LPR vengono sfruttate per derivare un efficiente algoritmo di partizionamento per buffer condivisi concorrentemente fra piú processi. Infine il tasso di miss di LPR viene confrontato con quello di OPT, la nota politica ottima off-line, per indagare la differenza fra una conoscenza esatta e una statistica del futuro della traccia.

Ottenere una forma chiusa per la politica di sostituzione su orizzonte finito si è dimostrato un problema ben piú difficile che su orizzonte infinito, ed è stato risolto nel caso di distribuzioni di accesso monotone. Argomenti diversi dimostrano rispettivamente l'ottimalità della politica Least Recently Used (LRU) per distribuzioni non crescenti e quella di Most Recently Used (MRU) per distribuzioni non decrescenti. I risultati sono stati ottenuti grazie all'introduzione di una significativa variante dell'usuale equazione di Bellman,

potenzialmente utile in altri problemi di controllo.

Nei Capitolo 6–7 viene studiato il problema dell'allocazione dei processori nel sistema Galois, uno strumento per la parallelizzazione automatica, per mezzo di esecuzione ottimistica (*speculative execution*), di algoritmi che presentino un cosiddetto parallelismo amorfo sui dati (*data amorphous parallelism*). Il sistema Galois viene modellizzato tramite concetti di teoria dei grafi e l'obiettivo dell'ottimizzazione è identificato nella massimizzazione del parallelismo col vincolo di mantenere basso il tasso di conflitti. Questo viene collegato ad una funzione, per cui vengono analiticamente derivate alcune proprietà che sono poi usate nella progettazione di un algoritmo capace di controllare il numero di processori in maniera stabile e veloce. A tal fine viene sviluppata un'estensione del noto teorema di Turán.

Acknowledgments

First of all I want to thank my advisor Gianfranco Bilardi for the challenging and interesting topics he suggested for investigation during my PhD. The border between computer science theory and metaphysics can be subtle, and I like his way of doing theoretical research with deep roots in applications.

I wish to express my gratitude to Prof. Augusto Ferrante who has kindly provided valuable expert advise on optimal control theory at many critical junctures of this research.

The topics covered in Chapters 6 and 7 have been developed in the Fall semester of 2008, which I spent as a visiting research assistant at the University of Texas at Austin. I am indebted to Prof. Keshav Pingali and his group for the opportunity to participate to the Galois project, so rich of stimulating research questions.

Contents

CHAPTER 1

Introduction to Replacement Policies for the Memory Hierarchy

Historically, the storage of most computer systems is organized as a hierarchy of levels, for technological and economical reasons [23]. Furthermore, fundamental physical constraints on information density and signal speed imply that the memory of large systems is inherently hierarchical [10]. The hierarchy of typical state-of-the-art servers includes the register file, three levels of cache memory, main memory, and magnetic disks. A further level, implemented as a solid-state disk, will soon become common between memory and magnetic disk. Memory hierarchies have been investigated extensively from several perspectives, e.g., hardware organization [16, 36, 23], operating systems [40], compiler optimization [2, 46], models of computation [39], and algorithm design [1].

A central aspect in the management of any memory hierarchy is the decision of which data to keep in which level. It is customary to focus on a hierarchy with two levels (see Fig. 1.1), respectively called here the *buffer* and the *backing storage*, as in [32], the extension to multiple levels being generally straightforward. We assume that a sequential computing engine generates a sequence of access requests $a_1, a_2 \ldots, a_t, \ldots$ for *items* stored in the hierarchy; throughout, we assume that items are equally sized blocks of data. For example, the buffer could model a cache memory and the backing storage could model the main memory, the items being cache lines. In another example, the buffer could model the main memory and the backing storage the disk, the items being virtual pages. A request for an item that is in the

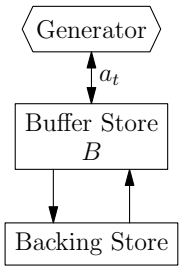


Figure 1.1: Two-level memory hierarchy.

buffer is called a *hit* and requires no data movement within the hierarchy. A request for an item that is not in the buffer is called a *miss*; to satisfy it, the requested item must be brought in the buffer, typically at a non negligible cost. If the buffer is full, then some item must be evicted from it to make room for the requested item. One is interested in criteria to select the items to be replaced so as to minimize subsequent misses. A *replacement policy* is a set of rules that define which item has to be evicted from the buffer upon a miss.

Belady [7] first investigated the idea of an optimal replacement policy, that minimizes the number of misses, for any given access sequence, buffer capacity, and initial buffer content. He proposed the *MIN policy* and claimed its optimality. Subsequently, Mattson, Gecsei, Sluts, and Traiger [32], defined and established the optimality of what they called *OPT policy*: Upon a miss, if the buffer contains items that will not be accessed in the future then evict (any) one of them, else evict the (unique) item whose next access is farthest in the future. MIN and OPT are closely related and generally regarded as the same policy (although they do differ in subtle ways). The main drawback of MIN and OPT is that, in practical situations, the sequence of items to be accessed is not known in advance, but rather unfolds with the computation, thus demanding *on-line* replacement decisions. Indeed, dozens of on-line replacement policies have been proposed in the literature and several have been implemented in various systems. To assess the quality of a given replacement policy, most studies resort to a methodology from one of the following classes.

Experimental. The number of misses is obtained for "benchmark" address traces, either by simulation or by measurement with hardware performance counters. The results can be compared among different on-line policies and/or against the performance of OPT, which provides a lower bound to the number of misses [36].

Probabilistic. The expected number of misses is derived or estimated

analytically for some policies, for specific stochastic distributions of accesses [17, 15].

Competitive ratio. Worst-case upper bounds are derived for the ratio between the number of misses of the target policy and that of OPT. When the buffer capacity C is the same for both policies, the competitive ratio of any on-line policy is at least C. More interestingly, when the on-line policy is given a larger buffer than OPT by some factor $\alpha > 1$, then the competitive ratio of some significant policies, such as Least Recently Used (LRU) and First In First Out (FIFO), can be bounded just in terms of α, as shown by Sleator and Tarjan [41].

It is natural to ask whether there is a universally "best" on-line policy, performing at least as well of any other policy on any input trace. The answer is negative since, given any two different replacement policies A and B, one can easily construct a trace for which A incurs fewer misses than B. However, if we view the trace as a stochastic process, then we can meaningfully explore the existence of optimal on-line policies, defined as those that minimize the expected number of misses. In general, the class of optimal policies will depend upon the statistical properties of the trace.

In the next chapters, we show that, under suitable statistical assumptions on the trace, it is possible as well as fruitful to cast miss minimization as a problem of optimal control theory. To this end, we briefly review the optimal control framework, slightly adapting the terminology and notation of Bertsekas [9]. We are given a dynamical system described by a state-transition equation of the form

$$x_{t+1} = f(x_t, u_t, w_t), \tag{1.1}$$

where x_t is the *state* at time t, while both u_t and w_t are inputs, with crucially different roles. Input u_t, called the *control*, can be chosen by whoever operates the system. In contrast input w_t, historically called the *disturbance* (although for us it will be the workload we are interested in processing), is determined by the environment and modeled as a stochastic process. At each step t, a cost is incurred, given by some function $g(x_t, u_t, w_t)$. Informally, the objective of optimal control is to find a *control policy* to determine the control as a function $u_t = \mu_t(x_t)$ of the state and so as to minimize the total cost

$$\mathbb{E}_w \left[\sum_{t \in I} g(x_t, u_t, w_t) \right], \tag{1.2}$$

where I is a time interval of interest. A key premise of most optimal control theory is the assumption of *past-independent disturbances* (PID) stating that,

given the current state x_t, the current disturbance w_t is statistically independent of past disturbances $\{w_\tau : \tau < t\}$. The PID assumption leads to a class of equations for the optimal control policy, such as the Bellman equation or the Hamilton-Jacobi-Bellman equation. The assumption also enables the exploitation of dynamic programming techniques, for the computation of the optimal control policy.

In order to formulate miss minimization as an optimal control problem we need to identify a suitable dynamical system and its cost function. Intuitively, we recognize that the disturbance has to model the address trace, since the latter is dictated to the memory manager by the environment (specifically, by the processes under execution). The control has to encode the replacement decisions, which are under the discretion of the memory manager. The cost per step can naturally be taken to be one if a miss is incurred and zero otherwise. A natural choice for the state would be the content of the buffer. However, the PID assumption essentially requires that the address trace $a_1, a_2, \ldots, a_t, \ldots$ be a sequence of mutually independent random variables, a scenario known as the Independent Reference Model (IRM) [17]. While often considered in the literature for its mathematical simplicity, the IRM does not capture the cornerstone property that memory hierarchies relay upon: the temporal locality of references. Intuitively, temporal locality means that recently referenced items are more likely to be accessed than other items, clearly implying dependence in the address trace. The outlined obstacle can be circumvented if the trace can be generated by a suitable dynamical system driven by a sequence of independent random variables. Then, such variables can be taken as the disturbance of a system whose state includes both the buffer content and the state of the trace generator. We shall show how this avenue is actually viable for the LRU-Stack Model (LRUSM) of the trace [35, 42], which has been widely considered in the literature, due to its ability to capture temporal locality. A different generalization of the IRM model is the Markov Reference Model (MRM), already suggested by [32], where the address trace $a_1, a_2, \ldots, a_t, \ldots$ is a finite Markov chain. A wealth of results are obtained by Karlin, Phillips, and Raghavan [26] for MRM, including the Commute Algorithm, a remarkable policy computable from the transition probabilities of the chain in polynomial time, whose expected miss rate is within a constant factor of optimum. We underscore that MRM and LRUSM are substantially different models; except in very specialized cases, the LRUSM trace is not directly a Markov process but rather a function of a Markov process whose states are exponentially many in the number of accessed items.

In Chapter 2, we develop the optimal control formulation of the replacement problem for the LRU-Stack Model, under which the address trace is

statistically characterized by the probability $s(j)$ of accessing the j-th most recently referenced item. We modify the standard assumption that the control is a function only of the state and allow it to depend on the disturbance as well: $u_t = \mu_t(x_t, w_t)$. This modification is necessary since eviction decisions are actually taken with full knowledge of the current access. The Bellman equation has to be modified accordingly. Throughout, we assume that the items being referenced belong to a finite universe of size V (the virtual space).

In Chapter 3, we investigate optimal policies over an infinite time interval. Two notions of optimality known in the control theory literature are considered: *bias* optimality and *gain* optimality. Bias optimality is a stronger property, but also rather difficult to deal with. We derive bias-optimal policy for a buffer of capacity $C = 2$, to illustrate the approach and the difficulty of the problem. We then derive gain-optimal policies, which minimize the miss rate, for arbitrary buffer capacity. These policies turn out to specified by two parameters, denoted as K and L, hence called K-L policies. They include, as special cases, LRU ($K = C - 1$, $L = C$) and Most Recently Used (MRU) ($K = 1$, $L = V$). For a given stack-distribution $s(\cdot)$, K and L are functions of C. We develop an algorithm that computes K and L for every C in time $O(V)$, by linking the problem to that of planar convex hull and adapting Graham's scan. We also show that, for any distribution $s(\cdot)$, the gain-optimal policy satisfies the highly desirable inclusion property [32], that is, the content of a buffer is always included in any larger buffer. We conclude this chapter by showing how to optimally partition a buffer between several processes executing concurrently.

In Chapter 4, we show that the ratio χ between the expected miss rate of the optimal on-line policy and that of OPT is at most $2\log(2V/(V-C))$. As an interesting corollary, when the buffer capacity C is a fixed fraction of the size of the virtual space V, we have $\chi = O(1)$. This indicates that the worst-case competitive ratio, approximately C, is a rather pessimistic metric in this case, as previously observed experimentally [7, 49].

In Chapter 5, we focus on optimization over a finite interval, or horizon in the terminology of control theory. Technically, the optimal control problem is considerably harder. As an indication, even if the system dynamics, its cost function, and the statistics of the disturbance are all time invariant, the optimal control policy is in general time-dependent. In this context, we establish two interesting results. For any monotonically non increasing stack distribution $s(\cdot)$ which is LRU is an optimal policy for any finite horizon, whereas MRU is optimal if the stack distribution is non decreasing. Despite the symmetry of the results the concepts used in the proofs are quite different. The standard approach based on (some variant of) the Bellman-Hamilton-

Jacobi equation, which requires "guessing" the optimal cost as a function of the initial state, does not seem applicable, as we have not succeeded in finding a closed form for such function. We have circumvented this obstacle by establishing an inductive invariant on the relative values of the cost for select pairs of states. This approach may have applicability to other optimal-control problems.

CHAPTER 2

Optimal Control Formulation of Replacement

In this chapter, we define a dynamical system whose optimal control corresponds to the minimization of the number of misses when the reference trace is a stochastic process defined by the LRU Stack Model, reviewed in the following section.

2.1 LRU Stack Model

Mattson et al. [32] made the far reaching observation that a number of replacement policies of interest, including LRU, MRU, and OPT, satisfy the following property.

Definition 2.1.1. *Given a replacement policy μ defined for all buffer capacities, let $B_t^\mu(C)$ be the content of the buffer of capacity C at time t, after processing references $a_1, \ldots, a_{t-1}$.*

We say that the inclusion property *holds at time t if, for any $C > 1$, $B_t^\mu(C-1) \subseteq B_t^\mu(C)$, with equality holding whenever the bigger buffer is not full ($|B_t^\mu(C)| < C$).*

We say that μ is a stack policy *if it satisfies the inclusion property at all times for all address traces, assuming that inclusion holds for the initial buffers $B_1^\mu(C)$, with $1 \leq C \leq V$.*

The optimal on-line policies developed in the next chapter are stack policies.

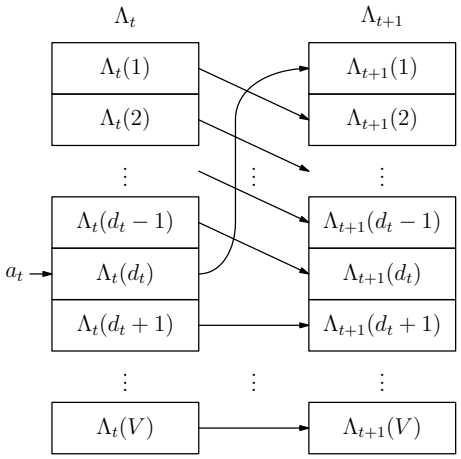


Figure 2.1: LRU stack update.

For a stack policy, the content of the buffers of all capacities can be compactly represented by an array Λ_t whose first C components yield the buffer of size C as

$$B_t^{\mu}(C) = [\Lambda_t(1), \ldots, \Lambda_t(C)]\,. \tag{2.1}$$

Array Λ is referred to as the *stack of the policy*. The *stack depth* of an access a_t is defined as its position in the policy stack at time t, so that

$$a_t = \Lambda_t(d_t). \tag{2.2}$$

A key observation [32] is that, upon an access of depth d, a buffer incurs a miss if and only if $C < d$. Thus, computing the stack depth is an efficient way to simultaneously track the performance of all buffer sizes.

Of particular interest here is the LRU stack, where the items are ordered according to the time of their most recent access; in particular, $\Lambda_{t+1}(1) = a_t$. Upon an access at depth d_t, the LRU stack is updated by a downward, unit cyclic shift of its prefix of length d_t, as illustrated in Fig. 2.1. The LRU stack has inspired an attractive stochastic model for the address trace [35, 42], where the address depths $d_1, d_2, \ldots$ are *independent and identically distributed* random variables, specified by the distribution

$$s_t(j) \stackrel{\Delta}{=} \mathrm{P}[a_t = \Lambda_{t-1}(j)], \quad j = 1, \ldots, V, \tag{2.3}$$

or equivalently by the cumulative sum

$$S(j) \stackrel{\Delta}{=} \sum_{i=1}^{j} s(i), \quad j = 1, \ldots, V. \tag{2.4}$$

For example, the case where $s(j)$ decreases with j captures a strict form of temporal locality, where the probability of accessing an item strictly decreases with the time elapsed from its most recent reference. It is a simple exercise to see that the actual trace $a_1, a_2, \ldots$ can be uniquely recovered from the stack-depth sequence $d_1, d_2, \ldots$, given the initial stack Λ_1.

2.2 The Control Problem

We are now ready to cast the replacement problem in the framework of optimal control theory that we have outlined in the introduction. Informally, the state of our dynamical system will model both the LRU stack (summarizing the relevant history of the input trace) and the buffer content (summarizing the relevant history of the replacement policy). Some reflection will convince us that the identity of the items is immaterial and that their only relevant property is whether or not they currently reside in the buffer. Therefore, we can condense the state information in a Boolean vector $x_t \in \{0,1\}^V$ such that

$$x_t(j) \triangleq \begin{cases} 1 & \Lambda_t(j) \text{ is in the buffer,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

With the above choice of state, we take the "disturbance" input of the system to be the sequence of stack depths, i.e., we let $w_t = d_t \in \{1, \ldots, V\}$. We let the control input $u_t \in \{0, 1, \ldots, V\}$ encode the eviction decisions with 0 denoting no eviction (the only admissible control in case of a hit) and $j > 0$ denoting the eviction of the item that is in position j of the LRU stack (which is an admissible control only when a miss occurs and the j-th item of the LRU stack is in the buffer, i.e., $x_t(j) = 1$).

We let f be the state transition function of the vector x under the LRUSM and g the instant cost function:

$$\begin{aligned} x_{t+1} &= f\left(x_t, u_t, w_t\right); \\ g(x_t, w_t) &\triangleq \begin{cases} 1 & \text{if a miss occurred,} \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{2.6}$$

Finally, we assume that a policy can set the control u_t with knowledge of both the state and the disturbance, that is, the item being accessed: $u_t = \mu_t(x_t, w_t)$. Many results in the control theory literature are stated under the assumption that the value of the disturbance is not available to the policy; however, analogous results usually hold for our model, with similar proofs.

Consider a policy $\pi = (\mu_1, \mu_2, \ldots, \mu_\tau)$ applied to our system during the time interval $[1, \tau]$, so that, for t in this interval, we have

$$x_{t+1} = f\left(x_t, \mu_t\left(x_t, w_t\right), w_t\right). \tag{2.7}$$

We define the cost of π, starting from state x_1, with time horizon τ as

$$J_\tau^\pi\left(x_1\right) \triangleq \mathbb{E}_w\left[\sum_{t=1}^{\tau} g\left(x_t, w_t\right)\right], \tag{2.8}$$

where the x_t's are subject to Eq. 2.7 and the expected value averages over disturbances. For our system, this is the expected number of misses in τ steps. The optimal cost is

$$J_\tau^*(x_1) \triangleq \min_\pi J_\tau^\pi\left(x_1\right). \tag{2.9}$$

The optimal cost satisfies the following dynamic-programming recurrence (analogous to Eq. 1.6 in Vol. 1 of [9])

$$J_\tau^*(x_1) = \mathbb{E}_w\left[\min_{u \in U(x_1, w)} \left\{g(x_1, w) + J_{\tau-1}^*\left(f\left(x_1, u, w\right)\right)\right\}\right], \tag{2.10}$$

where $U(x_1, w)$ denotes the set of controls admissible when the state is x_1 and the disturbance is w. It is often convenient to work with a vector $\vec{J}^*$ whose components are the values $J_\tau^*(x)$, in some conventionally chosen order of the states, and to rewrite Eq. 2.10 in compact form as

$$\vec{J}_\tau^* = \mathrm{T}\, \vec{J}_{\tau-1}^*, \tag{2.11}$$

with T being the *optimal-cost update operator*.

Often, the temporal horizon of interest in applications is long and not known a priori. Thus, it becomes attractive to consider policies that are optimal over an infinite horizon; an added benefit is that such policies are provably *stationary*, under very mild conditions. Usually, the cost defined in Eq. 2.8 diverges as $\tau \to \infty$, thus alternate definitions of optimality are considered in the literature [9, 29, 5, 12]. We base our study of the replacement problem on gain and bias optimality.

Gain Optimality refers to a policy μ^* that achieves the lowest *average cost*, i.e. $\mu^* = \arg\min_\mu \lim_{\tau\to\infty} J_\tau^\mu(x)/\tau$, which can be shown to be independent of x.

Bias Optimality refers to a policy μ^* that is as good as any other policies for τ long enough, i.e., $\forall x \ \lim_{\tau\to\infty} J_\tau^\mu(x) - J_\tau^{\mu^*}(x) \geq 0$.

We focus on infinite horizon problems in Chapters 3 and 4, and on finite horizon in Chap. 5.

CHAPTER 3

Optimal Replacement for the Infinite Horizon

We begin by deriving a bias-optimal policy for a buffer capacity $C = 2$. In spite of the apparent simplicity of the problem, the solution is non trivial and illustrates the difficulty of the general case. Further evidence of this difficulty is that, for arbitrary buffer capacity, bias-optimal policies do not satisfy the inclusion property. We then turn our attention to gain optimality, deriving optimal policies, with the inclusion property, for arbitrary stack-depth distributions. As an interesting application, we discuss how to optimally partition a buffer time shared by different processes.

3.1 The Bellman Equation

The Bellman equation is a classical result that characterizes bias-optimal control policies for a dynamical system in infinite horizon as solutions of a fixed point equation (see [9]). We show below that the equation holds in our model as well:

Theorem 3.1.1. *Let Δ be a dynamical system whose control u_t can be chosen with knowledge of the disturbance w_t. If*

$$\exists\lambda\,\exists\vec{h}: \quad \lambda\vec{1} + \vec{h} = \mathrm{T}\,\vec{h}, \tag{3.1}$$

then λ is the optimal average cost *of Δ and $\vec{h}$ is the vector of the* differential

costs *of the states, i.e.*

$$\forall x, y \in X \quad \lim_{\tau \to +\infty} J_\tau^*(x) - J_\tau^*(y) = h(x) - h(y). \tag{3.2}$$

To prove this theorem we need more notations and some lemmas:

Definition 3.1.2. *A system is said to be of* type 1 *(the standard model) if we are allowed to choose its control u at time t only as a function of the state at the same time:*

$$u_t = \mu(x_t) \in U(x_t) \tag{3.3}$$

where

$$U: \quad X \to \mathscr{P}(U) \tag{3.4}$$

and

$$\mu \in \mathcal{P}: \quad X \to U \tag{3.5}$$

Definition 3.1.3. *A system is said to be of* type 2 *(our model) if we are allowed to choose its control u at time t as a function of the state and the disturbance at the same time:*

$$u_t = \mu(x_t, w_t) \in U(x_t, w_t) \tag{3.6}$$

where

$$U: \quad X \times W \to \mathscr{P}(U) \tag{3.7}$$

and

$$\mu \in \mathcal{P}: \quad X \times W \to U \tag{3.8}$$

Definition 3.1.4. *A system $\Delta = (X, W, U, g, f, U(\cdot,\cdot))$ of type 2 is said to be* equivalent *to a system $\Delta' = (X', W', U', g', f', U'(\cdot))$ of type 1 if and only if $X = X'$, $W = W'$, $g = g'$ and*

$$\begin{aligned} &\forall x \in X \quad \forall w \in W \\ &\exists u \in U(x,w) : f(x,u,w) = y \\ &\iff \quad \exists u' \in U'(x) : f'(x,u',w) = y \end{aligned} \tag{3.9}$$

Remark 3.1.5. *Given an initial state x_1 and a realization of w_t for both systems, and a sequence of controls u_t for system Δ is always possible to find u'_t such that the state trajectories x_t (and hence the costs) of the two equivalent systems are the same.*

Lemma 3.1.6. *$\forall \Delta$ of type 2 $\exists \Delta'$ of type 1 s.t. Δ and Δ' are equivalent.*

Proof. Let

$$\begin{aligned} U' &\triangleq \mathcal{P} \qquad \forall x \in X \quad U'(x) \triangleq \mathcal{P} \\ f'(x, u', w) &\triangleq f(x, u'(x, w), w) \end{aligned} \tag{3.10}$$

Then $\forall x \in X \quad \forall w \in W$

- Let $y = f(x, u, w)$. $\forall u \in U(x, w)$ if we set $u' = \mu : \mu(x, w) = u$ we have

$$\begin{aligned} f'(x, u', w) &= f(x, \mu(x, w), w) \\ &= f(x, u, w) = y \end{aligned} \tag{3.11}$$

- Let $y = f'(x, u', w)$. $\forall u' = \mu \in U'(x) = \mathcal{P}$ if we set $u = \mu(x)$ we have

$$\begin{aligned} f(x, u, w) &= f(x, \mu(x, w), w) \\ &= f(x, u'(x, w), w) = f'(x, u', w) = y \end{aligned} \tag{3.12}$$

□

Optimal cost update equations. Let w_t be a random variable with values in W, i.i.d. for different t's. Furthermore let w_t be stationary and independent of x_t and u_t. Let Δ be a system of type 2 and Δ' an equivalent system of type 1. Then the cost update equation for Δ can be written as:

$$\forall x \in X \quad J^*_\tau(x) = \mathbb{E}_w \left[\min_{u \in U(x,w)} \left\{ g(x, w) + J^*_{\tau-1}(f(x, u, w)) \right\} \right] \tag{3.13}$$

$$\vec{J}^*_\tau = \mathrm{T}\, \vec{J}^*_{\tau-1} \tag{3.14}$$

whereas the same equation for Δ' is:

$$\forall x \in X \quad J^*_\tau(x) = \min_{u' \in U'(x)} \left\{ \mathbb{E}_w \left[g(x, w) + J^*_{\tau-1}(f'(x, u', w)) \right] \right\} \tag{3.15}$$

$$\vec{J}^*_\tau = \mathrm{T}'\, \vec{J}^*_{\tau-1} \tag{3.16}$$

Remark 3.1.7. *Since equivalent systems can reproduce each other's state evolution, their optimal costs are the same, in particular*

$$\mathrm{T}\, \vec{J}^*_{\tau-1} = \mathrm{T}'\, \vec{J}^*_{\tau-1} \tag{3.17}$$

We recall the classical Bellman equation theorem:

Theorem 3.1.8 (Standard Bellman equation)**.** *Given a dynamical system Δ' of type 1, if $\exists\lambda$ and $\exists\vec{h}$ such that*

$$\lambda\vec{1} + \vec{h} = \mathrm{T}\,\vec{h} \tag{3.18}$$

then λ is the optimal average cost of Δ and $\vec{h}$ are the differential costs of the states, i.e.

$$\forall x, y \in X \quad \lim_{\tau\to+\infty} J^*_\tau(x) - J^*_\tau(y) = h(x) - h(y) \tag{3.19}$$

We are now ready to prove our version of the Bellman equation for a system Δ of type 2:

Proof of Thm. 3.1.1. Consider a system Δ' equivalent to Δ. Applying Thm. 3.1.8 we have that, if we can solve Bellman equation for Δ' then we have found its optimal average cost and differential costs vector. Since the two systems are equivalent this would mean that they are also the costs of Δ. Hence we have

$$\exists\lambda\,\exists\vec{h} : \lambda\vec{1} + \vec{h} = \mathrm{T}'\,\vec{h} \quad \Rightarrow\ \lambda \text{ and } \vec{h} \text{ costs for } \Delta \tag{3.20}$$

but since $\mathrm{T}'\,\vec{h} = \mathrm{T}\,\vec{h}$ we finally have

$$\exists\lambda\,\exists\vec{h} : \lambda\vec{1} + \vec{h} = \mathrm{T}\,\vec{h} \quad \Rightarrow\ \lambda \text{ and } \vec{h} \text{ costs for } \Delta \tag{3.21}$$

□

3.2 Buffer of Capacity $C = 2$

When $C = 2$, the state of our dynamical system can be identified by the unique index $j \in \{2, \ldots, V\}$ such that the buffer contains the items in positions 1 and j of the LRU stack. The Bellman equation becomes

$$\begin{aligned} h(j) = 1 &- s(j) + h(2) - \lambda \\ &+ S(j-1)\min\{0, h(j) - h(2)\} \\ &+ (1 - S(j))\min\{0, h(j+1) - h(2)\}\,. \end{aligned} \tag{3.22}$$

The $h(j)$'s are defined up to an additive constant, so we can set $h(2) = 0$ to simplify the equation:

$$\begin{aligned} h(j) = 1 &- s(j) - \lambda \\ &+ S(j-1)\min\{0, h(j)\} \\ &+ (1 - S(j))\min\{0, h(j+1)\}\,. \end{aligned} \tag{3.23}$$

The solutions will satisfy

$$h(j) = h(j) - h(2) = \lim_{\tau \to +\infty} J^*_\tau(j) - J^*_\tau(2). \tag{3.24}$$

We now "guess" the rather complex form of the solutions, in terms of the auxiliary functions

$$\begin{aligned}
\beta(j) &\triangleq \max_{l \geq 1} \bar{s}(j, j+l-1). \\
\Phi(j) &\triangleq \Big\{ l \geq 1 : \forall k \in \{0, \ldots, l-1\} \\
&\qquad \bar{s}(j+k, j+l-1) \geq \beta(2) \Big\} \cup \big\{0\big\} \\
\phi(j) &\triangleq \max \Phi(j) \\
S^+(j) &\triangleq \bar{s}(j, j+\phi(j)-1)\phi(j) \\
\rho(j) &\triangleq \begin{cases} \bar{s}(j, j+\phi(j)-1) - \beta(2) & \phi(j) \neq 0 \\ \beta(j) - \beta(2) & \phi(j) = 0 \end{cases}
\end{aligned} \tag{3.25}$$

where $\Phi(j)$ is the subsequence of items that are visited when applying the policy induced using β as a priority, $\phi(j)$ the length of this subsequence and $S^+(j)$ its area.

Theorem 3.2.1. *Bellman equation (3.23) is solved using the following λ and $h(j)$:*

$$\begin{aligned}
\lambda &= 1 - \beta(2) \\
h(j) &= \beta(2) - s(j) - \frac{S(j-1)}{1 - S(j-1)} \phi(j)\rho(j) \\
&\quad - \phi(j+1)\rho(j+1) \\
&= \frac{S(j-1)}{1 - S(j-1)} \left[\phi(j)\beta(2) - S^+(j)\right] \\
&\quad + \beta(2)\left[1 + \phi(j+1)\right] - \left[s(j) + S^+(j+1)\right]
\end{aligned} \tag{3.26}$$

Proof. Let ψ be

$$\psi(j) \triangleq \frac{1}{1 - S(j-1)} \tag{3.27}$$

then

$$h(j) = \begin{cases} -\psi(j)\rho(j)\phi(j) \\ \qquad\qquad \text{for } \rho(j) > 0 \, (\Rightarrow \phi(j) > 0) \\ \beta(2) - s(j) - \phi(j+1)\rho(j+1) \\ \qquad\qquad \text{for } \rho(j) < 0 \, (\Rightarrow \phi(j) = 0) \end{cases} \tag{3.28}$$

This implies

$$\min\{0, h(j)\} = -\psi(j)\rho(j)\phi(j) \tag{3.29}$$

Using this term we can see that the chosen λ and $h(j)$ satisfies Eq. (3.23). □

The solution to the Bellman equation is likely to be quite more complex for arbitrary buffer capacity C, since the number of states grows as $\binom{V-1}{C-1}$. We also show next that bias-optimal policies are not stack policies, hence they can not be compactly specified in terms of priorities.

Theorem 3.2.2. *The bias-optimal policy, both over finite and infinite horizon, is not necessarily a stack policy.*

Proof. We will exhibit a counterexample of a distribution s that induces a unique bias-optimal policy, not induced by a priority (and hence not a stack policy). More in detail we first obtain by dynamic programming (executed by a computer program) the finite horizon optimal policies for two different buffer sizes C_1 and C_2 (being $C_1 < C_2$). Starting with buffers that satisfy the inclusion ($B_0(C_1) \subseteq B_0(C_2)$) we show that, exists a temporal horizon τ and state positions j_1 and j_2 such that, when in a state with both positions filled, for $C = C_1$ the optimal policy evicts at j_1, whereas for $C = C_2$ it evicts at j_2.

We then give evidence that the result is valid for all large enough temporal horizons, implying that the bias-optimal policy in infinite horizon does not have the inclusion property.

Consider the following s distribution, with $V = 8$ and $\beta = \frac{1}{16}$.

$s(j)$:	β	3β	3β	0	4β	0	0	5β
$j \in [1, V]$	1	2	3	4	5	6	7	8

We are given an initial LRU stack Λ_0 and we consider the following initial buffers $B_0(2)$ and $B_0(3)$, satisfying the inclusion property $B_0(2) \subset B_0(3)$:

$$B_0(2) = [\Lambda_0(1), \Lambda_0(4)] \tag{3.30}$$

$$B_0(3) = [\Lambda_0(1), \Lambda_0(4), \Lambda_0(7)] \tag{3.31}$$

If an access arrives at $x_0 = \Lambda_0(8)$ a miss occurs in both buffers, and hence an eviction is needed. The possible policies for $C = 2$ are

- $\mu_2(1)$: Evict $\Lambda_0(1)$
- $\mu_2(4)$: Evict $\Lambda_0(4)$

whereas for $C = 3$ they are

- $\mu_3(1)$: Evict $\Lambda_0(1)$
- $\mu_3(4)$: Evict $\Lambda_0(4)$
- $\mu_3(7)$: Evict $\Lambda_0(7)$

($\Lambda_0(8)$ has just been accessed and cannot be evicted). We consider a time horizon of 5 steps and obtain, by the computer execution of the dynamic programming algorithm, the expected number of misses of these possible policies, summarized as follows:

$$M_2(1) = 2.893 \qquad M_2(4) = 3.000 \tag{3.32}$$

$$M_3(1) = 1.952 \qquad M_3(4) = 1.939 \qquad M_3(7) = 2.143 \tag{3.33}$$

Note that there are no identical costs, and hence the optimal policies are unique (no other policy achieves the same miss rate). In particular for $C = 2$ the optimal policy prefers to evict $\Lambda_0(1)$ rather than $\Lambda_0(4)$, whereas for $C = 2$ it evicts $\Lambda_0(4)$ rather than $\Lambda_0(1)$ or $\Lambda_0(7)$. The resultant buffers do not have anymore the inclusion property and therefore the optimal policy is not a stack one.

We have computed the costs of the different choices for the time horizons from 2 to 20 (see Fig. 3.1): from the data obtained we can see that the costs tend differences to an asymptote, giving evidence that even the bias-optimal policy in infinite horizon are not in general stack policies.

□

3.3 Arbitrary Buffer Capacity – K-L Policy

Fortunately, gain-optimality is more amenable to a systematic analysis. Gain optimal policies turn out to be stack policies, easy to describe and to implement.

Relaxing the capacity constraint We will actually obtain them in a more general contest, i.e. after relaxing the buffer capacity constraint to equal C on average, rather than at each step. Typically the buffer capacity is considered to be fixed and, when the buffer is full, an eviction is required every time a miss occurs, otherwise no evictions are performed. In the more general relaxed model we target an average buffer use and can do evictions (even many at once) when we want to, without the need for a miss to have occurred (similarly we can skip evictions when misses occur). This model

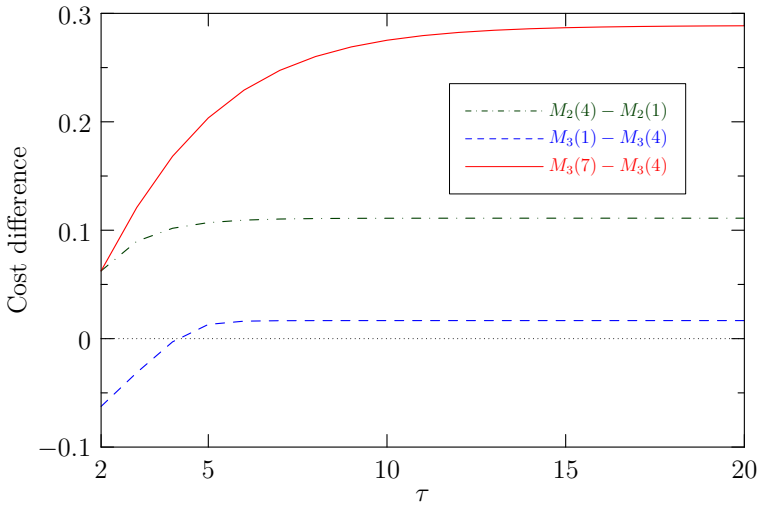


Figure 3.1: The difference in the expected misses for the possible policies at $t = 1$ as a function of the temporal horizon τ. The optimal policy limit (bias-optimal in infinite horizon) appears to be not a stack policy.

has applications when the buffer capacity is larger than the used one, e.g. when a buffer is shared by many processes and its portions are assigned by a scheduler or when we have to pay for the buffer use and we are interested in limiting the total cost spent by a process.

In what follows we will prove that the optimal policy for the usual *on-demand* model (eviction made only when misses occur) is optimal even in the broader contest obtained relaxing the fixed capacity constraint.

Outline of the Optimal Policies Given a buffer capacity C the first optimal policy that we obtain, called K-L policy, is characterized by two parameters K and L that represent positions in the LRU stack. A K-L starting with an empty buffer will evict in positions $L + 1$ (preferably) or (otherwise) at $K + 1$. The steady state will have the items in the top K positions always in the buffer, all the items at positions deeper than L outside the buffer and $C - K$ over the $L - K$ remaining items between K and L in the buffer. This policy generalizes both LRU (obtained for $K = C - 1$ and $L = C$) and MRU ($K = 1$ and $L = V$).

In the next section we define a new stack policy, called Least Profit Rate (LPR), based on the intuitive notion of profit (defined in Sec. 3.4). We will see that this policy, when we fix C, converges at steady state to the corresponding K-L policy, and is thus optimal. The LPR stack policy sheds new light on the functions $K(C)$ and $L(C)$, that describes how K and L vary

as a function of the buffer capacity. We will see that an access distribution s induces a segmentation σ_i of $\{1, \ldots, V\}$ and this segmentation describes the couples $K(C)$ and $L(C)$ for all possible C values as:

$$K_C = \sigma_i < C \leq \sigma_{i+1} = L_C \tag{3.34}$$

We will develop an algorithm that computes the complete segmentation in *linear* time, whereas a straightforward computation that does not exploit the stack policy characterization takes *cubic* time.

3.3.1 The Model

Let $p_t^{in}(j)$ be the probability for item $\Lambda_{t-1}(j)$ to be in the buffer B when a_t is referenced and before updating the LRU stack. The hit probability at time t can be written as

$$\mathrm{P}_t^{hit} = \sum_{j=1}^{V} p_t^{in}(j) s(j) \tag{3.35}$$

The hit rate from time 1 to time τ, using the stationarity of s, can be written as

$$\begin{aligned} \bar{\mathrm{P}}_\tau^{hit} &= \frac{1}{\tau} \sum_{t=1}^{\tau} \mathrm{P}_t^{hit} = \frac{1}{\tau} \sum_{t=1}^{\tau} \sum_{j=1}^{V} s(j) p_t^{in}(j) \\ &= \sum_{j=1}^{V} s(j) \left(\frac{1}{\tau} \sum_{t=1}^{\tau} p_t^{in}(j) \right) = \sum_{j=1}^{V} s(j) \bar{p}_\tau^{in}(j) \end{aligned} \tag{3.36}$$

where

$$\bar{p}_\tau^{in}(j) \triangleq \frac{1}{\tau} \sum_{t=1}^{\tau} p_t^{in}(j) \tag{3.37}$$

Remark 3.3.1. *An optimal policy for item eviction must maximize the hit rate* $\bar{\mathrm{P}}_\tau^{hit}$.

Let us define two events:

- $\mathrm{IN}_t(j)$: the item in position j in the LRU stack is in the buffer at time t
- $\mathrm{EV}_t(j)$: the item in position j in the LRU stack at time t has just been evicted

Think to the state update as divided in two parts: the stack rotation followed by the eviction. Let t^- be the time *after* the LRU stack update and *before* the eviction. Note that $\mathrm{EV}_t(j)$ is a subset of $\mathrm{IN}_{t^-}(j)$, so the following relation holds:

$$\begin{aligned} \mathrm{P}[\mathrm{IN}_t(j)] &= \mathrm{P}[\mathrm{IN}_{t^-}(j) \cap \overline{\mathrm{EV}}_t(j)] \\ &= \mathrm{P}[\mathrm{IN}_{t^-}(j)] - \mathrm{P}[\mathrm{EV}_t(j)], \end{aligned} \tag{3.38}$$

where

$$\begin{aligned} \mathrm{P}[\mathrm{IN}_{t^-}(j)] &= S(j-1)\,\mathrm{P}[\mathrm{IN}_{t-1}(j) | d_{t-1} < j] \\ &\quad + (1 - S(j-1))\,\mathrm{P}[\mathrm{IN}_{t-1}(j-1) | d_{t-1} \geq j] \\ &= S(j-1)\,\mathrm{P}[\mathrm{IN}_{t-1}(j)] \\ &\quad + (1 - S(j-1))\,\mathrm{P}[\mathrm{IN}_{t-1}(j-1)]. \end{aligned} \tag{3.39}$$

Finally, we obtain this fundamental relation

$$\begin{aligned} p_t^{in}(j) &= S(j-1) p_{t-1}^{in}(j) \\ &\quad + (1 - S(j-1))\, p_{t-1}^{in}(j-1) - p_t^{ev}(j). \end{aligned} \tag{3.40}$$

Theorem 3.3.2. *If $\forall j\ p_1^{in}(j) = 0$ (buffer initially empty) then $\forall \tau\ \bar{p}_\tau^{in}(j)$ is monotonic non increasing with j.*

Proof. To lighten the following formulas, let $S \triangleq S(j-1)$. If we sum the fundamental recurrence over t we obtain:

$$\begin{aligned} \sum_{t=1}^{\tau} p_t^{in}(j) &= p_1^{in}(j) + S \sum_{t=1}^{\tau-1} p_t^{in}(j) \\ &\quad + (1-S) \sum_{t=1}^{\tau-1} p_t^{in}(j-1) + \sum_{t=2}^{\tau} p_t^{ev}(j) \\ &= S \sum_{t=1}^{\tau} p_t^{in}(j) + p_1^{in}(j) - S p_\tau^{in}(j) \\ &\quad + \sum_{t=2}^{\tau} p_t^{ev}(j) + (1-S) \sum_{t=2}^{\tau} p_t^{in}(j-1) \\ &\quad + (1-S) p_1^{in}(j-1) - (1-S) p_\tau^{in}(j-1) \end{aligned} \tag{3.41}$$

$$\begin{aligned} \sum_{t=1}^{\tau} p_t^{in}(j) &= \sum_{t=2}^{\tau} p_t^{in}(j-1) + \frac{1}{1-S} p_1^{in}(j) \\ &\quad - \frac{S}{1-S} p_\tau^{in}(j) + p_1^{in}(j-1) \\ &\quad - p_\tau^{in}(j-1) - \frac{1}{1-S} \sum_{t=2}^{\tau} p_t^{ev}(j) \end{aligned} \tag{3.42}$$

If $p_1^{in}(j) = p_1^{in}(j-1) = 0$ we have

$$\sum_{t=1}^{\tau} p_t^{in}(j) = \sum_{t=1}^{\tau} p_t^{in}(j-1) - \eta \tag{3.43}$$

with $\eta \geq 0$, so $\bar{p}_\tau^{in}(j) \leq \bar{p}_\tau^{in}(j-1)$. □

Theorem 3.3.3. *When τ increases $\bar{p}_\tau^{in}(j)$ tends to a monotonic non increasing function of j, regardless of the initial p_1^{in} state.*

Proof. For large τ we have:

$$\bar{p}_\tau^{in}(j) \simeq \bar{p}_\tau^{in}(j-1) - \frac{1}{1 - S(j-1)} \bar{p}_\tau^{ev}(j) \tag{3.44}$$

□

Corollary 3.3.4. *If we never evict an item at position j in the LRU stack, then $\bar{p}_\tau^{in}(j) \simeq \bar{p}_\tau^{in}(j-1)$ for large values of τ.*

3.3.2 The Reduced Optimization Problem

To solve the infinite horizon problem we need to maximize the average hit rate under the constraints that the average buffer occupation is equal to C, for the non increasing $\bar{p}^{in} \triangleq \lim_{\tau \to \infty} \bar{p}_\tau^{in}$, i.e.

$$\text{maximize} \sum_{j=1}^{V} s(j) \bar{p}^{in}(j) \quad \text{s.t.} \quad \sum_{j=1}^{V} \bar{p}^{in}(j) = C \tag{3.45}$$

This problem is equivalent to the one addressed in [47, 48] for fixed memory buffer using the stationary p^{in} instead of $\bar{p}^{in}$.

Theorem 3.3.5. *For a given distribution s there is always an optimal $\bar{p}^{in}$ that maximizes the hit rate having the following form (see Fig. 3.5):*

$$j \mapsto \bar{p}^{in}(j) = \begin{cases} 1 & j \leq K \\ \frac{C-K}{L-K} & K < j \leq L \\ 0 & j > L \end{cases}$$

where $1 \leq K < C$ and $C \leq L \leq V$.

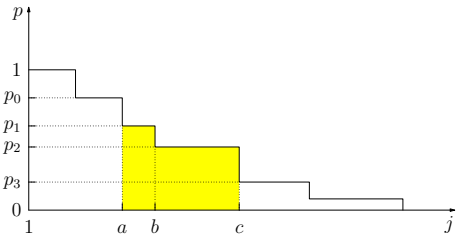


Figure 3.2: General p function shape.

Lemma 3.3.6. *Let s and p be functions from natural to real numbers*

$$s : \mathbb{N} \to \mathbb{R} \qquad\qquad p : \mathbb{N} \to \mathbb{R}^+$$

with p monotonic non-increasing such as

$$\sum_{j=1}^{\infty} p(j) = A$$

Let Ψ_p be

$$\Psi_p = \sum_{j=1}^{\infty} s(j)p(j)$$

Then Ψ_p is maximized for p defined as a step of length λ for some λ:

$$\arg\max_{p'} \Psi_{p'} = p(j) = \begin{cases} \frac{A}{\lambda} & j \leq \lambda \\ 0 & j > \lambda \end{cases} \tag{3.46}$$

Proof (reductio ad absurdum). If the optimal p had another shape then it would have several steps (p is monotonic) and not just one, as shown in Fig. 3.2. We consider now these extra steps and show that removing one of them yields to an improved or unchanged Ψ_p, hence proving that exists an optimal p function which does not have more than one step.

In maximizing the sum $\sum_{j=1}^{\infty} p(j)s(j)$, we focus now on the addenda (see Fig. 3.2)

$$\psi_p = \sum_{j=a}^{c-1} p(j)s(j) = S_1 p_1 + S_2 p_2$$

where

$$S_1 = \sum_{j=a}^{b-1} s(j) = \bar{s}_1(b-a)$$
$$S_2 = \sum_{j=b}^{c-1} s(j) = \bar{s}_2(c-b)$$

ψ_p can then be rewritten as:

$$\psi_p = p_1(b-a)\bar{s}_1 + p_2(c-b)\bar{s}_2$$

We now want to vary p_1 and p_2 values, keeping constant the area B below function p ($B \triangleq p_1(b-a) + p_2(c-b)$) without altering p monotonicity. It is therefore convenient to write ψ_p using a convex combination of 0 and B as

$$\begin{aligned}\psi_p(\alpha) &= \alpha B\bar{s}_1 + (1-\alpha)B\bar{s}_2 \\ &= \frac{\alpha B}{b-a}S_1 + \frac{(1-\alpha)B}{c-b}S_2 \\ &= p_1'(\alpha)S_1 + p_2'(\alpha)S_2\end{aligned}$$

with the monotonicity constraints

$$p_1'(\alpha) \in \left[\frac{B}{c-a}, p_0\right] \qquad\qquad p_2'(\alpha) \in \left[p_3, \frac{B}{c-a}\right]$$

ψ_p is monotonic in α:

$$\frac{\mathrm{d}\psi_p}{\mathrm{d}\alpha} = B(\bar{s}_1 - \bar{s}_2)$$

for $S_1 \neq S_2$. To see how is convenient to change p_1' and p_2' we need to analyze the following cases in detail:

- $\bar{s}_1 < \bar{s}_2$: ψ_p decreases with α, so it has its maximum for the lowest value of $p_1'(\alpha)$, i.e. $p_1' = p_2' = \frac{B}{c-a}$, thus erasing the step in b (see Fig. 3.3)

- $\bar{s}_1 > \bar{s}_2$: ψ_p increases with α and has its maximum for the highest p_1' and the lowest p_2'. If $(b-a)(p_0-p_1) > (c-b)(p_2-p_3)$ we reach the maximum for $p_1' = p_0$ (Fig. 3.3) erasing the step in a, otherwise we have the maximum for $p_2' = p_3$ (Fig. 3.4), erasing the step in c.

- $\bar{s}_1 = \bar{s}_2$: ψ_p is constant, so we can increase α and eliminate one of the steps in a or c

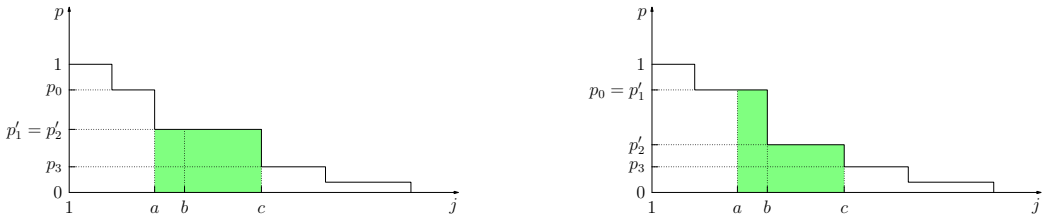


Figure 3.3: Removing an extra step to increase Ψ_p. In the first case (step in b) we have $\bar{s}_1 < \bar{s}_2$, in the second one (step in a) $\bar{s}_1 > \bar{s}_2$.

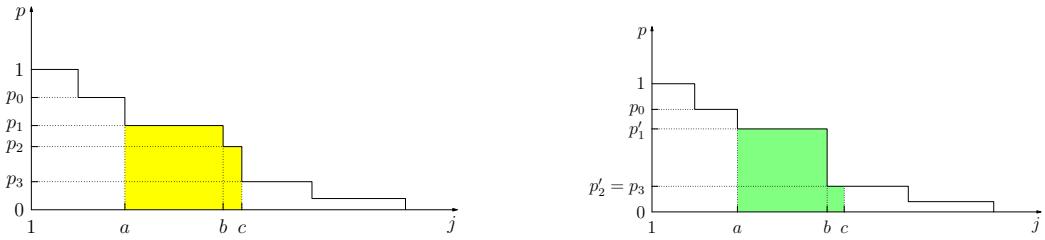


Figure 3.4: Step elimination in c when $(b-a)(p_0-p_1) > (c-b)(p_2-p_3)$.

In all these cases we obtain an improved or unchanged function removing a step, therefore iterating this process we obtain the one-step function (3.46). □

Definition 3.3.7. *Let $\bar{s}$ be the moving average of s:*

$$\bar{s}(j) \triangleq \sum_{i=1}^{j} \frac{s(i)}{j}$$

Definition 3.3.8. *Let $\bar{s}(i,j)$ be the average of s between i and j:*

$$\bar{s}(i,j) = \sum_{k=i}^{j} \frac{s(i)}{j-i+1}$$

Lemma 3.3.9. *Let s be a function from natural to real numbers $s : \mathbb{N} \to \mathbb{R}$ and let Ψ have the optimal form shown in Lemma 3.4*

$$\Psi(n) = \frac{A}{n} \sum_{i=1}^{n} s(i)$$

with $A \in \mathbb{R}^+$, then λ is a point of maximum for Ψ if and only if is a point of maximum for $\bar{s}$

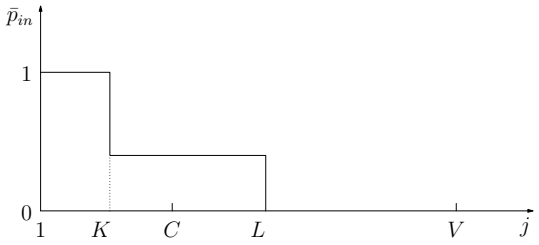


Figure 3.5: Optimal $\bar{p}^{in}$ function shape.

Proof.

$$\Psi(n) = A\sum_{i=1}^{n}\frac{s(i)}{n} = A\bar{s}(n)$$

□

Proof of Thm. 3.3.5. We want to maximize

$$\bar{\mathrm{P}}_T^{hit} = s(1) + \sum_{j=2}^{V} s(j)\bar{p}_T^{in}(j) = s(1) + \Psi_{p,T}$$

with

$$\sum_{j=2}^{V}\bar{p}_T^{in}(j) = C - 1$$

If we restrict s to values $j > 1$ we are in the hypotheses of Lemma 3.3.6, with the further hypothesis that we want $\bar{p}^{in}(j) \leq 1$. The best $\bar{p}^{in}$ function for $j > 1$ would be a step of height $(C-1)/(\lambda-1)$, with λ maximizing $\bar{s}(2,\lambda)$ (because of Lemma 3.3.9).

Since we have the additional hypothesis $\bar{p}^{in}(j) \leq 1$ we can accept λ only if $\lambda \geq C$, yielding $K = 1$ and $L = \lambda$.

If $\lambda < C$ we cannot have a single step of width $C-1$ for $j > 1$. Using considerations similar to the one developed in the proof of Lemma 3.3.6 we can remove most steps to obtain a better hit rate, but we may be unable to do it if $\bar{S}_1 > \bar{S}_2$ and $P_1 = 1$, because of the impossibility of further increasing P_1. In this case we would end up with a double step, the first of height one, ending in $K < C$ and a second one, lower, ending in L, for some $L > C$. □

Theorem 3.3.10. *Given an optimal $\bar{p}^{in}$ function with parameters K and L we can construct a replacement policy that generates $\bar{p}^{in}$, starting from an empty buffer, using the following eviction rules:*

- *evict $\Lambda(L+1)$ whenever in buffer;*

- *otherwise evict* $\Lambda(K+1)$ *(which is always in buffer after the rotation).*

Proof. We have the following regions:

1. $\bar{p}^{in}(j) = 1$ for $1 \leq j \leq K$ because of Corollary 3.3.4, provided that $\bar{p}^{in}(1) = 1$ and that we never evict for $j \leq K$

2. $\bar{p}^{in}(j) = \bar{p}^{in}(j-1)$ for $K+1 < j \leq L$, because of Corollary 3.3.4 provided that we never evict for $K+1 < j \leq L$

3. $\bar{p}^{in}(j) = 0$ for $j > L$ because

   - At the beginning the items after L were unbuffered (since the buffer is initially empty)
   - If there are no items in the buffer after position L, every time that a buffered item moves to position $L+1$ it is because a miss occurred, so we can evict $\Lambda(L+1)$ if buffered, still keeping positions after L unbuffered.

□

3.4 The Least Profit Rate Replacement policy

The above K-L policy definition present some problems:

1. It is completely defined only for states that have an item in positions K or L in the LRU stack

2. We do not have an efficient way of computing the parameters K and L as a function of C, since the straightforward algorithm for doing it is $O(V^2)$

Furthermore we do not know any strong properties of the K-L policy. E.g., it would be very useful if it could be regarded as a stack policy, as defined in [32]. We recall that a policy is a stack one if and only if it has the inclusion property for its buffers, as in Definition 2.1.1. The eviction choice in a stack policy can be seen as induced by a priority list (in general time-dependent) and hence a way of proving the inclusion property is by exhibiting an appropriate priority that can be used to choose which item is to be replaced. Stack policies (which include, e.g., OPT, LRU and LFU) can be described compactly by their priorities and have the useful property that their miss rate is computable at the same time for all buffer capacities.

Least Profit Rate Policy Below, we introduce the concepts of *profit* and of *profit rate* for an item z, at a given time t. We assume that the address trace is a stochastic process where the current reference x_t is a random function of an underlying markovian observable state. Consider now a rule ρ to determine, for any underlying state, whether z is kept in the buffer or evicted upon reaching that state. Let then $t+\Delta$, with $\Delta > 0$, be the earliest time after t such that z is either referenced or evicted at time $t+\Delta$. We call ρ-profit of z the probability π_ρ that z is referenced at time $t+\Delta$, which is a measure of how useful it would be to keep z in the buffer under the rule. Clearly, $E_\rho[\Delta]$ is a measure of the storage investment made on z to ripe that profit. Therefore, the quantity $\pi_\rho/E_\rho[\Delta]$ is a measure of profit per unit time, under rule ρ. Finally, we call *profit rate* of z, denoted $\xi(z)$, the maximum profit rate achievable for z, as a function of ρ. (It ought to be clear that profits and profit rates depend upon the current underlying state of the trace, although this dependence has not been reflected in the notation, for simplicity).

The *Least Profit Rate* (LPR) eviction policy is based on evicting, upon a miss, a page in the buffer with minimum profit rate. Profit rates can then be viewed as priorities hence, assuming that ties are resolved consistently for all buffer sizes, LPR is a priority policy and satisfies the inclusion property. Intuitively, LPR is a reasonable heuristics, but in general is not necessarily an optimal policy. However, for the LRUSM, the LPR policy is optimal, as shown in this section. Furthermore, an analytical expression can be derived for profit rates. Profit rates also lead to an efficient computation of the parameters $K(C)$ and $L(C)$ of an optimal policy for all buffer capacities C. To compute the LPR priorities in the LRUSM we first need a lemma.

Lemma 3.4.1. *The expected time spent by an item in each position of the LRU stack is the same.*

Given a cached item at position i in the LRU stack, if we choose to keep it in the buffer until it goes past position j then the average probability of being hit that it will experience is given by the average value of s between i and j, because of Lemma 3.4.1. This profit function has a maximum for some value of j that will define our LPR policy, as shown below.

Definition 3.4.2. *The* LPR *policy evicts the item i in the buffer such that*

$$i^* \stackrel{\Delta}{=} \arg\min_i \max_j \bar{s}(i,j) \tag{3.47}$$

Proof of Lemma 3.4.1. Let t_i^j be the expected time spent by an item in position $j \geq i$ conditioned to the fact the item surely arrives in position i without

getting accessed. We have that

$$t_j^j = 1 + S(j-1)t_j^j \quad \Rightarrow \quad t_j^j = \frac{1}{1 - S(j-1)}$$

The probability that an item, starting from position j does not arrive in position $j+1$ is the sum, for $k \geq 0$, of the probabilities that first arrive k accesses above j and then j is accessed. So the probability for an item in position j to arrive in position $j+1$ is

$$P_{j+1}^j = 1 - s(j) \sum_{k=0}^{+\infty} [S(j-1)]^k = \frac{1 - S(j)}{1 - S(j-1)}$$

and hence the time spent in position $j+1$ is

$$t_{j+1}^j = P_{j+1}^j t_{j+1}^{j+1} = \frac{1 - S(j)}{1 - S(j-1)} \frac{1}{1 - S(j)} = t_j^j$$

Furthermore

$$t_{j+k}^j = P_{j+1}^j P_{j+2}^{j+1} \cdots P_{j+k}^{j+k-1} t_{j+k}^{j+k} = t_j^j$$

As a special case, if we know that an item starts from position 1 (in which is going to spend exactly 1 time step) then is going to spend on average $t_j \triangleq t_j^1 = 1$ in each position on the LRU stack in its lifetime. □

After introducing two lemmas we are now able to prove that the K-L policies are induced by the LPR stack policy and we devise a linear algorithm that, given s, can compute $K(C)$ and $L(C)$ for all buffer capacities C.

Lemma 3.4.3. *Let s be a function from natural to positive real numbers: $s : \mathbb{N} \to \mathbb{R}^+$ and let $\bar{s}$ be its moving average. Let λ be the point of maximum for $\bar{s}$. Then for each $q \leq \lambda$ we have*

$$\bar{s}(\lambda) \leq \bar{s}(q, \lambda)$$

Proof. We can rewrite $\bar{s}(\lambda)$ as

$$\begin{aligned} \bar{s}(\lambda) =& \sum_{j=1}^{\lambda} \frac{s(j)}{\lambda} = \frac{q-1}{\lambda} \sum_{j=1}^{q-1} \frac{s(j)}{q-1} \\ &+ \frac{\lambda - q + 1}{\lambda} \sum_{j=q}^{\lambda} \frac{s(j)}{\lambda - q + 1} = \\ =& \frac{q-1}{\lambda} \bar{s}(q) + \frac{\lambda - q + 1}{\lambda} \bar{s}(q, \lambda) \end{aligned}$$

Being $\bar{s}(\lambda)$ a convex combination of $\bar{s}(q)$ and $\bar{s}(q,\lambda)$ the following inequality holds:

$$\min\{\bar{s}(q), \bar{s}(q,\lambda)\} \leq \bar{s}(\lambda) \leq \max\{\bar{s}(q), \bar{s}(q,\lambda)\}$$

Since for hypothesis we have $\bar{s}(\lambda) \geq \bar{s}(q)$ we must have

$$\bar{s}(q) \leq \bar{s}(\lambda) \leq \bar{s}(q,\lambda)$$

□

Lemma 3.4.4. *Let s be a function from natural to positive real numbers: $s : \mathbb{N} \to \mathbb{R}^+$, let $\bar{s}$ be its moving average and $\bar{s}(q,j)$ be the average of s between q and j. Let λ be the point of maximum for $\bar{s}$. Then for each $r > \lambda$ we have*

$$\bar{s}(\lambda+1, r) \leq \bar{s}(\lambda)$$

Proof. We have the following convex combination:

$$\bar{s}(r) = \frac{\lambda}{r}\bar{s}(\lambda) + \frac{r-\lambda}{r}\bar{s}(\lambda+1, r)$$

with $\bar{s}(\lambda) \geq \bar{s}(r)$ so we must have

$$\bar{s}(\lambda+1, r) \leq \bar{s}(r) \leq \bar{s}(\lambda)$$

□

Theorem 3.4.5. *$K(C)$ and $L(C)$ can be computed using the $\Theta(V^2)$ Algorithm 1.*

Algorithm 1: Computing K and L – Quadratic algorithm.

```
σ ← 1;
repeat
    K ← σ;
    σ ← arg max_{j>K} s̄(K+1, j);
until σ ≥ C ;
L = σ;
```

Proof. Let K and L be the parameters as computed by Algorithm 1. We now consider an alternative two-point eviction policy with parameters q and L_q instead of K and L. We want to prove that the hit rate is improved or not altered moving q and L_q to K and L.

We first note that an optimal q should not be chosen smaller than K. Let $\sigma_1 = 1, \sigma_2, \ldots, \sigma_{n-1} = K, \sigma_n = L$ be the sequence of σ's computed by Algorithm 1 with $\sigma_{h-1} \leq q < \sigma_h \leq \sigma_{n-1}$. Because of Lemmas 3.4.3 and 3.4.4, we have that

$$\begin{aligned} &\forall L_q > \sigma_h \\ &\bar{s}(q+1, \sigma_h) \geq \bar{s}(\sigma_{h-1}+1, \sigma_h) \geq \bar{s}(\sigma_h+1, L_q) \end{aligned}$$

hence, as shown in the proof of Lemma 3.3.6, moving q to σ_h would not decrease the hit rate, proving that we can restrict on values of q not smaller than K.

Let $q \geq K$. First of all we note that P^{hit} can be rewritten as:

$$\mathrm{P}^{hit} = \bar{s}(q)q + \bar{s}(q+1, L_q)(C-q)$$

so we can choose the L_q which maximizes $l \mapsto \bar{s}(q+1, l)$ to obtain the best hit rate given q. Since $q = K$ would bring $L_q = L$ we can now focus on $q > K$. Because of Lemmas 3.4.3 and 3.4.4, we have that

$$\begin{aligned} &\forall L_q > L \\ &\bar{s}(q+1, L) \geq \bar{s}(K+1, L) \geq \bar{s}(L+1, L_q) \end{aligned}$$

therefore an optimal L_q cannot be bigger than L: $L_q \leq L$. We then have two possible cases:

- $\bar{s}(K+1, q) > \bar{s}(q+1, L_q)$: because of Lemma 3.4.3 we have

  $$\bar{s}(L_q+1, L) > \bar{s}(K+1, q) > \bar{s}(q+1, L_q)$$

  hence we could move the step from L_q to L without worsen the hit rate. But at this point, since

  $$\bar{s}(K+1, q) < \bar{s}(q+1, L)$$

  we can remove the step in q ending with $q = K$ and $L_q = L$.

- $\bar{s}(K+1, q) < \bar{s}(q+1, L_q)$: we can improve (or not alter) the hit rate removing the step in q, thus obtaining $q' = K$ and hence $L_{q'} = L$.

□

Theorem 3.4.6. *The K-L replacement policy is induced by the rule: evict the item in position*

$$i^* \stackrel{\Delta}{=} \arg\min_i \max_j \bar{s}(i, j)$$

Proof. Starting with an empty buffer we evict the first time when the top C position of the LRU stack are filled. The position $i^* = \arg\min_i \max_j \bar{s}(i,j)$ is exactly $K+1$ (as can be seen applying Lemmas 3.4.3 and 3.4.4). The following position l that has $\max_j \bar{s}(l,j) < \max_j \bar{s}(i^*,j)$ is $L+1$, so each time an item reaches that position is evicted (it can reach it only after a miss, since the positions after L are not in the buffer).

If the buffer is not initially empty we can however prove this theorem introducing a common hypothesis in control theory for a system. If $\forall j\ s(j) \neq 0$, then the state with the top C positions in the buffer is recurrent under any policy (i.e. the system is unichain [9]) and we can still apply the previous arguments. □

Corollary 3.4.7. *$K(C)$ and $L(C)$ can be computed for all buffer sizes C using the $\Theta(V)$ Algorithm 2, which is adapted from [30] and is a specialization of the Graham scan [21, 8].*

Algorithm 2: Computing K and L as functions of C – Linear algorithm. $K_C = \sigma_k < C \leq \sigma_{k+1} = L_C$ for some k.

```
1  ν[1] ← 1; w[1] ← s(1); d[1] ← 1;
2  w[V + 1] ← 0; d[V + 1] ← 1;
3  for j ← V downto 2 do
4  |   ν[j] ← j; w[j] ← s[j]; d[j] ← 1;
5  |   n ← ν[j] + 1;
6  |   while w[j]/d[j] ≤ w[n]/d[n] do
7  |   |   ν[j] ← ν[n];
8  |   |   w[j] ← w[j] + w[n];
9  |   |   d[j] ← d[j] + d[n];
10 |   |   n ← ν[j] + 1;
11 j ← 1; k ← 1;
12 while j ≤ V do
13 |   print "σ_k = ν[j]";
14 |   j ← ν[j] + 1; k ← k + 1;
```

3.5 Buffer Partitioning

An interesting application of the LPR policy arises when we want to partition a buffer of capacity C among n independent processes. We assume that the

i-th process accesses a private address space size V_i, according to the LRUSM with stack-depth distribution s_i. At each step the i-th process has probability ψ_i ($\sum_{i=1}^{n} \psi_i = 1$) to be the one to run. We want to determine the capacity C_i to devote to process i (with $\sum_{i=1}^{n} C_i = C$) so as to minimize the global miss rate over an infinite temporal horizon, under the hypothesis that each process is using the optimal LPR policy for its distribution s_i.

Let $M_i(C_i)$ be the expected miss rate for process i as a function of the buffer capacity C_i. We want to find a partition $[C_i^*]_i$ such that the overall miss rate is minimized over the possible partitions:

$$[C_i^*]_i = \arg\min_{[C_i]_i} \sum_i \psi_i \, M_i(C_i).$$

It has been proved [43, 45, 18] that, if all functions

$$\gamma_i(C_i) = M_i(C_i - 1) - M_i(C_i),$$

called *marginal gains*, are non increasing (and hence the miss rates are convex), then the optimal partition can be found by a simple, efficient greedy algorithm. It turns out that the miss rate of the optimal LPR policy is convex (which is not necessarily true for LRU). In fact, a relatively straightforward analysis shows that the marginal gains of the optimal policy are constant within every segment of the partition of s obtained by Alg. 2 (specifically, if C_i is in the segment $[\sigma_k + 1, \sigma_{k+1}]$, then $\gamma_i(C_i) = \bar{s}_i\,(\sigma_k + 1, \sigma_{k+1}))$ and decrease from segment to segment.

Based on these properties and on the linearity of our preprocessing algorithm (Alg. 2), the optimal buffer partition can be computed in linear time by the very simple algorithm Alg. 3.

Algorithm 3: Optimal partitioning. σ_k^i represents the last item slot of the k-th chunk of the LRU stack partition of the i-th process. At the end the vector p contains the number of slots to allocate to each process.

$\forall i \quad k[i] \leftarrow 1, p[i] \leftarrow 1;$
$R \leftarrow C - n;$
while $R \neq 0$ **do**
  $i^* \leftarrow \arg\max_i \bar{s}\left(\sigma^i_{k[i]} + 1, \sigma^i_{k[i]+1}\right);$
  $l \leftarrow \sigma^{i^*}_{k[i^*]+1} - \sigma^{i^*}_{k[i^*]};$
  $\Delta \leftarrow \min\{l, R\};$
  $p[i^*] \leftarrow p[i^*] + \Delta;$
  $R \leftarrow R - \Delta;$
  $k[i^*] \leftarrow k[i^*] + 1;$

CHAPTER 4

On-line vs. Off-line Optimality in Page Replacement

Intuitively, the optimal on-line policy of the preceding chapter makes the best possible use of the statistical knowledge of the future address trace. How does the performance so achieved compare with that of a policy like OPT, which makes the best use of the complete knowledge of the future trace? To explore this question we study the ratio χ between the miss rates of LPR and OPT:

$$\chi \stackrel{\Delta}{=} \max_s \chi(s) \tag{4.1}$$

$$\text{where} \quad \chi(s) \stackrel{\Delta}{=} \frac{M^{\mathrm{LPR}}(s)}{M^{\mathrm{OPT}}(s)}, \tag{4.2}$$

a kind of average competitive ratio. In this chapter we will derive a result of the form $\chi \leq f(V, C)$.

Determining M^{OPT} for a given stack-depth distribution $s(\cdot)$ appears to be difficult, but we can derive a useful lower bound, that we prove after giving some definitions.

$$\begin{aligned} L_G^{\mathrm{OPT}}(s) &\stackrel{\Delta}{=} G \left(\sum_{g=0}^{C+G-1} \frac{1}{1-S(g)} \right)^{-1} \\ L^{\mathrm{OPT}}(s) &\stackrel{\Delta}{=} \max_{G \in \{1,\ldots,V-C\}} L_G^{\mathrm{OPT}}(s) \end{aligned} \tag{4.3}$$

Theorem 4.0.1. *Under the LRUSM, the miss rate of OPT is bounded from below as*

$$M^{\mathrm{OPT}}(s) \geq L^{\mathrm{OPT}}(s) \tag{4.4}$$

Proof. Let ϕ_j be the random variable, with support the positive integers, describing the time steps needed to access a stack position at depth lower than j. We can see that ϕ_j is a geometric random variable, with parameter $p_j = 1 - S(j)$, $\mathrm{P}[\phi_j = k] = (1-p_j)^{k-1}p$ and expected value $1/p$.

The time steps needed to observe $C+G$ distinct items are described by the random variable τ, defined as

$$\tau \triangleq \sum_{j=0}^{C+G-1} \phi_j \tag{4.5}$$

whose expected value is, by linearity,

$$\mathbb{E}[\tau] = \sum_{j=0}^{C+G-1} \frac{1}{1-S(j)} \tag{4.6}$$

Let us consider a sequence of q intervals with $C+G$ distinct items, of i.i.d. lengths τ_i. Since each interval with $C+G$ references to distinct items contains at least G misses, the miss rate for any policy (including OPT) is bounded by

$$M(s) \geq \lim_{q\to\infty} \mathbb{E}\left[\frac{qG}{\sum_{i=1}^{q}\tau_i}\right] \quad \forall G \tag{4.7}$$

which can be rewritten as

$$\begin{aligned} M(s) &\geq G \lim_{q\to\infty} \mathbb{E}\left[\frac{q}{\sum_{i=1}^{q}\tau_i}\right] = G \lim_{q\to\infty} \mathbb{E}\left[\frac{1}{\bar{\tau}_q}\right] \\ \text{where } \bar{\tau}_q &\triangleq \sum_{i=1}^{q} \frac{\tau_i}{q} \end{aligned} \tag{4.8}$$

Since $\bar{\tau}_q$ converges in distribution to the delta peaked at $\mathbb{E}[\tau]$ (large number law) and in its support $x \in \mathbb{Z} : x \geq C+G$ the function $1/x$ is continuous and bounded we obtain [31]

$$M(s) \geq G \lim_{q\to\infty} \mathbb{E}\left[\frac{1}{\bar{\tau}_q}\right] = G\,\mathbb{E}\left[\frac{1}{\lim_{q\to\infty}\bar{\tau}_q}\right] = \frac{G}{\mathbb{E}[\tau]} \tag{4.9}$$

□

Corollary 4.0.2.

$$\chi(s) \triangleq \frac{M^{\mathrm{LPR}}(s)}{M^{\mathrm{OPT}}(s)} \leq \frac{M^{\mathrm{LPR}}(s)}{L^{\mathrm{OPT}}(s)}. \tag{4.10}$$

4.1 Uniform distribution miss-rate ratio

We now derive $\chi(s_u)$ when $s_u(j) = 1/V$ is the constant access distribution function, i.e. the distribution that accesses uniformly at random all the items. We will prove later that $\chi = \chi(s_u)$, i.e. the value we obtain is the largest possible for all distributions. Eq. (4.3), with $G = \frac{V-C}{2}$, yields

$$L^{\mathrm{OPT}}(s_u) \simeq \frac{V-C}{2V \log\left(\frac{2V}{V-C}\right)}, \tag{4.11}$$

whereas, straightforwardly,

$$M^{\mathrm{LPR}}(s_u) = \frac{V-C}{V}, \tag{4.12}$$

whence, using Eq. 4.11,

$$\chi(s_u) \leq \frac{M^{\mathrm{LPR}}(s_u)}{L^{\mathrm{OPT}}(s_u)} \simeq 2 \log\left(\frac{2V}{V-C}\right). \tag{4.13}$$

Remark 4.1.1. *Actually the value* $G^* = \arg\max_G L_G^{\mathrm{OPT}}(s_u)$ *is not* $\frac{V-C}{2}$. *Approximating the harmonic functions with logarithms we can write, for* $G \neq V - C$*:*

$$L_G^{\mathrm{OPT}}(s_u) = \frac{G}{V \log\left(\frac{V}{V-C-G}\right)} \tag{4.14}$$

We can find G^* *setting the derivative of* $L_G^{\mathrm{OPT}}(s_u)$ *to zero: let* $\mu = \frac{C}{V}$, $\beta = 1 - \mu$ *and* $\gamma^* = \frac{G^*}{V}$, *then* γ^* *is the unique zero for* $\gamma \in [0, \beta[$ *of the following function*

$$f(\gamma) = (\gamma - \beta) + \exp\left(\frac{\gamma}{\gamma - \beta}\right) \tag{4.15}$$

A rough approximation (see Fig.4.1) is given by

$$\gamma^* \simeq \frac{3}{2}\left(\sqrt{\mu} - \mu\right) \tag{4.16}$$

4.2 Worst-Case Miss-Rate Ratio

The next theorem, preceded by a lemma, states that, given a distribution s, we can obtain a nearly uniform distribution s' that induces the same online optimal policy K-L with the same miss-rate, but a lower L^{OPT}. We then show that the χ obtained with this kind of functions is the same obtained by the uniform one s_u shown in Eq. (4.13), which is thus optimal.

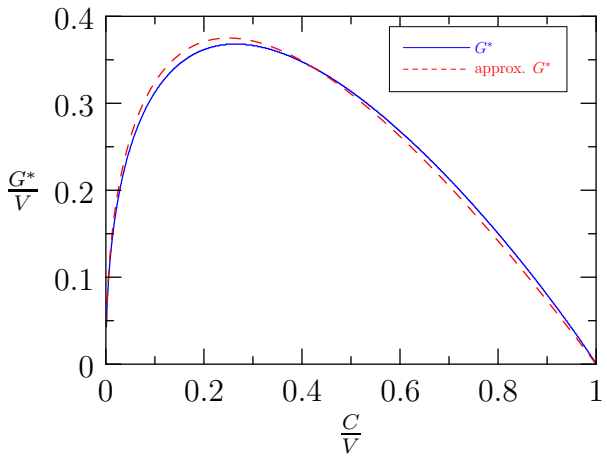


Figure 4.1: Uniform distribution. G^* and its approximation.

Lemma 4.2.1. *If* $\forall j\ S_1(j) \geq S_2(j)$ *then* $L^{\mathrm{OPT}}(s_1) \leq L^{\mathrm{OPT}}(s_2)$

Proof. From the definition we can see that L^{OPT} is decreasing in $\sum_{g=0}^{C+G-1} \frac{1}{1-S(g)}$, and hence decreasing in S. □

Theorem 4.2.2. *Let* s *be an LRU stack access distribution,* LPR *the associated optimal online policy. Let* $s_{\sigma,\eta}$ *be the following distribution, parametrized by* σ *and* η *in* $[0,1]$*:*

$$s_{\sigma,\eta}(j) = \begin{cases} \sigma & j = 1 \\ \eta & j \in \{2, \ldots, L+d\} \\ 1 - S'(L+d) & j = L+d+1 \\ 0 & j > L+d+1 \end{cases} \tag{4.17}$$

where

$$d \triangleq \left\lfloor \frac{1 - S'(L)}{\eta} \right\rfloor \tag{4.18}$$

Let $X(s)$ *be the set of distribution that have the same* K *and* L *parameters of* s *with the same miss-rate and let* s' *be the distribution in* X *that has the lowest miss-rate:*

$$s' \triangleq \arg\min_{x \in X} L^{\mathrm{OPT}}(x) \tag{4.19}$$

Then s' *is a* $s_{\sigma,\eta}$ *distribution for some suitable parameters* σ *and* η*.*

Proof. Given s we can obtain a distribution s' decreasing OPT's lower bound setting

$$\eta \triangleq \bar{s}(K+1, L) \qquad \sigma \triangleq S(K) - (K-1)\eta \tag{4.20}$$

which has $\forall j\ S'(j) \geq S(j)$ and hence, because of Lemma 4.2.1, has a smaller L^{OPT}. □

Corollary 4.2.3.

$$\chi \leq 2 \log \left(\frac{2V}{V - C} \right) \tag{4.21}$$

Proof. Thanks to the similarity between $s_{\sigma,\eta}$ and s_u, if we repeat the analysis in Section 4.1 we obtain the same bound. □

The following corollary agrees with many results observed in simulations on real traces [7, 49].

Corollary 4.2.4. *If $C \leq \beta V$, for some $\beta < 1$, then $\chi = O(1)$,*

Theorem 4.2.5. *The bound given in Eq. 4.13 is existentially tight, since for $C = V - 1$, we have*

$$M^{\mathrm{LPR}} = \frac{1}{V} \qquad M^{\mathrm{OPT}} \simeq \frac{1}{V \, \mathrm{H}_V} \qquad \chi \simeq \mathrm{H}_V \tag{4.22}$$

where H_V is the V-th harmonic number ($\mathrm{H}_V = \sum_{i=1}^{V} \frac{1}{i} = \log V + \gamma + \Theta\left(\frac{1}{V}\right)$, with $\gamma \simeq 0.577$ being the Euler-Mascheroni constant).

Proof. Let N be the time between two consecutive misses using OPT's policy. We have a miss if and only if in the last N references there are V distinct items, and this happens on average every $V \, \mathrm{H}_V$ steps, so the expected value of N is $\mathbb{E}[N] = V \, \mathrm{H}_V$. Since N is sharply concentrated around its expected value [34], i.e.

$$\mathrm{P}\left[N > V \, \mathrm{H}_V + cV\right] \simeq 1 - e^{-e^{-|c|}} \tag{4.23}$$

we obtain $\mathbb{E}[1/N] \simeq 1/\,\mathbb{E}[N]$ and the thesis follows. □

CHAPTER 5

Optimal Replacement for Finite Horizon

From a practical perspective, when dealing with sufficiently long address traces, a policy that is optimal over an infinite horizon will likely achieve near optimal performance. For shorter traces, transient effects may play a significant role, whence the interest in understanding the structure of optimal policies over a finite horizon. In principle, the optimal policy can be computed by a dynamic-programming algorithm based on Eq. 2.10, but the exponential number of states makes this approach of rather limited applicability. An alternate route, which has been successfully followed for several optimal control problems, is to guess a closed form characterization of a policy π and its corresponding optimal cost function $J^{\pi}_{\tau}(\cdot)$. Under very mild conditions, if the guess satisfies Eq. 2.10, then π is an optimal policy. We have been unable to take this route; the obstacle lies in finding a tractable form for the optimal cost. Ultimately, we have circumvented this obstacle for monotone stack-depth distributions, by realizing that what is really needed to make an optimal choice between two states is not the absolute value of their costs, but rather their relative value. And even the latter is only needed when the two states are both candidate next states in the same step. The results are stated next.

Theorem 5.0.6. *Let s be non increasing, with $s(j) \geq s(j+1)$ for $j \in \{1, V-1\}$. Then, for any finite horizon $\tau \geq 1$, LRU is an optimal replacement policy.*

Theorem 5.0.7. *Let s be non decreasing, with $s(j) \leq s(j+1)$ for $j \in \{1, V-1\}$. Then, for any finite horizon $\tau \geq 1$, MRU is an optimal replacement policy.*

Thus, for monotone stack-depth distributions, the finite-horizon optimal policy is time invariant, hence it is also optimal over an infinite horizon. This property does not hold for arbitrary distributions.

In spite of the symmetry between the statements of the above two theorems, their proofs, given in the following two sections, require significantly different ideas.

5.1 Non-Increasing Access Distribution

Definition 5.1.1. *Let $R_d(x)$ denotes the state resulting by applying a right cyclic-shift to the prefix of length d of x (strictly speaking, if $x(d) = 0$ then $R_d(x)$ is a pseudo-state, as it is not in the admissible state set). The* Least Recently Used *(LRU) policy is defined (for a miss, $x(d) = 0$) by*

$$\begin{aligned} \mathrm{LRU}(x, d) &= \max\{j : \ y(j) = 1\} \\ &\textit{where } y = R_d(x). \end{aligned} \tag{5.1}$$

In other words, the item evicted is the one in the deepest position of the (resulting) stack, among those that are in the buffer.

Definition 5.1.2. *Two states y and z are said to form a* critical pair *if their structure is related as follows:*

$$\begin{aligned} y &= 1\pi 1\iota 0\sigma, \\ z &= 1\pi 0\iota 1\sigma. \end{aligned} \tag{5.2}$$

for any π, ι and for any $\sigma \in 0^*$.

Remark 5.1.3. *A critical pair represents a choice between what would the LRU policy do (obtaining y) and what would a different eviction policy do (obtaining z), when choosing the item to evict after the stack rotation.*

Remark 5.1.4. *Given a critical pair as in the definition above we write $y <_c z$ to remark that in the last stack position in which y and z differ y has a zero and z has a one. We similarly define the operator $\leq_c$ as:*

$$y \leq_c z \quad \Leftrightarrow \quad y = z \ \vee \ y <_c z \tag{5.3}$$

Lemma 5.1.5. *The evolution of a critical pair under LRU preserves its criticality and order, i.e.:*

$$\forall y \forall z \ : \ y <_c z \quad \forall w \quad f^{\mathrm{LRU}}(y, w) \leq_c f^{\mathrm{LRU}}(z, w) \tag{5.4}$$

Proof. We can divide the w access in four cases:

Hit for both y and z The two stacks rotate and they are still a critical pair with $y <_c z$

Miss for both y and z The two evictions in the last filled positions make the states equal if $\iota \in 0^*$, otherwise they yield $y <_c z$

Hit for y and miss for z The eviction in z yields $y = z$

Miss for y and hit for z The eviction in y brings $y = z$ if $\iota \in 0^*$ and $y <_c z$ otherwise

□

Theorem 5.1.6. *Let s be monotonic non decreasing, then LRU is the optimal eviction policy for every time horizon τ. Furthermore:*

$$\forall \tau \, \forall y \, \forall z \, : \, y \leq_c z \quad J^*_\tau(y) \leq J^*_\tau(z) \tag{5.5}$$

Proof (by induction on τ). **Base case.** For $\tau = 1$ we have

$$\forall x \quad J^*_1(x) = \mathbb{E}_w \left[g(x, w) \right] \triangleq \bar{g}(x) \, , \tag{5.6}$$

from which, using s monotonicity, follows

$$\forall y \, \forall z \, : \, y \leq_c z \quad J^*_1(y) \leq J^*_1(z) \, . \tag{5.7}$$

Induction. We assume now the previous inductive hypothesis for all $t < \tau$, obtaining

$$\begin{aligned} \forall x \quad J^*_\tau(x) &= \bar{g}(x) + \mathbb{E}_w \left[\min_u J^*_{\tau-1} \left(f(x, w, u) \right) \right] \\ &= \bar{g}(x) + \mathbb{E}_w \left[J^*_{\tau-1} \left(f^{\mathrm{LRU}}(x, w) \right) \right] . \end{aligned} \tag{5.8}$$

Since $\bar{g}(y) \leq \bar{g}(z)$ and since, by the inductive hypothesis and Lemma 5.1.5,

$$J^*_{\tau-1} \left(f^{\mathrm{LRU}}(y, w) \right) \leq J^*_{\tau-1} \left(f^{\mathrm{LRU}}(z, w) \right) \tag{5.9}$$

we finally obtain $J^*_\tau(y) \leq J^*_\tau(z)$ and the induction is completed. □

Proof of Thm. 5.0.6. Follows directly from Thm. 5.1.6 □

5.2 Non-Decreasing Access Distribution

First we note that we can decompose the miss rate in a very useful manner: imagine to put an observer in every out-of-buffer item, with the observer following the item going down the LRU stack during the system evolution; when an out-of-buffer item is accessed the observer moves to the item evicted by the replacement policy. Thus the set Ω of the observers remains the same during the evolution and does not depend on the time t.

Let d_t be the access depth at time t, let ω be an observer and $l_t^\mu(\omega, x_1, d_{t'<t})$ be the stack depth of the ω observer at time t. We are interested in the event *the item observed by ω is accessed at time t* (i.e. $d_t = l_t^\mu(\omega, x_1, d_{t'<t})$). We can decompose the misses in τ steps using observers $\omega \in \Omega$ as:

$$\begin{aligned}\Gamma_\tau^\mu(x_1) &= \sum_{t=1}^{\tau} \mathrm{P}_{\mathrm{MISS}}(t) \\ &= \sum_{t=1}^{\tau} \sum_{\omega\in\Omega} \mathrm{P}[d_t = l_t^\mu(\omega, x_1, d_{t'<t})] \\ &= \sum_{\omega\in\Omega} \sum_{t=1}^{\tau} \mathrm{P}[d_t = l_t^\mu(\omega, x_1, d_{t'<t})]\end{aligned} \tag{5.10}$$

Let ω_j be the observer which is at depth j at time zero. Under MRU the evolution of an observer ω_j does not depend on x_1 but only on the initial position of the observed item (i.e. $l_t^{\mathrm{MRU}}(\omega_j, x_1, d_{t'<t}) = l_t^{\mathrm{MRU}}(\omega_j, d_{t'<t}) = l_t(\omega_j)$ for brevity), hence the contribution observers gives to the cost of different initial states depends only on their initial positions. E.g. if we have two states x_1 and x_2 which differ for only two observers ω_i and ω_j we can write their costs as:

$$\begin{aligned}\Gamma_1 &= \sum_{\omega\in\Omega\setminus\{\omega_i\}} \sum_{t=1}^{\tau} \mathrm{P}[d_t = l_t(\omega)] + \sum_{t=1}^{\tau} \mathrm{P}[d_t = l_t(\omega_i)] \\ \Gamma_2 &= \sum_{\omega\in\Omega\setminus\{\omega_j\}} \sum_{t=1}^{\tau} \mathrm{P}[d_t = l_t(\omega)] + \sum_{t=1}^{\tau} \mathrm{P}[d_t = l_t(\omega_j)]\end{aligned} \tag{5.11}$$

where the first term is equal in both the costs, because it is due to observers which start in the same position for both states, and thus:

$$\Gamma_1 - \Gamma_2 = \sum_{t=1}^{\tau} \mathrm{P}[d_t = l_t(\omega_i)] - \sum_{t=1}^{\tau} \mathrm{P}[d_t = l_t(\omega_j)] \tag{5.12}$$

i.e. the difference in the costs depends only in the items observed by the different observers.

Let us define $\gamma_\tau(i)$ as

$$\gamma_\tau(i) = \sum_{t=1}^{\tau} \mathrm{P}[d_t = l_t(\omega_i)] \tag{5.13}$$

i.e. the contribution to the total cost due to items observed by ω_i, the observer that at time zero is in position i (not in the buffer). As noted before this contribution under MRU does not depend on the initial state (given that i is not in the buffer).

We now just need to prove that $\gamma_\tau(i) \leq \gamma_\tau(j)$ if $i < j$ and we do it by induction in the following lemma:

Theorem 5.2.1. *Let s be non decreasing:*

$$\forall j \in \{1, V-1\} \quad s(j) \leq s(j+1) \tag{5.14}$$

and MRU the policy we apply. If

$$\begin{aligned} &\forall t \leq \tau \quad \forall i \, \forall j : i < j \\ &\gamma_t(2) \leq \gamma_t(i) \leq \gamma_t(j) \leq 1 + \gamma_t(2) \end{aligned} \tag{5.15}$$

then

$$\begin{aligned} &\forall i \, \forall j : i < j \\ &\gamma_{\tau+1}(2) \leq \gamma_{\tau+1}(i) \leq \gamma_{\tau+1}(j) \leq 1 + \gamma_{\tau+1}(2) \end{aligned} \tag{5.16}$$

Proof. **Base case.** For $t = 1$ we have

$$\forall k \quad \gamma_t(k) = s(k)$$

and hence

$$\gamma_t(2) \leq \gamma_t(i) \leq \gamma_t(j)$$

Furthermore we have that

$$\gamma_t(k) = s(k) \leq 1 \leq 1 + \gamma_t(2)$$

Induction.

$$\begin{aligned} \gamma_{\tau+1}(i) &= s(i)(1 + \gamma_\tau(2)) \\ &\quad + S(i-1)\gamma_\tau(i) + (1 - S(i))\gamma_\tau(i+1) \\ &\leq s(i)(1 + \gamma_\tau(2)) + (1 - s(i))\gamma_\tau(i+1) \\ &\leq s(i)(1 + \gamma_\tau(2)) + (1 - s(i))\gamma_\tau(j) \\ &= s(i)(1 + \gamma_\tau(2)) + s(j)\gamma_\tau(j) \\ &\quad + (1 - s(i) - s(j))\gamma_\tau(j) \end{aligned}$$

$$\begin{aligned}\gamma_{\tau+1}(j) &= s(j)(1+\gamma_\tau(2)) \\ &\quad + S(j-1)\gamma_\tau(j) + (1-S(j))\gamma_\tau(j+1) \\ &\geq s(j)(1+\gamma_\tau(2)) + (1-s(j))\gamma_\tau(j) \\ &= s(j)(1+\gamma_\tau(2)) + s(i)\gamma_\tau(j) \\ &\quad + (1-s(i)-s(j))\gamma_\tau(j) \\ \\ &\Rightarrow \gamma_\tau(i) \leq \gamma_\tau(j)\end{aligned}$$

Finally under MRU we have

$$\begin{aligned}\gamma_{\tau+1}(k) &= s(k)(1+\gamma_\tau(2)) \\ &\quad + S(k-1)\gamma_\tau(k) + (1-S(k))\gamma_\tau(k+1) \\ &\geq \min\{1+\gamma_\tau(2), \gamma_\tau(k), \gamma_\tau(k+1)\} \\ &= \gamma_\tau(k)\end{aligned}$$

and

$$\begin{aligned}\gamma_{\tau+1}(k) &= s(k)(1+\gamma_\tau(2)) \\ &\quad + S(k-1)\gamma_\tau(k) + (1-S(k))\gamma_\tau(k+1) \\ &\leq \max\{1+\gamma_\tau(2), \gamma_\tau(k), \gamma_\tau(k+1)\} \\ &= 1+\gamma_\tau(2)\end{aligned}$$

and therefore

$$\gamma_{\tau+1}(k) \leq 1+\gamma_{\tau+1}(2)$$

□

Proof of Thm. 5.0.7. Follows directly from Thm. 5.2.1 □

CHAPTER 6

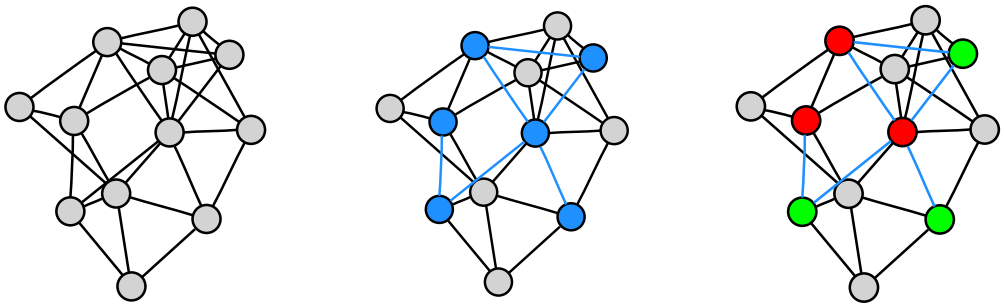
An Introduction to the Galois System

6.1 Preliminaries

The advent of on-chip multiprocessors has made parallel programming a mainstream concern. Unfortunately writing correct and efficient parallel programs is a challenging task, not solvable by the average programmer. Hence, in recent years, many projects [25, 20, 4, 38] have been started to try to automate parallel programming for some classes of algorithms, mostly focusing regular problems, such as the DFT [19, 37] or linear algebra routines [11]. Actually the automation is even more difficult when the algorithms present irregular data access, typical, e.g., of algorithms on graphs or other pointer-based data structures.

A typical property that many irregular applications exhibit is *amorphous data parallelism* [33], in which data accesses can be organized in worklists, which dynamically evolve during the program execution. Examples of algorithms within this category are:

- *Survey propagation* [13] – A powerful heuristic for NP-complete problems
- *Boruvka's algorithm* [14] – Minimum spanning tree algorithm
- *Delauney triangulation and refinement* [22] – Mesh generation algorithms
- *Agglomerative clustering* [44] – Clustering algorithm used in data-mining



(i) (ii) (iii)

Figure 6.1: Optimistic parallelization in the Galois System. (i) Nodes represent possible computations, edges conflicts between them. (ii) m nodes are chosen at random and run concurrently. (iii) At runtime the conflicts are detected, some nodes abort and their execution is rolled back, leaving a maximal independent set in the subgraph induced by the initial nodes choice.

- *Barnes-Hut* [6] – Computational kernel for n-body interactions simulations

Galois The Galois Project [28], developed at the University of Texas at Austin, aims to provide automatic optimization tools for amorphous data parallel problems by means of *optimistic parallelization*: the tasks in the worklists are executed speculatively in parallel and, whenever at runtime a conflict is detected the execution is rolled-back.

The Model For our purposes we can model the Galois System as working on a dynamic graph $G_t = (V_t, E_t)$, where the nodes represent computations we want to do, but we have no initial knowledge of the edges, which represent conflicts between computations (see Fig. 6.1). At time step t the system picks m_t nodes (the *active* nodes) and tries and process them concurrently. When it processes a node it figures out if it has some connections with other executed nodes and, if a neighbor node happens to have been processed before it, aborts, otherwise the node is considered processed, is removed from the graph and some operations may be performed in the neighborhood, such as adding new nodes with edges or altering the neighbors.

We are interested in building a control system that can help the Galois System to choose, for each temporal step t, an m_t which guarantees as high parallelism as possible with few conflicts, hence achieving a good use of the processors time.

Remark 6.1.1. *Note that if we naïvely try to minimize the total execution*

time the system is forced to use always all the available processors, whereas if we try to minimize the time wasted from aborted processes the system uses only one processor.

6.2 Optimization Goal

Let t be the time step, $n_t \triangleq |V_t|$, m_t the number of processors chosen to be run at time step t, π_m the permutation of the m_t nodes as they try to commit in the Galois System (i.e., if $i < j$ and $\pi_m(i)$ and $\pi_m(j)$ conflict, then $\pi_m(i)$ commits and $\pi_m(j)$ aborts), $k_t(\pi_m)$ the number of aborted processes due to conflicts and $r_t(\pi_m) \in [0, 1)$ the *ratio of conflicting processors* observed at time t (i.e. $r_t(\pi_m) \triangleq k_t(\pi_m)/m_t$). We define $\bar{r}_t(m)$ to be the expected r we obtain when the system is run with m processors:

$$\bar{r}_t(m) \triangleq \mathbb{E}_{\pi_m}\left[r_t(\pi_m)\right]$$

where the expectation is computed uniformly over the possible prefixes of length m of the nodes permutations.

We can now precisely formulate the control problem in the following way: given $r(\tau < t)$ choose $m_t = \mu_t$ where $\bar{r}_t(\mu_t) \simeq \rho$.

Remark 6.2.1. *If we want to dynamically control the number of processors, ρ must be chosen different from zero, otherwise the system converges to use one single processor, thus not being able to identify available parallelism. A value of $\rho = 20 \div 30\%$ is often reasonable, together with the constraint $m_t \geq 2$.*

In the following we suppose that the properties of G_t are varying slowly compared to the convergence of m_t toward μ_t under the algorithms we develop (hypothesis that is verified in the benchmark applications shown in [27]), so we can consider $G_t = G$ and $\mu_t = \mu$. An important property of $\bar{r}$ is given by the following theorem.

Theorem 6.2.2. *If the processors are chosen uniformly at random then $\bar{r}_t(m)$ is non-decreasing in m.*

To prove Thm. 6.2.2 we first need a lemma:

Lemma 6.2.3. *Let $\bar{k}(m) \triangleq \mathbb{E}_{\pi_m}\left[k(\pi_m)\right]$. Then $\bar{k}$ is a non-decreasing convex function, i.e. $\Delta_{\bar{k}}(m) \geq 0$ and $\Delta^2_{\bar{k}}(m) \geq 0$.*

Proof. Let $\tilde{k}(\pi_m, i)$ be the expected number of conflicting nodes running $r = m+i$ nodes concurrently, the first m of which are π_m and the last i are chosen uniformly at random among the remaining ones. We have

$$\tilde{k}(\pi_m, 1) = k(\pi_m) + \mathrm{P}\left[(m+1)\text{-th conflicts}\right] \tag{6.1}$$

which brings

$$\bar{k}(m+1) = \mathbb{E}_{\pi_m}\left[\tilde{k}(\pi_m, 1)\right] = \bar{k}(m) + \eta \tag{6.2}$$

with $\eta = \bar{k}(m+1) - \bar{k}(m) = \Delta_{\bar{k}}(m) \geq 0$. Now consider

$$\tilde{k}(\pi_m, 2) = k(\pi_m) + \mathrm{P}\left[(m+1)\text{-th conflicts}\right] + \mathrm{P}\left[(m+2)\text{-th conflicts}\right] \tag{6.3}$$

If the $(m+1)$-th node does not add any edge, then we have $\mathrm{P}\left[(m+1)\text{-th conflicts}\right] = \mathrm{P}\left[(m+2)\text{-th conflicts}\right]$, but since it may add some edges the probability of conflicting the second time is in general larger and thus $\Delta^2_{\bar{k}}(m) \geq 0$. □

Proof of Thm. 6.2.2. Since $\bar{r}(m) = \bar{k}(m)/m$, its finite difference can be written as

$$\Delta_{\bar{r}}(m) = \frac{m\Delta_{\bar{k}}(m) - \bar{k}(m)}{m(m+1)} \tag{6.4}$$

Because of Lemma 6.2.3 and being $\bar{k}(1) = 0$ we have

$$\bar{k}(m+1) \leq m\Delta_{\bar{k}}(m) \tag{6.5}$$

which finally brings

$$\Delta_{\bar{r}}(m) \geq \frac{\Delta_{\bar{k}}(m)}{m(m+1)} \geq 0 \tag{6.6}$$

□

CHAPTER 7

Controlling Parallelism in the Galois System

7.1 Exploiting Parallelism

A measure of the available parallelism for a given conflict graph has been identified by the Galois group computing at each temporal step a maximal independent set, processing the nodes in this set, applying changes required by the used algorithm to the graph, and then iterating the process [27].

The expected size of a maximal set gives a reasonable and computable esteem of the available parallelism. Furthermore we can obtain a lower bound of it once we know (or can estimate) the sparsity of the graph G. In fact the average degree of the computations/conflicts graph is linked to the expected size of a maximal independent set of the graph by the following theorem, in the variant shown in [3]:

Theorem 7.1.1. *(Turán, stronger formulation). Let $G = (V, E)$ be a graph, $n = |V|$ and let d be the average degree of G. Then the expected size of a maximal independent set, obtained from a random permutation, is at least $s = n/(d + 1)$.*

Remark 7.1.2. *The above bound is existentially tight: let K_d^n be the graph made up of $s = n/(d + 1)$ cliques of size $d + 1$, then the average degree is d and the size of every maximal (and maximum) independent set is exactly s. Furthermore, every other graph with the same number of nodes and edges has a smaller average maximal independent set.*

The properties of the graph K_d^n has suggested us to formulate the following extension of Turán's theorem, that allows, when given a target conflict ratio ρ, to compute a lower bound for the parallelism the Galois System can exploit:

Theorem 7.1.3. *The worst case for the Galois System among the graphs with the same number of nodes and edges is obtained for the graph K_d^n (for which we can analytically approximate the performance, as shown in Sec. 7.1.1). In graph-theoretic language we want to prove that the average maximal independent set size of the subgraph induced by a uniformly random choice of nodes is minimum for the graph K_d^n.*

To prove it we first need the following lemma.

Lemma 7.1.4. *The function $\eta_j(x) \triangleq \prod_{i=1}^{j}(n-i-x)$ is convex for $x \in [0, n-j]$*

Proof. We prove by induction on j that

$$\eta_j(x) \geq 0 \qquad \eta_j'(x) \leq 0 \qquad \eta_j''(x) \geq 0 \tag{7.1}$$

Base case Let $\eta_0(x) = 1$. The properties above are easily verified.

Induction Since $\eta_j(x) = \eta_{j-1}(x)(n-j-x)$, we obtain

$$\eta_j'(x) = -\eta_{j-1}(x) + (n-j-x)\eta_{j-1}'(x) \tag{7.2}$$

which is non-positive by inductive hypotheses. Similarly

$$\eta_j''(x) = -2\eta_{j-1}'(x) + (n-j-x)\eta_{j-1}''(x) \tag{7.3}$$

is non-negative. □

Proof of Thm. 7.1.3. Consider a random permutation π of the nodes of a generic graph G that has the same number of nodes and edges of K_d^n. We assume the prefix of length m of π (i.e. $\pi(1), \ldots, \pi(m)$) forms the active nodes and focus on the following independent set IS in the subgraph induced: a node v is in $\mathrm{IS}(G, \pi)$ if and only if it is in the first m positions of π and it has no neighbors preceding it. Let $b(G)$ be the expected size of $\mathrm{IS}(G, \pi)$ averaged over all possible π's (chosen uniformly):

$$b(G) \triangleq \mathbb{E}_\pi \left[\#\,\mathrm{IS}(G, \pi)\right] \tag{7.4}$$

Since for construction b is a lower bound for the average maximal induced independent set AMIS(G) which gives Galois' performance ($b(G) \leq \text{AMIS}(G)$) whereas $b(K_d^n) = \text{AMIS}(K_d^n)$, we just need to prove that $b(K_d^n) \leq b(G)$.

Given a generic node v of degree d_v and a random permutation π, its probability to be in IS(G, π) is

$$\mathrm{P}\left[v \in \text{IS}(G,\pi)\right] = \frac{1}{n}\sum_{j=1}^{m}\prod_{i=1}^{j}\frac{n-i-d_v}{n-i} \tag{7.5}$$

By the linearity of the expectation we can write b as

$$b(G) = \frac{1}{n}\sum_{v=v_1}^{v_n}\sum_{j=1}^{m}\prod_{i=1}^{j}\frac{n-i-d_v}{n-i} = \mathbb{E}_v\left[\sum_{j=1}^{m}\prod_{i=1}^{j}\frac{n-i-d_v}{n-i}\right] \tag{7.6}$$

$$b(K_d^n) = \sum_{j=1}^{m}\prod_{i=1}^{j}\frac{n-i-d}{n-i} = \sum_{j=1}^{m}\prod_{i=1}^{j}\frac{n-i-\mathbb{E}_v[d_v]}{n-i} \tag{7.7}$$

To prove that AMIS(G) $\geq$ AMIS(K_d^n) is thus enough showing that

$$\forall j \qquad \mathbb{E}_v\left[\prod_{i=1}^{j}(n-i-d_v)\right] \geq \prod_{i=1}^{j}(n-i-\mathbb{E}_v[d_v]) \tag{7.8}$$

which can be done applying Jensen's inequality [24], since in Lemma 7.1.4 we have proved the convexity of $\eta_j(x) \triangleq \prod_{i=1}^{j}(n-i-x)$. □

7.1.1 Analysis of the Worst-Case Performance

Theorem 7.1.5. *Let d be the average degree of $G = (V, E)$, $n = |V|$ and $s = n/(d+1)$ (for simplicity we suppose $s \in \mathbb{N}$). The conflict ratio is bounded by*

$$\bar{r}(m) \leq 1 - s\frac{1-e^{-\frac{m}{s}}}{m} \tag{7.9}$$

Proof. We can study the performance of the Galois System for the graph K_d^n reducing it to a balls and bins problem [34]: with some approximations we can imagine that we are throwing m balls (the processors) uniformly at random in $s = n/(d+1)$ bins (the connected/conflicting components) and we are interested in the quantity of bins that have at least one ball (working processors). In the real case the probability of accessing a connected component decreases with the number of past accesses, hence what we are going to obtain here is an upper bound of the conflict ratio which is more accurate when m is small.

We know that the distribution of balls in each bin is a binomial, good approximated by the Poisson distribution as m and s increase. Let $\lambda = m/s$, then the probability for a bin to be hit is

$$\mathrm{P}\left[\text{bin } i \text{ is hit}\right] = 1 - \left(1 - \frac{1}{s}\right)^m \simeq 1 - e^{-\lambda}$$

Let X_i be a random variable that is 1 when bin i is hit and 0 otherwise. We have that $\mathbb{E}[X_i] = 1 - e^{-\lambda}$ and, by the linearity of the expectation, the average number of bins hit is

$$\mathbb{E}\left[\sum_i X_i\right] = \sum_i \mathbb{E}[X_i] = s\left(1 - e^{-\lambda}\right)$$

Which brings as a bound for the conflict ratio

$$\bar{r}(m) \leq \frac{m - s\left(1 - e^{-\lambda}\right)}{m} = 1 - \frac{1 - e^{-\lambda}}{\lambda}$$

□

Example. *Applying Thm. 7.1.5 we are sure that using $s/2$ processors we will have at most a conflict ratio of $\bar{r} = 21.3\%$.*

7.2 Controlling m

In Fig. 7.1 we can see some $\bar{r}$ plots of the worst case approximation, together with curves obtained from simulations, for some random graphs in which the edges have been chosen uniformly at random until reaching the desired average degree d (a class of graph more likely to represent the graphs Galois deals with in many applications). It is clear from the plots that the $\bar{r}$ function is almost linear in m for the values we are interested in and hence we can try end exploit this characteristic to obtain a choice of m_t that quickly converges to μ.

Recurrence A:

$$m_{t+1}^A = \frac{\rho}{r_t} m_t$$

Another interesting recurrence, which has a slower convergence but it is less susceptible to noise (the variance that makes r_t realizations different from $\bar{r}_t$), is the following:

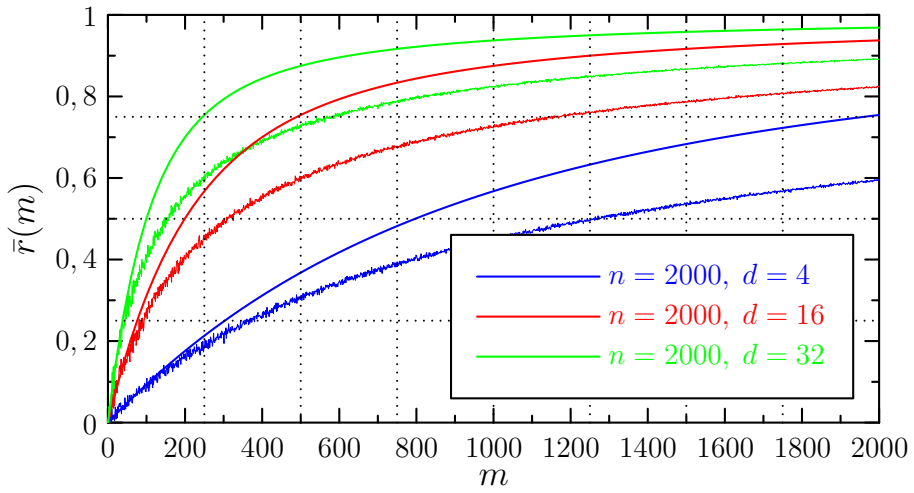


Figure 7.1: A plot of $\bar{r}(m)$ for some random conflicts-graphs (edges chosen uniformly at random until average degree d is reached) along with the worst case bound. Note that for $\bar{r} < 30\%$ the functions are well approximated by straight lines.

Recurrence B:

$$m^B_{t+1} = (1 - r_t + \rho)m_t$$

7.2.1 The Control Algorithm

A first problem with these recurrences is that r_t has a big variance, especially when m is small, so to avoid continually oscillating values it is better to apply the changes every T steps, using the averaged values obtained in these intervals to reduce the variance. To further reduce oscillations we can decide to apply a change only if the observed r_t is sufficiently different from ρ (e.g. more than 6%), thus avoiding small variations in the steady state, which interfere with locality exploitation because of the data moving from one processor to another.

A second problem that must be considered is that for small values of m the variance is much bigger, so it is better to tune separately this case using different parameters.

The hybrid algorithm proposed (see Algorithm 4 and Fig. 7.2) uses Recurrence A if the difference between r and ρ is big, Recurrence B if it is smaller and it does not change m if the difference is negligible. All the decisions are made using values averaged for T steps. In this way we can exploit the quick convergence of Recurrence A and use Recurrence B for a finer tuning of the control.

Algorithm 4: Pseudo-code of the proposed hybrid control algorithm

```
   // Tunable parameters
1  m_0 = 2;      m_max = 1024;      m_min = 2;
2  T = 4;        r_min = 3%         α_0 = 25%;      α_1 = 6%;
   // Variables
3  m ← m_0;
4  r ← 0;
5  t ← 0;
   // Main loop
6  while nodes to elaborate ≠ 0 do
7      t ← t + 1;
8      if m > m_max then m ← m_max;
9      else if m < m_min then m ← m_min;
10     Launch Galois with m nodes;
11     r ← r + new conflict ratio;
12     if (t mod T) = T − 1 then
13         r ← r/T;
           α ← |1 − r/ρ|;
14
15         if α > α_0 then
16             if r < r_min then r ← r_min;
               m ← ⌈(ρ/r) m⌉;
17
18         else if α > α_1 then
19             m ← ⌈(1 − r + ρ) m⌉;
20
```

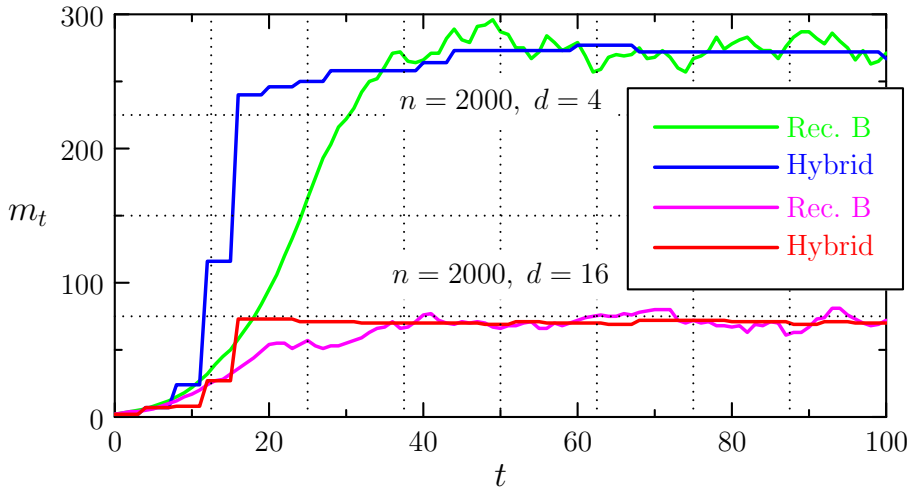


Figure 7.2: Comparison between two realizations of the hybrid algorithm and one the only uses Recurrence B, for two different random graphs. The hybrid version has different parameters for m greater or smaller than 20. ρ was chosen to be 20%. The proposed algorithm proves to be both quick in convergence and stable.

CHAPTER 8

Conclusions and Future Work

This thesis shows that (optimal) control theory offers a well-established, rigorous framework to study the computer systems optimization problems, as seen in the two case studies on page replacement and processor allocation. We conjecture that many other problems of resource management in computing systems with dynamic resources and workload are amenable to the same approach. However, even if successfully cast in control terms, these problems will generally require novel ideas to be solved, because the underlying dynamical systems are significantly different from those that have classically received the most attention, such the almost thoroughly investigated linear systems with quadratic cost. Investigating how to control computing systems with rich combinatorial structure is likely to open new perspectives for control theory as well as for computer science.

Focusing on the replacement problem, a number of interesting and challenging issue remain open, such as finding bias-optimal policies for arbitrary buffer size and arbitrary stack-depth distributions. It is well known that not all real workloads are accurately described by the LRU Stack Model: optimal policies under different models deserve to be explored. Within the LRUSM, we have considered policy design assuming a known stack-depth distribution: what performance guarantees can be achieved if the distribution is not known a priori is yet another intriguing question.

As for the Galois System a useful variation of the way we have modeled it is obtained if we assign different execution time to successful and aborted processes (as happens in some applications) and try to maximize the parallelism achieving a target conflict time ratio. Under this new model the linear

approximation used in the control algorithm could be not realistic, and some stronger properties of $\bar{r}$ may be required to obtain fast convergence. Another effect that could be taken in account in the modeling is that rolling-back a process often requires some computation time and then, keeping a low, constant conflict ratio has the positive side effect of speeding up the application execution, besides avoiding CPU time wasting. The control mechanism is now in implementation phase by the Galois group at the University of Texas at Austin, and so the model will soon benefit from interactions with the system developers.

Bibliography

[1] Aggarwal, A., Alpern, B., Chandra, A. K., and Snir, M. A model for hierarchical memory. In *STOC* (1987), ACM, pp. 305–314.

[2] Allen, R., and Kennedy, K. *Optimizing compilers for modern architectures: a dependence-based approach*. Morgan Kaufmann, 2002.

[3] Alon, N., and Spencer, J. *The probabilistic method*. Wiley-Interscience, 2000.

[4] An, P., Jula, A., Rus, S., Saunders, S., Smith, T. G., Tanase, G., Thomas, N., Amato, N. M., and Rauchwerger, L. Stapl: An adaptive, generic parallel C++ library. In *LCPC* (2001), H. G. Dietz, Ed., vol. 2624 of *Lecture Notes in Computer Science*, Springer, pp. 193–208.

[5] Arapostathis, A., Borkar, V. S., Fernández-Gaucherand, E., Ghosh, M. K., and Marcus, S. I. Discrete-time controlled markov processes with average cost criterion: a survey. *SIAM J. Control and Optimization 31*, 2 (March 1993), 282–344.

[6] Barnes, J., and Hut, P. A hierarchical 0 (N log iV) force-calculation algorithm. *Nature 324* (1986), 4.

[7] Belady, L. A. A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal 5*, 2 (1966), 78–101.

[8] Bernholt, T., Eisenbrand, F., and Hofmeister, T. A geometric framework for solving subsequence problems in computational biology efficiently. In *SCG '07: Proceedings of the twenty-third annual symposium on Computational geometry* (New York, NY, USA, 2007), ACM, pp. 310–318.

[9] Bertsekas, D. P. *Dynamic Programming and Optimal Control.* Athena Scientific, 2000.

[10] Bilardi, G., and Preparata, F. P. Horizons of parallel computation. *J. Parallel Distrib. Comput. 27*, 2 (1995), 172–182.

[11] Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. *ScaLAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[12] Blackwell, D. Discrete dynamic programming. *The Annals of Mathematical Statistics* (1962), 719–726.

[13] Braunstein, A., Mézard, M., and Zecchina, R. Survey propagation: An algorithm for satisfiability. *Random Struct. Algorithms 27*, 2 (2005), 201–226.

[14] Eppstein, D. Spanning trees and spanners. In *Handbook of Computational Geometry*, J. Sack and J. Urrutia, Eds. Elsevier, 2000, pp. 425–461.

[15] Fernández, E. B., Lang, T., and Wood, C. Effect of replacement algorithms on a paged buffer database system. *IBM J. Res. Dev. 22*, 2 (1978), 185–196.

[16] Fotheringham, J. Dynamic storage allocation in the atlas computer, including an automatic use of a backing store. *Commun. ACM 4*, 10 (1961), 435–436.

[17] Franaszek, P. A., and Wagner, T. J. Some distribution-free aspects of paging algorithm performance. *J. ACM 21*, 1 (1974), 31–39.

[18] Frederickson, G. N., and Johnson, D. B. The complexity of selection and ranking in x+y and matrices with sorted columns. *J. Comput. Syst. Sci. 24*, 2 (1982), 197–208.

[19] Frigo, M., and Johnson, S. G. The design and implementation of FFTW3. *Proceedings of the IEEE 93*, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[20] Frigo, M., Leiserson, C. E., and Randall, K. H. The implementation of the Cilk-5 multithreaded language. In *PLDI* (1998), pp. 212–223.

[21] Graham, R. L. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters 1*, 4 (1972), 132–133.

[22] Guibas, L. J., Knuth, D. E., and Sharir, M. Randomized incremental construction of delaunay and voronoi diagrams. *Algorithmica 7*, 4 (1992), 381–413.

[23] Hennessy, J. L., and Patterson, D. A. *Computer architecture: a quantitative approach*. Morgan Kaufmann, 2006.

[24] Jensen, J. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica 30*, 1 (1906), 175–193.

[25] Kalé, L. V., and Krishnan, S. Charm++: A portable concurrent object oriented system based on C++. In *OOPSLA* (1993), pp. 91–108.

[26] Karlin, A. R., Phillips, S. J., and Raghavan, P. Markov paging. *SIAM J. Comput. 30*, 3 (2000), 906–922.

[27] Kulkarni, M., Burtscher, M., Cascaval, C., and Pingali, K. Lonestar: A suite of parallel irregular programs. In *ISPASS* (2009), IEEE, pp. 65–76.

[28] Kulkarni, M., Pingali, K., Walter, B., Ramanarayanan, G., Bala, K., and Chew, L. P. Optimistic parallelism requires abstractions. *Commun. ACM 52*, 9 (2009), 89–97.

[29] Lewis, M. E., and Puterman, M. L. Bias optimality. In *Handbook of Markov Decision Processes: Methods and Applications*, E. A. Feinberg and A. Shwartz, Eds. Springer, 2002, pp. 89–111 (Chap. 3).

[30] Lin, Y.-L., Jiang, T., and Chao, K.-M. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *J. Comput. Syst. Sci. 65*, 3 (2002), 570–586.

[31] Loève, M. *Probability Theory*. Springer-Verlag, New York, 1977.

[32] Mattson, R. L., Gecsei, J., Slutz, D. R., and Traiger, I. L. Evaluation techniques for storage hierarchies. *IBM Systems Journal 9*, 2 (1970), 78–117.

[33] Méndez-Lojo, M., Nguyen, D., Prountzos, D., Sui, X., Hassaan, M. A., Kulkarni, M., Burtscher, M., and Pingali, K. Structure-driven optimizations for amorphous data-parallel programs. In *PPOPP* (2010), R. Govindarajan, D. A. Padua, and M. W. Hall, Eds., ACM, pp. 3–14.

[34] Mitzenmacher, M., and Upfal, E. *Probability and computing: Randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005.

[35] Oden, P. H., and Shedler, G. S. A model of memory contention in a paging machine. *Commun. ACM 15*, 8 (1972), 761–771.

[36] Przybylski, S. A. *Cache and memory hierarchy design: a performance-directed approach.* Morgan Kaufmann, 1990.

[37] Püschel, M., Moura, J., Johnson, J., Padua, D., Veloso, M., Singer, B., Xiong, J., Franchetti, F., Gacic, A., Voronenko, Y., Chen, K., Johnson, R., and Rizzolo, N. Spiral: Code generation for dsp transforms. *Proceedings of the IEEE 93*, 2 (Feb. 2005), 232–275.

[38] Reinders, J. *Intel threading building blocks.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2007.

[39] Savage, J. E. *Models of computation: Exploring the power of computing.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1997.

[40] Silberschatz, A., Galvin, P. B., and Gagne, G. *Operating System Principles.* Wiley India Pvt. Ltd., 2005.

[41] Sleator, D. D., and Tarjan, R. E. Amortized efficiency of list update and paging rules. *Commun. ACM 28*, 2 (1985), 202–208.

[42] Spirn, J. R., and Denning, P. J. Experiments with program locality. In *AFIPS '72 (Fall, part I)* (New York, NY, USA, 1972), ACM, pp. 611–621.

[43] Stone, H. S., Turek, J., and Wolf, J. L. Optimal partitioning of cache memory. *IEEE Trans. Comput. 41*, 9 (1992), 1054–1068.

[44] Tan, P.-N., Steinbach, M., and Kumar, V. *Introduction to Data Mining.* Addison-Wesley, 2005.

[45] Thiébaut, D., Stone, H. S., and Wolf, J. L. Improving disk cache hit-ratios through cache partitioning. *IEEE Trans. Comput. 41*, 6 (1992), 665–676.

[46] Wolfe, M. J., Shanklin, C., and Ortega, L. *High performance compilers for parallel computing.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[47] Wood, C., Fernández, E. B., and Lang, T. Minimization of demand paging for the LRU stack model of program behavior. *IBM Los Angeles Scientific Center Reports G320*, 2689 (July 1977).

[48] Wood, C., Fernández, E. B., and Lang, T. Minimization of demand paging for the LRU stack model of program behavior. *Inf. Process. Lett. 16*, 2 (February 1983), 99–104.

[49] Young, N. E. The k-server dual and loose competitiveness for paging. *Algorithmica 11*, 6 (1994), 525–541.