CS–1995–08

Adaptive Disk Spindown via Optimal Rent-to-Buy in Probabilistic Environments[1]

P. Krishnan
Brown University
Philip M. Long
Duke University
Jeffrey S. Vitter
Duke University

Department of Computer Science

Duke University

Durham, North Carolina 27708–0129

March, 1995

Adaptive Disk Spindown via Optimal Rent-to-Buy in Probabilistic Environments*

P. Krishnan† — Dept. of Computer Science, Brown University

Philip M. Long‡ — Dept. of Computer Science, Duke University

Jeffrey Scott Vitter§ — Dept. of Computer Science, Duke University

Abstract

In the single rent-to-buy decision problem, without a priori knowledge of the amount of time a resource will be used we need to decide when to buy the resource, given that we can rent the resource for \$1 per unit time or buy it once and for all for \$$c$. In this paper we study algorithms that make a sequence of single rent-to-buy decisions, using the assumption that the resource use times are independently drawn from an unknown probability distribution. Our study of this rent-to-buy problem is motivated by important systems applications, specifically, problems arising from deciding when to spindown disks to conserve energy in mobile computers [DKM, LKH, MDK], thread blocking decisions during lock acquisition in multiprocessor applications [KLM], and virtual circuit holding times in IP-over-ATM networks [KLP, SaK].

We develop a provably optimal and computationally efficient algorithm for the rent-to-buy problem and evaluate its practical merit for the disk spindown scenario via simulation studies. Our algorithm uses $O(\sqrt{t})$ time and space, and its expected cost for the tth resource use converges to optimal as $O(\sqrt{\log t/t})$, for any bounded probability distribution on the resource use times. Alternatively, using $O(1)$ time and space, the algorithm almost converges to optimal.

We describe the results of simulating our algorithm for the disk spindown problem using disk access traces obtained from an HP workstation environment. We introduce the natural notion of *effective cost* which merges the effects of energy conservation and response time performance into one metric based on a user specified parameter a, the relative importance of response time to energy conservation. (The buy cost c varies linearly with a.) We observe that by varying a, we can model the tradeoff between power and response time well. We also show that our algorithm is best in terms of effective cost for almost all values of a, saving effective cost by 6–25% over the optimal online algorithm in the competitive model (i.e., the 2-competitive algorithm that spins down the disk after waiting c seconds). In addition, for small values of a (corresponding to when saving energy is critical), our algorithm when compared against the optimal online algorithm in the competitive model reduces excess energy by 17–60%, and when compared against the 5 second threshold reduces excess energy by 6–42%.

---

*U.S. Mail contact for the authors: Department of Computer Science, Box 90129, Duke University, Durham, NC 27708–0129.

†Support was provided in part by an IBM Fellowship, by NSF research grant CCR–9007851, by Army Research Office grant DAAH04–93–G–0076, and by Air Force Office of Scientific Research grant F49620–94–1–0217. The author is currently visiting Duke University. Email pk@cs.duke.edu.

‡Supported in part by Air Force Office of Scientific Research grant F49620–92–J–0515 and F49620–94–1–0217. Email plong@cs.duke.edu.

§Supported in part by National Science Foundation research grant CCR–9007851, by Air Force Office of Scientific Research grants F49620–92–J–0515 and F49620–94–1–0217 and by a Universities Space Research Association/CESDIS associate membership. Email jsv@cs.duke.edu.

1 Introduction

The *single rent-to-buy decision* problem can be described as follows: we need a resource for an unknown amount of time, and we have the option to rent it for \$1 per unit time, or to buy it once and for all for \$$c$. For how long do we rent the resource before buying it? The best algorithm with full prior knowledge of how long the resource will be needed (an offline algorithm) will buy the resource immediately if the resource will be needed for at least c time units and rent otherwise. An online algorithm (i.e., one without *a priori* knowledge of how long the resource will be needed) that rents the resource for c units of time and then buys it incurs a cost of at most 2 times the cost of the best offline algorithm. This competitive factor[1] of 2 is the best possible (for deterministic algorithms) in the worst case [KMM]. If we know of a probability distribution on the time the resource is needed, we can usually find a rent-to-buy strategy whose expected cost is substantially less than that of the online algorithm that waits c time units before buying.

In this paper we are interested in the rent-to-buy problem described above with two important additional features motivated by practical applications. Many interesting systems problems can be modeled well by a *sequence* of single rent-to-buy problems. To solve the tth single rent-to-buy problem (or the tth *round*), the online algorithm can use what it has learned from the previous $t-1$ rounds. (The online algorithm that waits for c time before buying in each round is still within a factor of 2 of the best possible.) We call this the *sequential rent-to-buy* problem, or just the *rent-to-buy* problem. In these real-life situations we can assume that the time for which the resource is needed in each round is drawn from a probability distribution. However, it is unreasonable to assume that the distribution is known a priori. We now describe three interesting problems modeled by a sequence of rent-to-buy decisions.

Energy conservation is an important issue in mobile computing. Portable computers run on battery power and can function for only a few hours before draining their batteries. Current techniques for conserving energy are based on shutting down components of the system after reasonably long periods of inactivity. Recent studies show that the disk sub-system on notebook computers is a major consumer of energy [DKM, LKH, MDK]. Most disks used for portable computers (e.g., the small, light-weight Kittyhawk from Hewlett Packard [Hew]) have multiple energy states. Conceptually, the disk can be thought of as having two states: the *spinning* state in which the disk can access data but consumes a lot of energy and a *spundown* state in which the disk consumes effectively no energy but cannot access data.[2] Spinning down a disk and spinning it up consumes a fixed amount of energy and time (and also produces wear and tear on the disk). During periods of inactivity, the disk can be spundown to conserve energy at the expense of increased latency for the next request. The *disk spindown problem* is to decide when to spindown the disk so as to conserve energy, with acceptable latency.

The disk spindown scenario can be modeled as a rent-to-buy problem as follows. A round is the time between any two requests for data on the disk. For each round, we need to solve the disk spindown problem. Keeping the disk spinning is viewed as renting, since energy is continuously expended to keep the disk spinning. Spinning down the disk is viewed as a buy, since the energy to spindown the disk and spin it back up upon the next request is independent of the remaining amount of time until the next disk access. The cost of the increased latency in serving the next disk access can also be integrated into the cost of the buy, if the algorithm is given as an input the

[1] A k-competitive algorithm incurs a cost of at most $O(1)$ plus k times the cost of the optimal offline algorithm.

[2] In general, the disks provide more than just two power management states, but only one state, the fully spinning state, allows access to data.

relative importance of conserving energy and responding quickly to disk accesses. (This is discussed in detail in Section 6.) Based on observations of disk access patterns in workstation environments [RuW], the times between accesses to disk (which define the rounds) can be assumed to be generated by a probability distribution. The disk spindown problem will be our main motivating application for this study.

Another interesting and important problem from multiprocessor applications, the *spin/block problem*, involves threads trying to acquire locks to protect access to shared data [KLM]. A round is defined by a thread requesting locked data and eventually acquiring the lock. In a round, the system can have the thread wait (or *spin*) until the lock is free, incurring a fixed cost per unit time for wasted processor cycles, or block and incur a higher context switch overhead. The spinning can thus be viewed as renting, and a block can be viewed as a buy. In this situation too, practical studies suggest that lock-waiting times can be assumed to obey some unknown but time-invariant probability distribution [KLM].

Deciding virtual circuit holding times in IP-over-ATM networks is another scenario modeled by the rent-to-buy framework [SaK]. When carrying Internet protocol (IP) traffic over an Asynchronous Transfer Mode (ATM) network, a virtual circuit is opened upon the arrival of an IP datagram, and the ATM adaptation layer has to decide how long to hold a virtual circuit open. There are many possible pricing policies for virtual circuit holding times. As described in [SaK, Section 5], in future ATM networks, it is expected that a large number of virtual circuits could be held open by paying a charge per unit time to keep the circuit open. Keeping the virtual circuit open can be thought of as a "rent" while closing it can be considered a "buy." The inter-arrival time of packets on a circuit (i.e., the resource use times in the rent-to-buy model) can be modeled as being drawn independently from a probability distribution [LPR, SaK].

An algorithm for the sequential rent-to-buy problem can be visualized in two ways. In any round, the algorithm can be thought of as making sequential binary decisions of "should I buy now?" Alternatively, we can think of the algorithm as setting a threshold or *cutoff* on the cost it is willing to accrue before buying, and behaving according to the cutoff. These two views are trivially equivalent; we adopt the second for convenience. There are two important requirements of any good online algorithm for the rent-to-buy problem: the algorithm should produce good cutoffs, and it should use minimal space and time to output its cutoffs. In this paper we develop online algorithms for the rent-to-buy problem in probabilistic environments, assuming that the resource use times are independently randomly drawn from a fixed but unknown probability distribution.

The most straightforward solution to the problem [KMM] is to store all past resource use times, and use that cutoff b for the current round which would have had the lowest total cost had we used it in the past. Straightforward application of results of Vapnik [Vap] implies that the expected rent-to-buy cost of this strategy converges to that of the best fixed cutoff. One can easily see that the cutoff b at any given time falls on (actually, near) one of the past resource use times; however, even taking this into account, this solution is computationally expensive. For the tth round, this solution would need space and time proportional to $O(t)$, and this is unacceptable in system environments.

In this paper, we develop an algorithm L for the rent-to-buy problem which, for arbitrary probability distributions with support on $[0, M]$, converges to optimal; i.e., the cost of the algorithm converges to the cost of the best algorithm with full prior knowledge of the distribution. More importantly, for the tth round that lasts x_t time, the algorithm uses $O(c\sqrt{t})$ space, generates its cutoffs in $O(1)$ time, and uses $O((\min\{x_t, c\})\sqrt{t}+\log(ct))$ time to update its data structures. Alternatively, our algorithm can be adapted to work in a situation when the space it can use is limited. Presented with $O(s)$ space, our algorithm L_s uses $O(1)$ time to generate cutoffs, $O((\min\{x_t, c\})s + \log(cs))$

time to update its structures, and almost converges to optimal, being away from optimal by an additive factor of $O(\min\{M, c\}/s)$. The $O(x_t)$ component of the time used in updating the data structure can be done "on the fly" as the round is progressing. For example, in the disk spindown scenario, let the tth idle time at disk be $z < c$ seconds. Before the idle period starts, algorithm L_s outputs its recommended spindown threshold using $O(1)$ time, and updates its data structure in $O(zs + \log(ct))$ time. The updates corresponding to the "zs" term can be done while the disk is waiting for the next access.

Most practical situations are well-modeled by bounded distributions. For example, in the disk spindown scenario, any reasonable algorithm will spin down the disk after a few minutes (say, 30 minutes) since the last access. Therefore, all idle times at disk greater than 30 minutes are practically equivalent, and can be assumed to be 30 minutes without loss of generality, resulting in a distribution with bounded support.

Simulations of our algorithm on disk access traces obtained from HP show that by giving a suitable value of c to our algorithm, we effectively trade power for response time (latency). In Section 6 we introduce the natural notions of excess energy and effective cost. The "excess energy" discounts from the total energy the portion that *every* algorithm would have to spend; the effective cost is a measure that merges the effects of energy conservation and response time performance into one metric based on a user specified parameter a, the relative importance of response time to energy conservation. (The buy cost c varies linearly with a.) We show that our algorithm L is best amongst the online algorithms considered in terms of effective cost for almost all values of a, saving effective cost by 6–25% over the optimal online algorithm in the competitive model (i.e., the 2-competitive algorithm that spins down the disk after waiting c seconds). In addition, for small values of a (corresponding to when saving energy is critical), our algorithm when compared against the 2-competitive algorithm reduces excess energy by 17–60%, and when compared against the 5 second threshold, it reduced excess energy by 6–42%.

1.1 Related Work

The single rent-to-buy problem has been studied in the worst-case setting and efficient deterministic and randomized algorithms have been developed for the problem by Karlin et al. [KMM]. In particular, 2-competitive deterministic algorithms and $e/(e-1)$-competitive randomized algorithms have been developed. In [KMM] it was claimed that there is an adaptive algorithm achieving a competitive ratio approaching $e/(e-1)$ on input sequences generated according to any time invariant probability distribution. However, their technique as stated is computationally inefficient.

For the disk spindown problem, current mobile computers spin disks down after about five minutes of inactivity. In [DKM, LKH], the authors propose a more aggressive spindown policy, and support their proposal by simulation studies on workstation and notebook traces. The studies suggest that the gain in energy often overshadows the loss in response time. In [DKM], the comparison of fixed-threshold strategies is made against optimal offline algorithms. The authors also mention trying out predictive disk spindown policies. Adaptive spindown policies that continually change the spindown threshold based on perceived inconvenience to the user are studied in [DKB]. In [Gre], Greenawalt looks at the disk spindown problem assuming a Poisson arrival of requests at disk, and studies disk spindown and reliability issues.

Karlin et al. in [KLM] have studied the spin/block problem empirically, evaluating different spin/block strategies including fixed-threshold and adaptive strategies. The virtual circuit problem has been empirically studied by Saran et al. [SaK], where they propose a Least Recently Used (LRU)-based holding time policy as performing well in their studies. The first LRU-based holding time policy they study is the 2-competitive algorithm described earlier in this paper, and their second holding time policy involves estimating the mean inter-reference interval with exponential averaging. In [KLP], Keshav et al. empirically study an adaptive policy for the virtual circuit problem that tries to estimate the distribution of inter-arrival times by keeping a histogram of observed inter-arrival times grouped into fixed size buckets.

In Section 2, we describe the main analytical results of the paper. We present algorithm A_ϵ in Section 3; algorithm A_ϵ lies at the heart of our optimal rent-to-buy algorithms, L and L_s. We analyze algorithm A_ϵ for space used, computational time, and convergence rate in Section 4. We describe how algorithm A_ϵ can be used to get algorithms L, L_s in Section 5. We explain in Section 6 how the disk spindown problem can be modeled in the rent-to-buy framework, when the user is concerned about energy conservation and response time performance. We present our experimental results in Section 7 and conclude in Section 8.

2 Definitions and Main Analytical Results

We denote the reals by $\mathbb{R}$, the nonnegative reals by $\mathbb{R}^+$, and the positive integers by $\mathbb{N}$. An *online rent-to-buy algorithm* is given the relative cost $c \geq 1$ of buying. It works in rounds, where in the tth round, it first formulates a cutoff on the amount of time it will wait before buying, and then gets the tth resource use time. A rent-to-buy algorithm defines a mapping from $\cup_{n \in \mathbb{N}} (\mathbb{R}^+)^n$ (the past resource use times) to $\mathbb{R}^+$ (the cutoff generated). In other words, $A(x_1, x_2, \ldots, x_t)$ is the cutoff generated by algorithm A in the $(t+1)$st round, when the previous resource use times were $x_1, \; x_2, \ldots, x_t$. If the resource use time in any round is x, then the cost of choosing cutoff b is

$$\mathrm{cost}_c(x, b) = \begin{cases} x & \text{if } x \leq b \\ b + c & \text{otherwise.} \end{cases}$$

For the disk spindown problem, the resource use time in round t corresponds to the tth idle time at disk, and a cutoff is a spindown threshold.

Our first main result is an algorithm L that approaches optimal and is efficient in terms of the space and time it uses.

Theorem 1 *For any $c > 1$, $M > 1$, there is a rent-to-buy algorithm L that on round t with resource use time x_t,*

- *uses $O(c\sqrt{t})$ space*
- *outputs its choice of cutoff in $O(1)$ time, and updates its data structures in $O((\min\{x_t, c\})\sqrt{t} + \log(ct))$ time, and*
- *incurs a cost that approaches optimal: there exists k such that for any distribution D on $[0, M]$, for all large enough $t \in \mathbb{N}$,*

$$\mathbf{E}_{\vec{x} \in D^t}(\mathrm{cost}_c(x_t, L(x_1, ..., x_{t-1}))) \leq \inf_a \mathbf{E}_{z \in D}(\mathrm{cost}_c(z, a)) + k\sqrt{\frac{\ln t}{t}}.$$

Note that in Theorem 1, the same k can used for any distribution with support on $[0, M]$. Further, the time and space bounds are independent of D as well. It is easy to adapt algorithm L

to get algorithm L' that successively increases its estimate of M, and converges to optimal for any distribution. However, the convergence rate of algorithm L' would depend on the distribution.

In many practical situations, we would like to fix the amount of space and time used by our algorithm while converging approximately, rather than exactly, to optimal. Algorithm L_s, a restricted space version of Algorithm L, can be used in this scenario.

Theorem 2 *When presented with $s > k\ln^2(M+c)\ln\ln(M+c)$ bytes of space, where k is a constant independent of M and c, for the tth round, algorithm L_s outputs its choice of cutoff in $O(1)$ time, updates its data structures in $O((\min\{x_t, c\})s + \log(cs))$ time, and for any probability distribution D on $[0, M]$ for all large enough $t \in \mathbb{N}$, converges approximately to optimal:*

$$\mathbf{E}_{\vec{x}\in D^t}(\mathrm{cost}_c(x_t, L_s(x_1, ..., x_{t-1}))) \leq \inf_a \mathbf{E}_{z\in D}(\mathrm{cost}_c(z, a)) + O\left(\frac{\min\{c, M\}}{s}\right).$$

We simulated our algorithms for the disk spindown problem using disk access traces obtained from an HP workstation environment. Our simulation results are described in Section 7.

One obvious approach to attack the rent-to-buy problem in probabilistic environments is to learn the distribution on times for the rounds, calculate the optimal cutoff for the estimated distribution, and output that cutoff for each round. This is unacceptable from the computational standpoint. In our algorithms, we bypass the estimation of the distribution, directly estimating the efficacy of different cutoff points. The analysis is complicated, however, by the fact that there are infinitely many cutoff points to evaluate at any given time on the basis of a finite number of samples from the distribution. We show for the rent-to-buy problem that to get a good solution, it is sufficient to consider a small finite set of possible cutoff points. The appropriate choice of this set depends on the distribution, and is done using the information gained in early rounds. We call this basic strategy that chooses the appropriate set of possible cutoffs and evaluates them to determine the best cutoff to use in any round as algorithm A_ϵ.

Our algorithm L is based on algorithm A_ϵ. It chooses from among successively larger finite sets of possible cutoff points to converge to optimal. A tree data structure, which is modified dynamically, is used to store the estimated quality of each considered cutoff point. Algorithm L_s sets appropriate parameters based on the available space s, and uses algorithm A_ϵ to converge approximately to optimal.

We first describe algorithm A_ϵ which lies at the heart of our optimal algorithms L and L_s.

3 The Main Idea: Algorithm A_ϵ

Algorithm A_ϵ takes as parameters ϵ and M, and attempts to achieve an expected cost on a given round which is at most ϵ greater than the expected cost incurred by the optimal cutoff. In keeping with the terminology commonly used with learning algorithms, we also call a resource use time an "example." Our algorithms estimate optimal cutoffs based on past resource use times; in other words, they estimate optimal cutoffs based on the examples they have seen.

Algorithm A_ϵ works in two stages. In the *first stage*, it uses a small number of examples to generate a small number of candidate cutoffs. (For the small number of rounds that constitute the first stage, the algorithm chooses an arbitrary cutoff, say buying immediately.) It fixes these candidate cutoffs and then starts its *second stage*. For the tth round in the second stage, it evaluates the candidate cutoffs on the past $t-1$ examples, and chooses the cutoff with minimum total cost. The important point is that these small number of candidate cutoffs when generated carefully are sufficient to achieve a small enough cost, as described in Section 4.2. Also, updating these cutoffs can be done efficiently, as described in Section 4.3. We call an ϵ such that $0 < \epsilon < 1/(\ln^2(M+c)\ln\ln(M+c))$ a *suitable* epsilon; for technical reasons, we assume in our discussions that ϵ is suitable.

Note that the problem is trivial if $c \geq M$, since no reasonable algorithm would ever buy in this case; the case of interest is when $c \ll M$.

3.1 First Stage

In the first stage, algorithm A_ϵ generates candidate cutoffs $b_0,\ b_1, \ldots,\ b_v$ by partitioning $[0, M]$ into v intervals. Intuitively, to be accurate in its estimations in the second phase, algorithm A_ϵ wants these candidate cutoffs to be close in one of two senses: either that the probability of a point falling between them is not too large, or in absolute distance. However, for computational efficiency, we do not want too many candidate cutoffs. Hence, algorithm A_ϵ attempts to partition $[0, M]$ into $v \leq \lceil 4c/\epsilon \rceil$ intervals, such that

1. each interval is at least $\epsilon/2$ in length, and

2. if an interval has length $> \epsilon/2$, then the interior of the interval has probability at most $\epsilon/2c$.

The endpoints of the intervals define the candidate cutoffs.

We say that an interval satisfies the *computational criterion* if it is at least $\epsilon/2$ in length, and that it satisfies the *density criterion* if the probability of the interval is at most $\epsilon/2c$. (In other words, at the end of the first stage, algorithm A_ϵ ensures that every interval satisfies the computational criterion, and intervals of length greater than $\epsilon/2$ satisfy the density criterion.) Conceptually, we can think of algorithm A_ϵ as generating v' intervals that each satisfy the density criterion, and then moving the potential cutoffs apart (discarding intervals of size 0) to get $v \leq v'$ intervals such that the computational criterion holds for each interval. As a result of the VC theory [BEH, VaC], it is easy to partition $[0, M]$ into v intervals satisfying the density criterion with high probability, by storing $\eta = \Theta(v \ln v)$ examples, and calling a procedure `generate_cutoffs(`w, η, σ`)` on $[0, M]$. The procedure `generate_cutoffs` breaks a specified interval into w intervals by taking a set σ of η examples, and ensuring that in any interval we have η/w examples from σ. (The procedure `generate_cutoffs` can be implemented by sorting σ to get κ and iteratively moving through η/w examples in κ to define the intervals.)

Algorithm A_ϵ implements its first stage in a space efficient manner by storing at most $O(v)$ examples at any time. It performs the first stage in three *phases*. In the first phase, algorithm A_ϵ partitions $[0, M]$ "roughly" into B big intervals, and in the second phase it refines these big intervals one by one into approximately v'/B intervals each. While refining a specific big interval, algorithm A_ϵ discards examples that do not fall in the big interval. In the third phase, algorithm A_ϵ moves potential candidate cutoffs apart to ensure that the computational criterion is met.

Formally, algorithm A_ϵ works as follows. Let $\delta = \epsilon/(4(c+M))$, and let the array σ store the examples being retained by algorithm A_ϵ. In the *first phase*, it divides the interval $[0, M]$ into $B = k\ln(1/\delta)/2$ *big intervals* (where k is a constant independent of ϵ, M, and is as defined in Lemma 2). It does this by collecting $\eta_1 = 4kB\ln(2B/\delta)$ examples and calling `generate_cutoffs(`B, η_1, σ`)` on interval $[0, M]$. The *second phase* consists of B subphases, where in the ith subphase, algorithm A_ϵ divides the ith big interval into $\lceil 4c/(B\epsilon) \rceil$ intervals. It does this by sampling at most $\eta_2' = 4B(\eta_2 + \ln(2B/\delta))$ examples, where $\eta_2 = 4kc\ln(4c/(\epsilon\delta))/(\epsilon B)$, and storing the first η_2 examples that fall within the ith big interval. Let $\eta_{2,i} \leq \eta_2$ be the number of examples stored in the ith subphase. Algorithm A_ϵ calls `generate_cutoffs`$(\lceil 4c/(B\epsilon)\rceil, \eta_{2,i}, \sigma)$ on the ith big interval. (We will see in Section 4.1 that $\eta_{2,i} = \eta_2$ with high probability.) At the end of the second phase, we are left with the required $v' \approx \lceil 4c/\epsilon \rceil$ intervals. In the third phase, algorithm A_ϵ ensures that the computational criterion is met. Let the ith interval at the end of the second phase be $[l'_i, r'_i)$. Algorithm A_ϵ sets $l_0 = 0$, $r_0 = \max(\epsilon/2, r'_0)$, and processes the intervals iteratively by setting $l_i = r_{i-1}$, and $r_i = \max(r'_i, l_i + \epsilon/2)$. The ith interval is defined to be $[l_i, r_i)$, and intervals such that $l_i = r_i$ are discarded. The total number of resulting intervals is $v \le \lceil 4c/\epsilon \rceil$.

The candidate cutoffs are defined to be $b_i = l_i$, $0 \le i < v$, $b_v = M$.

3.2 Second stage

In the second stage, algorithm A_ϵ repeatedly chooses the cutoff from among those in $\{b_0,\ b_1, ..., b_v\}$ that performed the best in the past. Formally, it formulates its tth cutoff in the second stage as follows. If $x_1,\ x_2, ..., x_{t-1}$ are the resource use times previously seen in the second stage, for all $i \in \mathbb{N}, 0 \le i \le u$, algorithm A_ϵ sets

$$q_i = \sum_{j=1}^{t-1} \mathrm{cost}_c(x_j, b_i).$$

It uses a b_i for which $q_i \le q_k$ for all $k \in \{0, ..., s\}$ as its cutoff for the tth round.

We now study the performance of algorithm A_ϵ in terms of space used, the convergence rate, and time required for updates.

4 Goodness of Algorithm A_ϵ

In Section 4.1, we see that algorithm A_ϵ can be implemented with $O(v)$ space, and generates good cutoffs with high probability. In Section 4.2 we see that the distance algorithm A_ϵ is away from optimal approaches ϵ as t gets large, and in Section 4.3 we see that in the second stage the strategies can be updated efficiently with a tree-based data structure.

4.1 Guarantees about the First Stage

Let $\delta = \epsilon/(4(c+M))$, $B = k\ln(1/\delta)/2$, $\eta_1 = 4kB\ln(2B/\delta)$, and $\eta_2 = 4kc\ln(4c/(\epsilon\delta))/(\epsilon B)$ be as defined in Section 3.1. From the discussion in Section 3.1, it follows that the space used by Algorithm A_ϵ in the first stage is bounded by the number of examples we use at any time plus the number of cutoffs we retain; i.e., the space used is bounded by $B+v+\max\{\eta_1, \eta_2\} = O(v) = O(c/\epsilon)$.

The operations in the third phase of the first stage ensure that every interval satisfies the computational criterion. We say that the first stage *fails* if at the end of the first stage there is an interval of length greater than $\epsilon/2$ not satisfying the density criterion. The event that the first stage fails is a subset of the event that at the end of the second phase, there is some interval that does not satisfy the density criterion.

Let ℓ_ϵ be the *total* number of examples we see in the first stage; i.e., all examples, including the ones we discard. We now see that the first stage fails with low probability (i.e., probability 2δ).

Lemma 1 *Let $\ell_\epsilon = \left\lceil kc\ln^2\left((c+M)/\epsilon\right)/\epsilon\right\rceil$ be the number of examples seen in the first stage (where k is as defined in Lemma 2), let $\delta = \epsilon/(4(c+M))$, and let E_1 be the event that the first stage fails. Then, for any ϵ that is suitable, $\mathbf{Pr}(E_1) \le \epsilon/(2(c+M))$.*

To prove the above lemma, we use a technique due to Kearns and Schapire [KeS]. Lemma 2 below follows immediately from the results of Blumer et al. [BEH] using the techniques of Vapnik and Chervonenkis [VaC]. Informally, Lemma 2 says that m points are enough to simultaneously estimate the probabilities of *every* interval.

Lemma 2 *Choose $0 < \alpha, \beta \leq 1/2, c \geq 1$, and a probability distribution D on $\mathbb{R}^+$. Then there exists $k > 0$ such that if $m = \left\lceil \frac{k}{\alpha}(\ln \frac{1}{\alpha} + \ln \frac{1}{\beta}) \right\rceil$ then*

$$\mathbf{Pr}_{\vec{x} \in D^m} \left(\exists a, b \ s.t. \ \mathbf{Pr}_D((a,b)) \geq 2\alpha \ and \ \frac{1}{m} |\{j : x_j \in (a,b)\}| \leq \alpha \right) \leq \beta$$

and

$$\mathbf{Pr}_{\vec{x} \in D^m} \left(\exists a, b \ s.t. \ \mathbf{Pr}_D((a,b)) \leq \alpha/2 \ and \ \frac{1}{m} |\{j : x_j \in (a,b)\}| \geq \alpha \right) \leq \beta.$$

The standard Chernoff bounds will be helpful to prove Lemma 1.

Lemma 3 (Chernoff) *For t independent Bernoulli trials each of which has a probability of success at least p, let $LE(p,t,r)$ denote the probability that there are at most r successes in the t trials. Then, for $0 < p < 1$, and $0 \leq q \leq p$,*

$$LE(p,t,qt) \leq e^{-(p-q)^2 t/2p}$$

Proof of Lemma 1: The value for ℓ_ϵ was obtained by assuming that the first phase requires us to look at $\eta_1 = 4kB\ln(2B/\delta)$ examples, and the ith subphase of the second phase requires us to look at a total of $\eta_2' = 4B(\eta_2 + \ln(2B/\delta))$ examples, where $\eta_2 = 4kc\ln(4c/(\epsilon\delta))/(\epsilon B)$, and $B = k\ln(1/\delta)/2$. We bound the probability of the first stage failing by the probability of the event that at the end of the second phase there is some interval that does not satisfy the density criterion.

We say that the first phase fails, if any big interval generated in the first phase has probability greater than $2/B$ or less than $1/2B$. We say that the ith subphase fails if any interval generated in the ith subphase has probability greater than $\epsilon/2c$; the second phase fails if for any i, the ith subphase fails. The lemma is proved if we can bound the probability of the first phase failing or any of the subphases failing by δ/B, since the net failure probability is then bounded by $(\delta/B)\cdot(B+1) \leq \epsilon/(2(c+M))$.

From Lemma 2, by setting $\alpha = 1/B$ and $\beta = \delta/(2B)$, we can easily verify that if we look at η_1 examples, the first phase fails with probability at most δ/B. We now assume that the first phase did not fail; i.e., the probability of any big interval is between $1/2B$ and $2/B$. We could fail in the ith subphase if we either do not get η_2 examples in the ith big interval, or if after using the η_2 examples we get an interval with probability $> \epsilon/2c$. From Lemma 3, by substituting $p = 1/(2B)$, $r = \eta_2$, and $t = \eta_2'$, we see that the probability that the number of examples that fall in the ith big interval is less than η_2 is at most $\delta/(2B)$. From Lemma 2, by setting $\alpha = \epsilon B/(4c)$ and $\beta = \delta/(2B)$, we see that the probability that the η_2 examples did not divide the ith big interval into subintervals with probability $\leq \epsilon/2c$ is at most $\delta/2B$. Hence the probability of the ith subphase failing is at most δ/B. □

4.2 Convergence of Algorithm A_ϵ

We have seen that the first stage works with high probability. The main result of this subsection is to bound the performance of A_ϵ.

Theorem 3 *Choose M, c such that $M > c \geq 1$. Choose any ϵ that is suitable, and let $\ell_\epsilon = \left\lceil kc\ln^2\left((c+M)/\epsilon\right)/\epsilon\right\rceil$ be the number of examples seen by algorithm A_ϵ in the first stage, where k is defined as in Lemma 1. There exists $k_1 > 0$ such that for sufficiently large $t \in \mathbb{N}$, for any distribution D on $[0, M]$,*

$$\begin{aligned}&\mathbf{E}_{(\vec{u},\vec{x})\in D^m\times D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})))\\ &\qquad\leq (\inf_a \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a))) + \epsilon + k_1(c+M)\sqrt{\frac{\ln((c+M)t/\epsilon)}{t}}.\end{aligned}$$

To prove the above theorem, we first show that if the first stage was successful, then one of the possible cutoffs b_j generated in the first stage is only $\epsilon/2$ away from optimal (Lemma 4). Intuitively, by choosing the cutoff with minimal cost in the second stage, we are close to b_j in cost. We then bound the error in expected cost resulting from the first stage failing and prove Theorem 3.

Lemma 4 *Choose $0 < \epsilon \leq 1/2$, $c \geq 1$, $s \in \mathbb{N}$, and a probability distribution D on $[0, M]$. Choose $0 = b_0 < b_1 < ... < b_s = M$. If for all $j \in \{1,...,s\}$, either $\mathbf{Pr}_D((b_{j-1}, b_j)) \leq \epsilon/2c$, or $b_j - b_{j-1} = \epsilon/2$, then there exists $i^* \in \{0,...,s\}$ such that*

$$\mathbf{E}_{z\in D}(\mathrm{cost}_c(z, b_{i^*})) \leq \inf_a \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a)) + \frac{\epsilon}{2}.$$

Proof: Intuitively, if the optimal cutoff lies between b_{j-1} and b_j, the way in which the candidate cutoffs were chosen ensures that the interval (b_{j-1}, b_j) is "small enough" (in probability or absolute size) so that one of b_{j-1} or b_j is close to optimal.

Assume without loss of generality that no b_i is exactly optimal; i.e., for all $\delta > 0$, there exists an $a^* \notin \{b_0,...,b_s\}$, such that $\mathrm{cost}_c(z, a^*) = \inf_a \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a)) + \delta$. Choose $\delta > 0$ and fix a^*, $b_{j-1} < a^* < b_j$. We now show that one of $i^* = j-1$ or $i^* = j$ satisfies the lemma.

Case 1. $\mathbf{Pr}(b_{j-1}, b_j) \leq \epsilon/2c$. In this case, we show that the lemma holds with $i^* = j-1$. If a resource use time z lies outside of the interval $[b_{j-1}, a^*)$, then the cutoff a^* incurs at least as much cost as the cutoff b_{j-1}, since $a^* > b_{j-1}$. If the resource use time $z \in (b_{j-1}, a^*]$, then the expected extra cost of cutoff b_{j-1} is at most $c \cdot \mathbf{Pr}_D((b_{j-1}, a^*)) \leq c\cdot(\epsilon/2c) \leq \epsilon/2$.

$$\begin{aligned}\mathbf{E}_{z\in D}(\mathrm{cost}_c(z, b_{j-1})) &\leq \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a^*) \mid z \notin [b_{j-1}, a^*)) \cdot \mathbf{Pr}_{z\in D}(z \notin [b_{j-1}, a^*))\\ &\quad + \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a^*) + c \mid z \in (b_{j-1}, a^*]) \cdot \mathbf{Pr}_D((b_{j-1}, a^*])\\ &\leq \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a^*)) + \epsilon/2 \qquad (\text{since } \mathbf{Pr}_D((b_{j-1}, b_j)) \leq \epsilon/2c)\\ &\leq \inf_a \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a)) + \delta + \epsilon/2.\end{aligned}$$

Case 2. $\mathbf{Pr}(b_{j-1}, b_j) > \epsilon/2c$. In this case, we show that the lemma holds with $i^* = j$. Note that $b_j - b_{j-1} = \epsilon/2$. For all $c > 1$ and all distributions D, $\mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a))$ viewed as a function of a is Lipschitz bounded in one direction in a sense. (This is in spite of the fact that this function of a has jump discontinuities in general.) That is, if $0 \leq a_1 < a_2$, then

$$\mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a_2)) - \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a_1)) \leq a_2 - a_1.$$

Hence,

$$\mathbf{E}_{z\in D}(\mathrm{cost}_c(z,b_j)) - \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a^*)) \leq b_j - a^* \leq b_j - b_{j-1} \leq \frac{\epsilon}{2},$$

which implies that

$$\mathbf{E}_{z\in D}(\mathrm{cost}_c(z,b_{j-1})) \le \inf_a \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,a)) + \delta + \epsilon/2.$$

Since $\delta > 0$ was chosen arbitrarily, this completes the proof. $\square$

The standard Hoeffding bounds will be useful to prove Theorem 3.

Lemma 5 (see [Pol]) *Choose $M > 0$, a probability distribution D on $[0, M]$, and $m \in \mathbb{N}$. Then*

$$\mathbf{Pr}_{\vec{x}\in D^m}\left(\left|\frac{1}{m}\sum_{i=1}^{m} x_i - \mathbf{E}_{u\in D}(u)\right| \ge \epsilon\right) \le 2e^{-2\epsilon^2 m/M^2}.$$

Proof of Theorem 3: Regardless of what happens in the first stage, for all $j \le s$ and for all $x \in \mathbb{R}^+$, we have $\mathrm{cost}_c(x, b_j) \le c + M$. Thus, applying Lemma 5, we get for each $j \le s$, $\alpha > 0$,

$$\mathbf{Pr}_{\vec{x}\in D^m}\left(\left|\frac{1}{t-1}\sum_{i=1}^{t-1}\mathrm{cost}_c(x_i,b_j) - \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,b_j))\right| \ge \alpha\right) \le 2e^{-2\alpha^2(t-1)/(c+M)^2}.$$

Approximating $s = \lceil 4c/\epsilon \rceil$ by $8c/\epsilon$, we get

$$\mathbf{Pr}_{\vec{x}\in D^m}\left(\exists (j \le s)\ s.t.\ \left|\frac{1}{t-1}\sum_{i=1}^{t-1}\mathrm{cost}_c(x_i,b_j) - \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,b_j))\right| \ge \alpha\right) \le \frac{16c}{\epsilon}e^{-2\alpha^2(t-1)/(c+M)^2}. \tag{1}$$

Let j^* be such that b_{j^*} is the cutoff amongst the candidates with minimum cost; i.e.,

$$\mathbf{E}_{z\in D}(\mathrm{cost}_c(z,b_{j^*})) = \min_j \mathbf{E}_{z\in D}(\mathrm{cost}_c(z,b_j)),$$

and let $\hat{j}^*$ be the index of the cutoff used by A_ϵ in the tth round. Recall that

$$\frac{1}{t-1}\sum_{i=1}^{t-1}\mathrm{cost}_c(x_i,b_{\hat{j}^*}) = \min_j\left\{\frac{1}{t-1}\sum_{i=1}^{t-1}\mathrm{cost}_c(x_i,b_j)\right\}.$$

Let E_1 be the event that the first stage was successful, i.e., for all intervals (b_{j-1}, b_j) generated in the first stage, $|b_j - b_{j-1}| = \epsilon/2$, or $\mathbf{Pr}_D((b_{j-1}, b_j)) < \epsilon/2c$. We have

$$\begin{aligned}
&\mathbf{E}_{(\vec{u},\vec{x})\in D^m\times D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})))\\
&\quad= \mathbf{E}_{(\vec{u},\vec{x})\in D^m\times D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})) \mid E_1)\cdot\mathbf{Pr}(E_1)\\
&\qquad+ \mathbf{E}_{(\vec{u},\vec{x})\in D^m\times D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})) \mid \neg E_1)\cdot\mathbf{Pr}(\neg E_1)\\
&\quad\le \mathbf{E}_{(\vec{u},\vec{x})\in D^m\times D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})) \mid E_1)\cdot\mathbf{Pr}(E_1)\\
&\qquad+ (c+M)\left(\frac{\epsilon}{2(c+M)}\right) \qquad \text{(Lemma 1)}\\
&\quad\le \mathbf{E}_{(\vec{u},\vec{x})\in D^m\times D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})) \mid E_1) + \tfrac{\epsilon}{2}.
\end{aligned} \tag{2}$$

Now, assume $u_1, ..., u_m$ make E_1 true. Fix $\alpha > 0$. Let E_2 be the event that all the estimates of $\mathbf{E}_{z\in D}(\mathrm{cost}_c(z, b_j))$ obtained through $x_1, ..., x_t$ are accurate to within α. Then

$$\begin{aligned}
&\mathbf{E}_{\vec{x}\in D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})))\\
&\quad= \mathbf{E}_{\vec{x}\in D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})) \mid E_2)\cdot\mathbf{Pr}(E_2)\\
&\qquad+ \mathbf{E}_{\vec{x}\in D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})) \mid \neg E_2)\cdot\mathbf{Pr}(\neg E_2)\\
&\quad\le \mathbf{E}_{\vec{x}\in D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1,...,u_m,x_1,...,x_{t-1})) \mid E_2)\\
&\qquad+\frac{16c(c+M)}{\epsilon}\exp\left(\frac{-2\alpha^2(t-1)}{(c+M)^2}\right),
\end{aligned} \tag{3}$$

by (1). By the triangle inequality, if $\mathbf{E}_{z\in D}(\mathrm{cost}_c(z, b_{\hat{j}^*})) \geq \mathbf{E}_{z\in D}(\mathrm{cost}_c(z, b_{j^*})) + 2\alpha$, then for either $v = j^*$ or $v = \hat{j}^*$,

$$\left|\mathbf{E}_{z\in D}(\mathrm{cost}_c(z, b_v)) - \frac{1}{t-1}\sum_{i=1}^{t-1} \mathrm{cost}_c(x_i, b_v)\right| \geq \alpha.$$

Thus, (3) and Lemma 4 imply that if E_1 is true, then

$$\begin{aligned}&\mathbf{E}_{\vec{x}\in D^t}(\mathrm{cost}_c(x_t, A_\epsilon(u_1, ..., u_m, x_1, ..., x_{t-1}))) \\ &\quad\leq (\inf_a \mathbf{E}_{z\in D}(\mathrm{cost}_c(z, a))) + \tfrac{\epsilon}{2} + 2\alpha + \frac{16c(c+M)}{\epsilon}\exp\left(\frac{-2\alpha^2(t-1)}{(c+M)^2}\right).\end{aligned}$$

Combining with (2) and setting $\alpha = 100(c+M)\sqrt{\ln((c+M)t/\epsilon)/t}$ completes the proof. □

4.3 Computation Time of Algorithm A_ϵ

We now describe how the predictions of A_ϵ are made efficiently. Let $\sigma_t = x_1, x_2, \ldots, x_{t-1}$ be the sequence formed by the first $t-1$ rounds in the second stage, where x_i, for $1 \leq i < t$, is the resource use time seen in round i. Recall from Section 3 that for the tth round, algorithm A_ϵ needs to output a strategy b_j that has minimum cost on the rounds in σ_t. Any updates to the data structures used by algorithm A_ϵ need to be made efficiently. We now describe a data structure maintained by algorithm A_ϵ that allows predictions to be output in $O(1)$ time and updates to be made in $O(\min\{x_t, c\}/\epsilon + \log(c/\epsilon))$ time. (Note that in problems of interest, $c \ll M$.)

Algorithm A_ϵ maintains the different candidate cutoffs as leaves of a balanced tree T. (See Figure 1.) We label the root of the tree by λ, and the leaves of the tree from left to right as $0 \ldots v$, such that the jth leaf corresponds to the cutoff b_j. (For simplicity, we use the name b_j for leaf j.) Let $T(x)$ be the subtree of T rooted at node x, and let $P(x)$ be the path from the root to (and including) node x. In particular, T is $T(\lambda)$.

With each (leaf and internal) node x, algorithm A_ϵ maintains three variables, *diff*(x), *min_cost*(x), and *min_cutoff*(x). The algorithm maintains the following invariants for all t before the tth round. (These invariants define the variables.) We refer to the total cost of an algorithm that repeatedly uses a given cutoff over a sequence of resource use times as the cost of that cutoff on the sequence. The cost of using cutoff b_j for σ_t is proportional to the sum of the *diff* values of the nodes in the path from the root to b_j, i.e., the cost of using cutoff b_j for σ_t is proportional to $\sum_{x\in P(b_j)} \mathit{diff}(x)$. The variable *min_cutoff*(x) is the cutoff b_j with minimum cost for σ_t amongst all cutoffs that are leaves of $T(x)$. The variable *min_cost*(x) is closely related to the cost of the best cutoff amongst the leaves of $T(x)$; in particular, it is the cost of the best cutoff amongst the leaves of $T(x)$ minus the sum of the *diff* values of the nodes in $P(\mathit{parent}(x))$. Formally, $\mathit{min\_cost}(x) = \min_{b_l\in T(x)}\{\sum_{1\leq i<t} \mathit{cost}(x_i, b_l)\} - \sum_{i\in P(\mathit{parent}(x))} \mathit{diff}(i)$. It is important to note that since two siblings in T have the same parent, the *min_cost* values at the two siblings can be directly compared to get the *min_cutoff* value at the parent.

The tree is initialized appropriately. After round $t-1$, algorithm A_ϵ outputs *min_cutoff*(λ) as its cutoff for the tth round. Let $b_j \leq x_t < b_{j+1}$. For the data structure to be consistent after request x_t (the tth round), the algorithm needs to increase the cost of each cutoff b_i for $0 \leq i \leq j$, by $b_i + c$ (which varies with i), and the cost of each cutoff b_i for which $i < m \leq s$, by x_t (which is independent of i). As shown in Figure 1, the data structure is kept consistent by adding $b_i + c$ to the *diff* value of each of the leaves $0 \ldots j$, and by adding x_t to the *diff* values of each right child of the nodes in $P(b_j)$ that is not itself in $P(b_j)$. (Notice that exactly one *diff* value in the path from

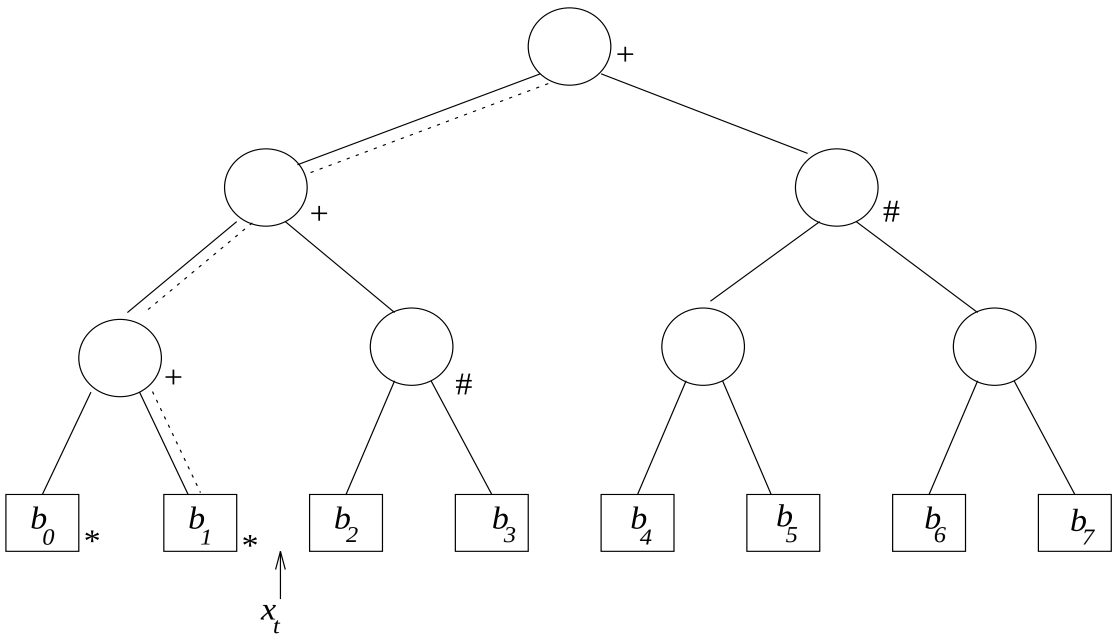


Figure 1: Snapshot of the data structure used by algorithm A_ϵ. In the situation depicted above, there are 8 candidate cutoffs labeled $b_0, \ldots, b_7$, appearing as leaves of the tree. The value x_t falls between b_1 and b_2. The path $P(b_1)$ is shown with dotted lines. The *diff* values of all nodes marked with a "*" are increased by the value of the cutoff at the node plus c. The *diff* values of the nodes marked with a "#" are increased by x_t. The *min_cutoff* and *min_cost* values of all marked nodes (whether marked with a "*" or "#" or "+") are updated.

each leaf to the root is updated.) Algorithm A_ϵ updates the *min_cutoff* and *min_cost* variables for the nodes whose *diff* values were changed and their ancestors. The *min_cost* values are updated using the relation $\mathit{min\_cost}(x) = \min\{\mathit{min\_cost}(\text{left\_child}(x)), \mathit{min\_cost}(\text{right\_child}(x))\} + \mathit{diff}(x)$. (The correctness of this update procedure follows by induction.) Also, $\mathit{min\_cutoff}(x)$ is updated to be the the *min_cutoff* of the child of x that has the smaller *min_cost*.

The number of leaves in the tree is $O(c/\epsilon)$. The time to update the *diff* values of the cutoffs b_i, $0 \leq i \leq j$ is $O(\min\{x_t, c\}/\epsilon)$, since each $[b_i, b_{i+1}]$ is at least $\epsilon/2$ in size. Updating the other *diff* values takes time proportional to the height of the tree, which is $O(\log(c/\epsilon))$. Hence, the amount of time to make the updates is $O((\min\{x_t, c\})/\epsilon + \log(c/\epsilon))$. The leaves $0 \ldots j$ and (most of) their ancestors can be updated online as time passes, with an extra $O(\log(c/\epsilon))$ processing required at the end.

5 Getting Algorithms L and L_s from Algorithm A_ϵ

In this section we prove Theorems 1 and 2 by developing our algorithms L and L_s.

5.1 Algorithm L

Our convergent algorithm L is obtained by running A_ϵ with continually decreasing ϵ. Clearly, if we start $A_{1/\sqrt{t}}$ sufficiently far back in the past and use the cutoffs generated by it for the tth round, we will have an algorithm that converges to optimal. For obvious computational reasons, we do not want to maintain too many A_ϵ's with different ϵ's at the same time.

Roughly speaking, algorithm L gets over this problem by starting a new A_ϵ with $\epsilon \approx 1/\sqrt{t}$ only in round j, such that $j \approx 4^i$. It "warms up" A_ϵ through 4^{i+2}, evaluating the strategies but not using the cutoffs generated by A_ϵ. When A_ϵ is sufficiently warmed up, algorithm L uses the cutoffs generated by A_ϵ until the 4^{i+3}rd round, and then discards A_ϵ. This continual learning helps algorithm L to converge to optimal, while maintaining only a small number of A_ϵ's at any one time.

Let ℓ_ϵ, the expected number of examples seen in the first stage by algorithm A_ϵ, be as defined in Lemma 1. Formally, algorithm L does the following.

Algorithm L
begin
  for each round t with resource use time x_t **do**
    if there is no *current* A_ϵ **then** use a default threshold
    else use the threshold generated by the *current* A_ϵ
    endif
    if $t = 4^i - \ell_{1/2^{i+2}}$ **then** start a copy of $A_{1/2^{i+2}}$ and call this an *active* A_ϵ **endif**
    if $t = 4^i$ and $i > 2$ **then**
      discard *current* A_ϵ, if one exists
      set *current* A_ϵ to be $A_{1/2^i}$
    endif
    feed resource use time x_t to each *active* A_ϵ
  endfor
end

At any sufficiently large time t, there are at most three active A_ϵ's; i.e., if $4^i \leq t < 4^{i+1}$, the active A_ϵ's are $A_{1/2^i}$, $A_{1/2^{i+1}}$, and $A_{1/2^{i+2}}$. Hence, the space used by algorithm L is at most three times the space used by algorithm $A_{1/2^{i+2}}$, which we know from Section 4.1 is $O(c/2^i) = O(c\sqrt{t})$. In round t, $4^i \leq t < 4^{i+1}$, algorithm $A_{1/2^i}$ has seen at least $4^i - 4^{i-2} = (15/16) \cdot 4^i$ examples in its second stage; from Theorem 3, algorithm $A_{1/2^i}$ is away from optimal by at most

$$\frac{1}{2^i} + k_1(c+M)\sqrt{\frac{\ln(15t(c+M)/2^{i+2})}{15 \cdot 4^{i-2}}} = O\left(\sqrt{\frac{\ln t}{t}}\right).$$

The update time bound follows from Section 4.3.

5.2 Algorithm L_s

Algorithm L_s is exactly A_ϵ, with ϵ set appropriately such that $s = B + v + \max\{\eta_1, \eta_{2,1}\}$. (See Section 4.1.) Since $\epsilon = \Theta(c/s)$, Theorem 2 follows from the discussion in Section 4. The lower bound on s arises from ϵ being suitable.

6 Adaptive Disk Spindown via Rent-to-Buy

As described in Section 1, the disk spindown scenario can be modeled as a rent-to-buy problem, where spinning the disk is equivalent to renting, and a spindown is equivalent to a buy. If energy conservation were the sole consideration of a disk spindown algorithm, the cost of a buy, c, is the ratio of the energy required to spindown the disk and spin it back up vs. the power to keep the disk spinning. In practice, there are two conflicting goals of a disk spindown policy: conserving energy and preserving response time performance. In *adaptive disk spindown*, the user specifies the relative importance a of latency w.r.t. conserving energy, and the cost of the increased latency is integrated into c, the cost of the buy. We now describe precisely how this is done.

Let P_s be the power consumed by a spinning disk. Typically, a spundown disk consumes $P_{sd} > 0$ power, where P_{sd} is much smaller than P_s. Let T be the net idle time at disk.[3] This implies that the disk would consume at least $T \cdot P_{sd}$ energy independent of the disk spindown algorithm. While comparing disk spindown algorithms for how well they do in terms of energy consumed, it is instructive to compare the *excess energy*, $\mathcal{E}_X$, consumed by a disk while using spindown algorithm X; we define $\mathcal{E}_X$ as the total energy consumed by algorithm X minus $T \cdot P_{sd}$. (This is essentially equivalent to saying that the power for keeping the disk spinning is $P_s - P_{sd}$, and the power consumed by a spundown disk is 0.)

The response time delay incurred while waiting for a spinup is proportional to the amount of time required to spinup a spundown disk. A natural measure of the net response time delay is, therefore, the number of operations that are delayed by a spinup. (Other measures of response time delay are possible as discussed in Section 7.2.5, Item 4.)

In adaptive disk spindown, the user specifies a parameter a, the relative importance of latency w.r.t. conserving energy. Let O_X be the number of operations delayed by a spinup for algorithm X. Given a disk (spindown) management algorithm X, and a user specified parameter a, we define EC_X, the *effective cost* of algorithm X, as

$$\mathrm{EC}_X = \mathcal{E}_X + a \cdot \mathrm{O}_X. \tag{4}$$

The goal of the disk spindown algorithm is to minimize the effective cost. The effective cost models the tradeoff between energy and response time in a natural fashion. In particular, a small value of a implies that energy conservation is the more important activity, while a larger value of a implies that response time is more critical.

Minimizing effective cost can be modeled in the rent-to-buy scenario thus. Given the relative importance a, we determine the buy cost c. By definition, the value of c is the ratio of the effective cost for a spindown vs. the effective cost per unit time to keep the disk spinning. Since a spindown delays one operation, the effective cost of a spindown is $E_{sd} + a$, where E_{sd} is the total energy consumed by a spindown and a spinup. The effective cost per unit time to keep the disk spinning is $P_s - P_{sd}$. Hence, $c = (a + E_{sd})/(P_s - P_{sd})$. For a given disk, the buy cost c is linearly related to the relative importance parameter a.

7 Experimental Results

In this section we describe the results of simulating our algorithm[4] L from Section 5.1 for the disk spindown problem. We first describe the methodology used in our simulations and then describe the results of the simulation.

[3] We assume that operations are synchronous, and that every algorithm sees the same sequence of idle times at disk. If this is not true, T can be defined as the minimum taken over all algorithms of the net idle time at disk.

[4] Instead of scheduling a new A_ϵ at $t \approx 4^i$, in our simulations we scheduled a new A_ϵ at $t \approx 2^i$.

Characteristic	Hewlett-Packard Kittyhawk C3014A	Quantum Go•Drive 120
Capacity (Mbytes)	40	120
Power consumed, active, (W)	1.50	1.65
Power consumed, idle, (W)	0.62	1.00
Power consumed, spundown (W)	0.27	0.20
Power consumed, spinup (W)	2.17	5.50
Normal time to spinup (s)	1.10	2.50
Normal time to spindown (s)	0.55	6.00
Avg time to read 1 Kbyte (ms)	22.50	26.7

Table 1: Disk characteristics of the Kittyhawk C3014A and Quantum Go•Drive 120. (This table appears in [DKM].)

7.1 Methodology

We simulated algorithm L using a disk access trace from a Hewlett-Packard 9000/845 personal workstation running HP-UX. This trace is described in [RuW], and a portion of this trace was also used in a previous study of disk spindown policies [DKM]. The trace was obtained by Ruemmler and Wilkes by monitoring the disk for roughly two months; it consisted of 416262 accesses to disk.

We studied our algorithm for two disks, the Kittyhawk C3014A and the Quantum Go•Drive. The characteristics of the two drives are given in Table 1. (This table is derived from [DKM].) For our studies, we merged the active and idle states of the disk into one active state; notice that a disk can read and write data only in the active state. By merging these two states we ensure that a "buy" corresponds to a spindown. As in [DKM], we assumed that a disk access takes the average time for seek and rotational latency. We also assumed that all operations and state transitions take the average or "typical" time specified by the manufacturer, if one is specified, or else the maximum time.

It is difficult to determine from a disk access trace *why* a specific access arrived at disk. We assumed that, if the disk is spundown, the application waits for the disk to spinup and complete the requested operation, and then performs the same sequence of operations as in the original system. In other words, although our simulations used disks that were different from the one on which the trace was collected, in our simulator we maintained the inter-arrival time of events at disk as in the original trace: if, in the original trace, the tth access at disk arrived Δ seconds after the $(t-1)$th access, in our simulation, we assumed that the tth access arrived Δ seconds after the $(t-1)$th access was completed by the disk. The basic problem with any strategy is that data dependency between different operations cannot be derived from the trace.

We performed simulations for different values of a, the relative importance of response time to energy. For each a, we computed the buy cost c using the strategy described in Section 6. We compared our algorithm L against the following online algorithms: the *two-competitive algorithm*, which spins down the disk after c seconds of inactivity, and fixed-threshold policies that spindown the disk after 5 seconds, 30 seconds, and 5 minutes of inactivity; we also compared algorithm L against the *optimal offline* rent-to-buy algorithm, which knows the future and spins down the disk immediately if the next access is to take place more than c seconds in the future. For each algorithm X, we computed $\mathcal{E}_X$, the excess energy consumed, O_X, the number of operations delayed by a spinup; from these values we computed EC_X, the effective cost of algorithm X, using (4).

For the HP trace, the maximum inter-arrival time was 1770.4 seconds; the maximum a we used corresponded to a c of 1770.4.

7.2 Results

In this section we present the results of our simulations. We first see how the effective cost varies with parameter a, and then look at how excess energy and number of operations delayed vary with a. Recall that the parameter a is linearly related to the buy cost c. In particular, for the Kittyhawk disk, $c = 2.54 + a/1.225$, and for the Go•Drive, $c = 10.33 + a/1.45$.

The discussion from Section 6 implies that algorithm L and the 2-competitive algorithm try to optimize for effective cost as defined by (4). In particular, for really small values of a, algorithm L will essentially try to reduce excess energy, and for really large values of a, algorithm L will essentially try to reduce number of operations delayed.

7.2.1 Effective Cost vs. a

Figures 2 and 3 show how the effective cost varies with parameter a using the Kittyhawk and Go•Drive disks respectively. Each figure plots the curves for all values of a, and a clearer view for when a is small.

We observe that algorithm L performs best amongst the online algorithms for (almost) all values of a. (It is roughly 1% worse than the 5 second threshold for a lying between 18 and 34 while using the Kittyhawk disk, and for a lying between 14 and 28 while using the Go•Drive.) In particular, the effective cost for algorithm L is 6–25% lesser than the effective cost of the 2-competitive algorithm (except for a small range of values of a between 34 and 60 with the Kittyhawk disk and for a between 28 and 58 for the Go•Drive when the effective costs for the two algorithms are roughly the same).

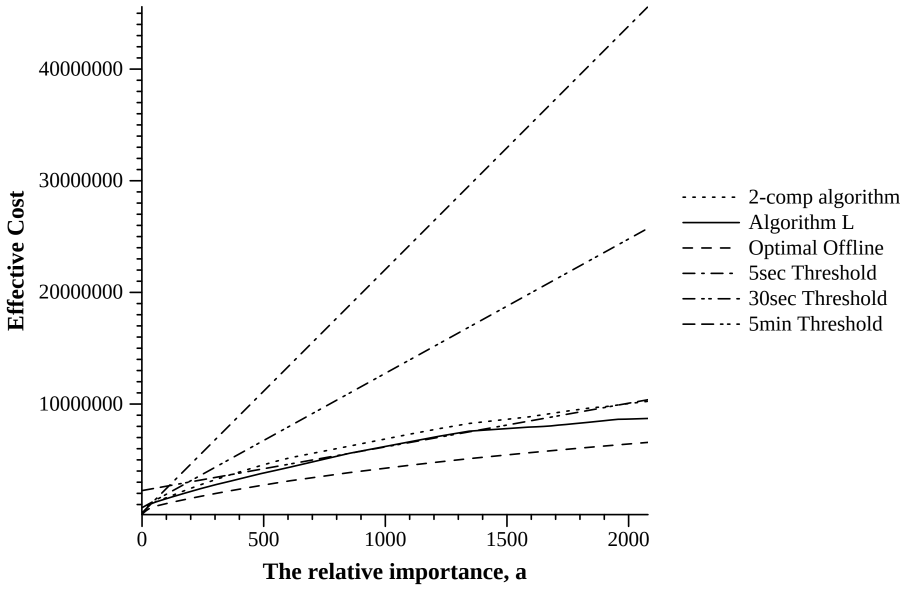
As should be expected, each fixed threshold algorithm performs well for a very limited range of values for a. Interestingly, the 5 second threshold for certain small values of a and the 5 minute threshold for certain large values of a performs better than the 2-competitive algorithm.

7.2.2 Excess Energy vs. a

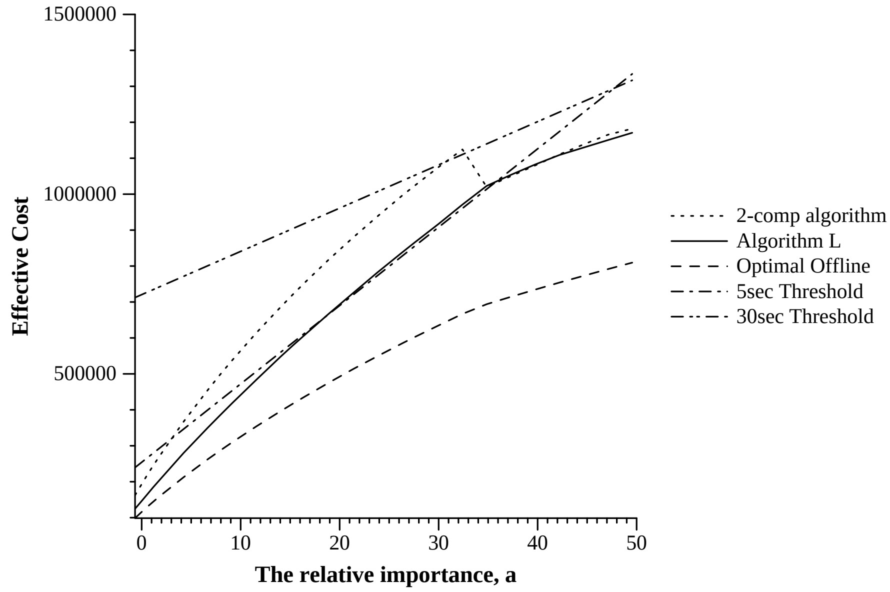
As discussed in Section 6, when a is small, conserving energy is more important. Figure 4 plots the variation of excess energy with a using the Kittyhawk and Go•Drive disks for the various algorithms.

We observe that for small values of a, algorithm L has the smallest excess energy amongst all online algorithms. In fact, it does better than the 5 second threshold, and its curve is almost parallel to the curve for the optimal offline algorithm. In particular, algorithm L saves 17–60% more excess energy as compared to the 2-competitive algorithm, and 6–42% more excess energy as compared to the 5 second spindown threshold for small values of a (i.e., $a < 25$).

We also observe that for small values of a, the 5 second threshold does better than the 2-competitive algorithm in terms of saving excess energy. (From Figures 2 and 3, we observe that for most of these values of a, the 5 second threshold is also better than the 2-competitive algorithm in terms of effective cost.)

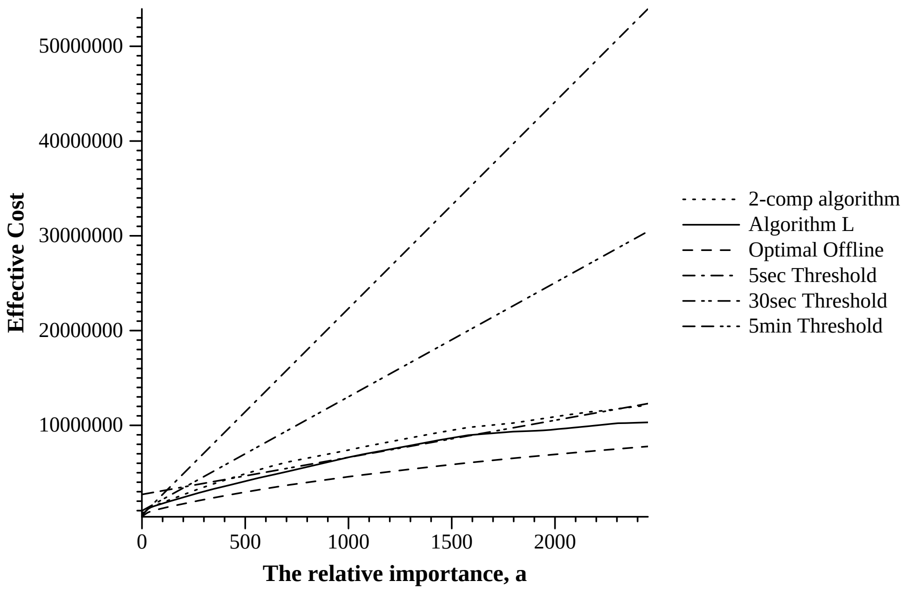


(a) For all values of a.

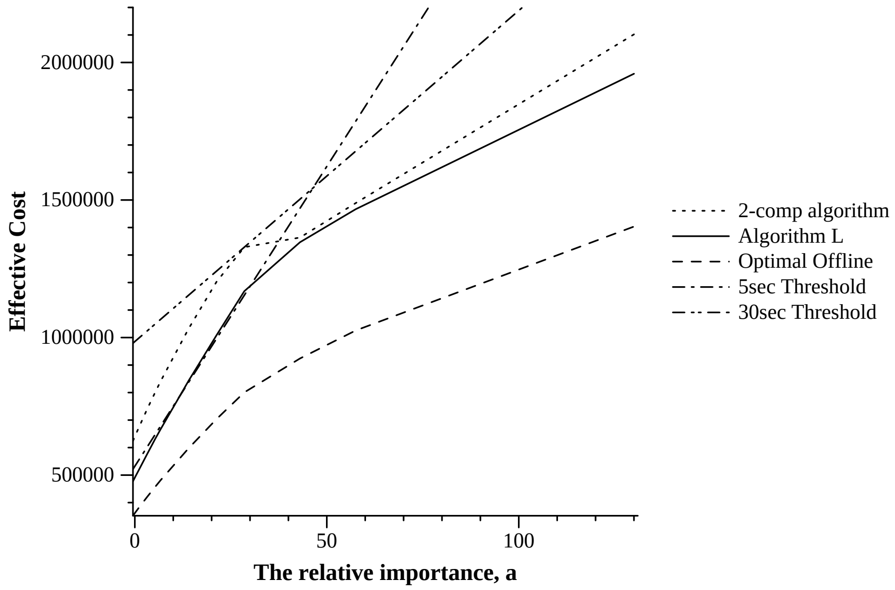


(b) For small values of a.

Figure 2: Variation of effective cost with a for the Kittyhawk disk. Figure (b) zooms the portion of the graph for small values of a. The effective cost of the 5 minute threshold is comparitively high (the curve lies above 2240000), and is omitted from Figure (b).

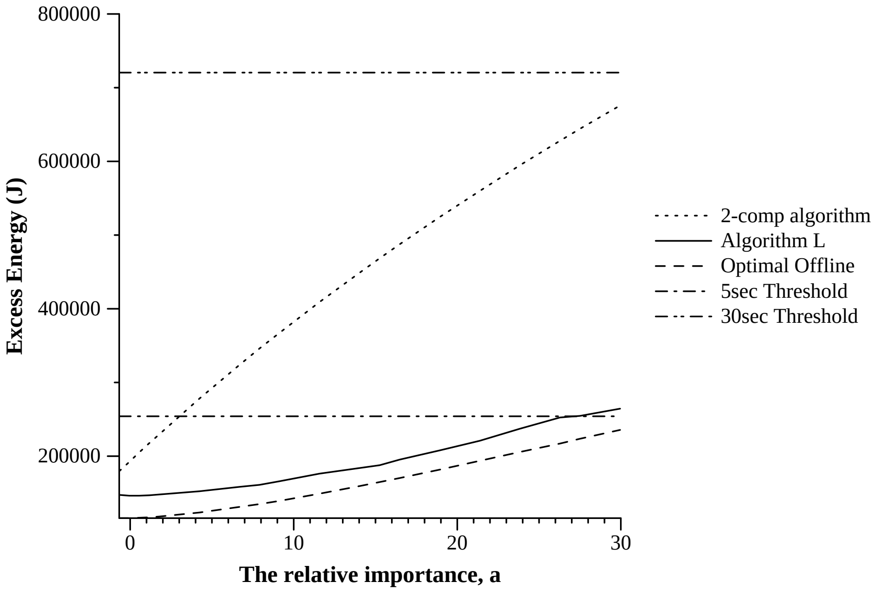


(a) For all values of a.

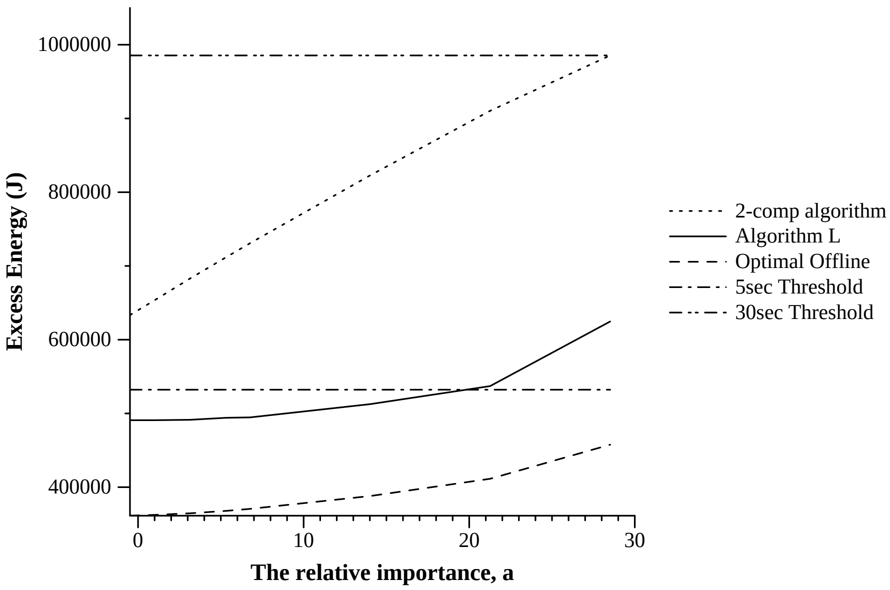


(b) For small values of a.

Figure 3: Variation of effective cost with a for the Go•Drive disk. Figure (b) zooms the portion of the graph for small values of a. The effective cost of the 5 minute threshold is comparitively high (the curve lies above 2700000), and is omitted from Figure (b); similarly, the curves for the 5 second and 30 second policies have been cropped at smaller values of a to show the details of the other three curves.



(a) Kittyhawk disk.



(b) Go•Drive disk.

Figure 4: Variation of excess energy with a for the Kittyhawk and Go•Drive disks. The excess energy of the 5 minute threshold using the Kittyhawk disk is 2249 KJ, and using the Go•Drive is 2708 KJ; the curves for the 5 minute threshold are omitted from the graphs.

7.2.3 Operations Delayed vs. a

As discussed in Section 6, when a is large, we want to reduce the number of operations delayed. Figure 5 plots the variation of number of operations delayed with a using the Kittyhawk and Go•Drive disks for the various algorithms.

We observe two interesting phenomenon: first, the curves for the 2-competitive algorithm and the optimal offline algorithm coincide for a large range of values for a. Second, algorithm L reduces number of operations delayed over both these algorithms for sufficiently large a.

7.2.4 Adaptability and Rent-to-Buy

A different way of viewing the tradeoff between excess energy and response time is presented in Figure 6. In this figure, excess energy is plotted as a function of number of operations delayed, and the different points on the curve are obtained by varying a; in particular, the value of a (or equivalently, c) decreases from left to right along the curve. (The curve for the Go•Drive is similar in shape and is omitted.)

Figure 6 clearly shows the tradeoff between excess energy and response time obtained by varying a. We observe that by increasing the value of one parameter a (equivalent to varying the value of the buy cost c), we can effectively trade power for response time. Concerns on how to effectively trade power for response time have been raised for the disk spindown problem [DKB, DKM], and the rent-to-buy model provides an elegant way of achieving this tradeoff.

7.2.5 Other Observations

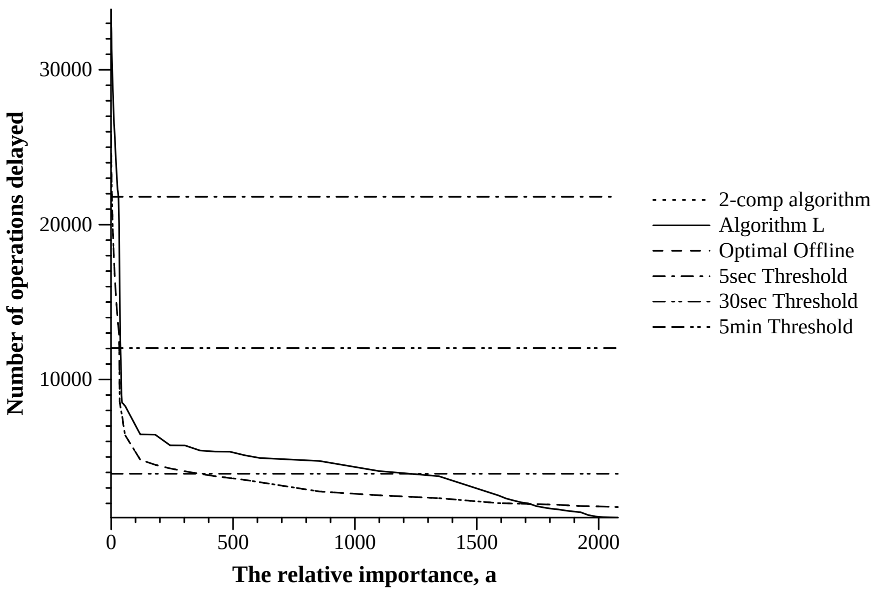
Some other observations from our simulations are as follows:

1. As mentioned in Section 7.2.2, energy conservation is crucial when a is small, and algorithm L is best amongst the online algorithms in terms of excess energy for small a. Interestingly, we observed that the excess energy of algorithm L is less than the excess energy of the 2-competitive algorithm for *all* values of a.

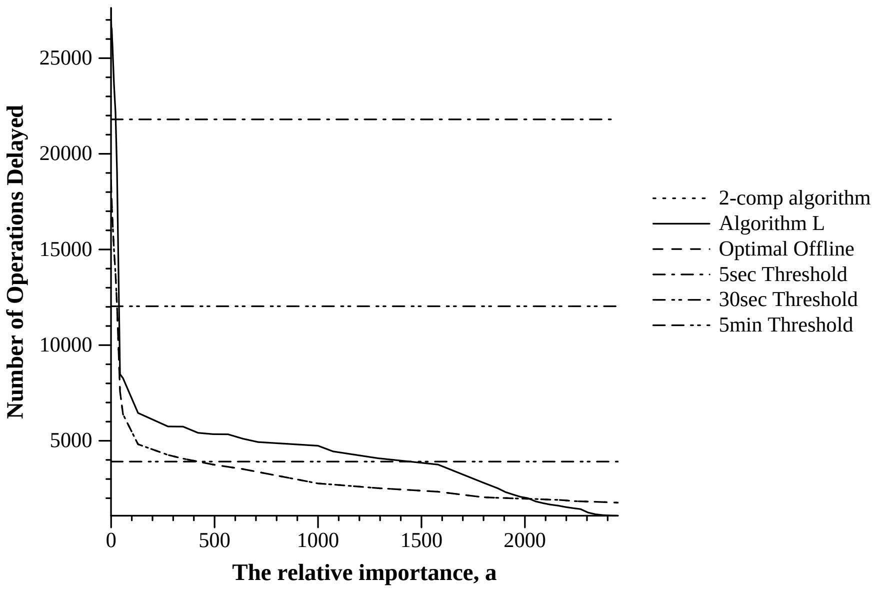
2. We also compared our algorithm L against L_s allowing at most 25 potential cutoffs for algorithm L_s. Not surprisingly, algorithm L performed better than algorithm L_s; however, preliminary results suggest that algorithm L typically saved only 2–5% more excess energy than algorithm L_s. Allowing more potential cutoffs for algorithm L_s might help.

3. In our simulations, we used at most 300 cutoffs for our algorithm L. The computation time for the algorithm was therefore minimal. Interestingly, algorithm L did not change its cutoffs too often in stage 2. (The cutoff changed between 14–56 times when measured over all values of a.)

4. For measuring response time performance, we used the metric of the number of operations delayed. An alternative measure of response time performance is R_X, the number of read operations delayed by a spinup for algorithm X [DKM]. This metric redefines the effective cost from (4) as $\mathcal{E} + a \cdot \mathrm{R}_X$. The rent-to-buy model can be easily modified to evaluate this measure, by having different costs for a spindown (i.e., different c's) depending on whether the operation is a read or a write. We plan to consider the effect of this modification to the rent-to-buy cost in future work.



(a) Kittyhawk disk.



(b) Go•Drive disk.

Figure 5: Variation of the number of operations delayed with a for the Kittyhawk and Go•Drive disks. The curves for the 2-competitive algorithm and the optimal offline algorithm coincide for a large range of values of a.

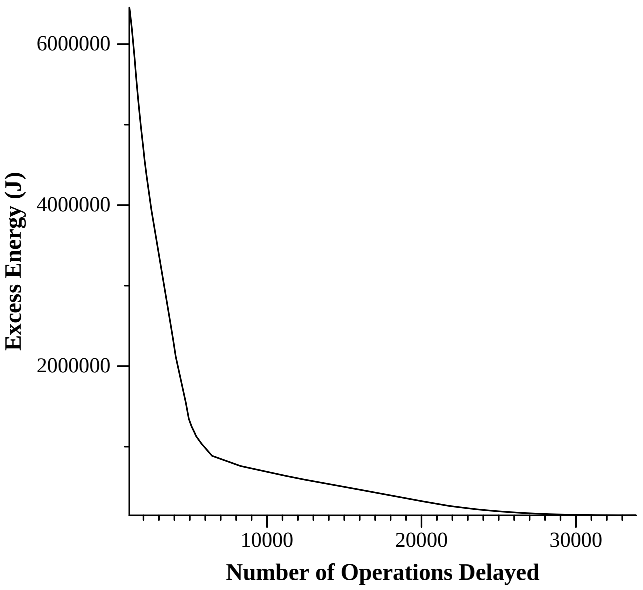


Figure 6: Excess Energy, $\mathcal{E}_L$, as a function of the number of operations delayed, O_L, for algorithm L. The graph was obtained by varying a (i.e., c); the value of a increases along the curve from left to right.

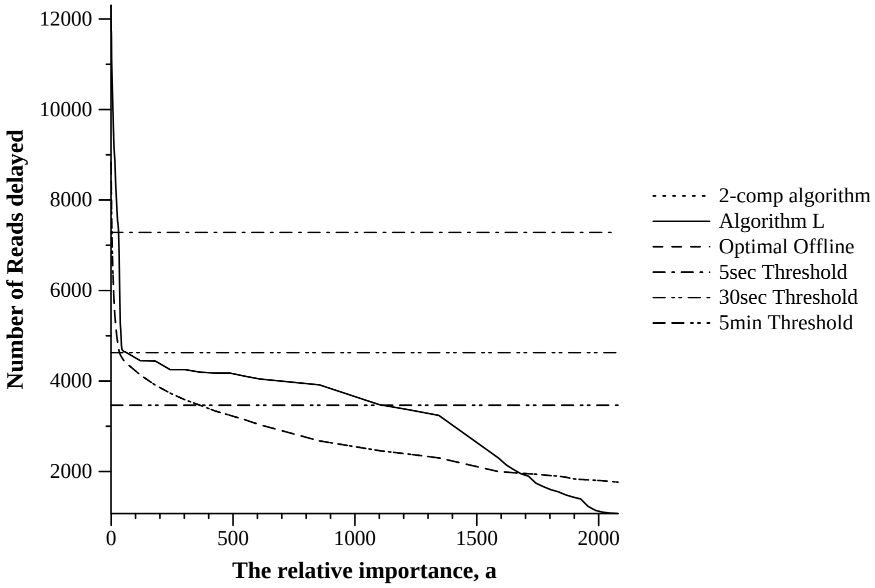


Figure 7: Number of reads delayed as a function of a for the various algorithms, while the rent-to-buy algorithms are optimizing using the definition of effective cost from (4). This graph is purely for illustration and comparison with Figure 5a. See Section 7.2.5, Item 4

.

For purely comparison purposes, Figure 7 plots the number of reads delayed as a function of a for the different algorithms; the algorithms are still optimizing for effective cost as defined by (4). (In other words, the rent-to-buy algorithms think they are optimizing for number of operations delayed, while we measure the number of reads delayed.) Interestingly, the curves from Figure 7 are similar to the corresponding curves from Figure 5a, suggesting that we should expect to obtain similar results as presented in this paper by using the number of reads delayed metric instead of the number of operations delayed metric, when we modify the definition for effective cost appropriately.

8 Conclusions

In this paper we have looked at the problem of a sequence of unit rent-to-buy choices where the resource use times are independently drawn from an unknown probability distribution. We have looked at computationally efficient strategies whose expected cost for the tth resource use converges to optimal as $t \to \infty$ for any bounded probability distribution on the resource use times. We have also looked at a fixed-space algorithm which almost converges to optimal. We are currently looking at modeling the resource use times as being generated by a Hidden Markov Model (HMM) and have optimality results for special types of HMMs. Recently, Markov models have been effectively used to analyze caching and prefetching algorithms assuming user requests to pages in cache are generated by Markov sources [KPR, KrV, ViK].

Simulations of our algorithm for the disk spindown problem using disk access traces obtained from HP suggest that the rent-to-buy model is a good way to study disk spindown and related systems issues; in particular, a single parameter c effectively models the tradeoff between power and response-time. We also introduced the new metric of "excess energy" that really reflects the relative performance in terms of energy consumed of one disk spindown algorithm against another. We introduced a natural notion of "effective cost" that incorporates the two metrics of excess energy, and number of operations delayed weighted by a user specified parameter a, into one cost. We observed that our algorithm L out-performed other online algorithms in terms of effective cost for almost all values of a; in particular, it had 6–25% lesser effective cost than the 2-competitive algorithm. In addition, for small values of a (corresponding to when saving energy is critical), we observed that our algorithm L saves 17–60% more of excess energy as compared to the 2-competitive algorithm, and 6–42% more excess energy as compared to the 5 second fixed threshold.

Acknowledgements

We thank John Wilkes and Hewlett-Packard Company for making their file system traces available to us. We thank Peter Bartlett for his comments, and Fred Douglis for his comments and interesting discussions related to the disk spindown problem.

References

[BEH] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the Vapnik Chervonenkis Dimension," *Journal of the ACM* (October 1989).

[DKB] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive Disk Spindown Policies for Mobile Computers," *Proceedings of the Second USENIX Symposium on Mobile and Location Independent Computing*, to appear.

[DKM] F. Douglis, P. Krishnan, and B. Marsh, "Thwarting the Power Hungry Disk," *Proceedings of the 1994 Winter USENIX* (January 1994).

[Gre] P. Greenawalt, "Modeling Power Management for Hard Disks," *Proceedings of the Symposium on Modeling and Simulation of Computer Telecommunication Systems* (1994).

[Hew] Hewlett Packard, "Kittyhawk HP C3013A/C3014A Personal Storage Modules Technical Reference Manual," March 1993, HP Part No. 5961-4343.

[KLM] A. R. Karlin, K. Li, M. S. Manasse, and S. Owicki, "Empirical Studies of Competitive Spinning for a Shared-Memory Multiprocessor ," *Proceedings of the 1991 ACM Symposium on Operating System Principles* (1991), 41–55.

[KMM] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive Randomized Algorithms for Non-Uniform Problems," *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms* (1990), 301–309.

[KPR] A. R. Karlin, S. J. Phillips, and P. Raghavan, "Markov Paging," *Proceedings of the 33rd Annual IEEE Conference on Foundations of Computer Science* (October 1992), 208–217.

[KeS] M. J. Kearns and R. E. Schapire, "Efficient Distribution-Free Learning of Probabilistic Concepts," *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science* (October 1990), 382–391.

[KLP] S. Keshav, C. Lund, S. J. Phillips, N. Reingold, and H. Saran, "An Empirical Evaluation of Virtual Circuit Holding Time Policies in IP-over-ATM Networks," *Proceedings of INFOCOM '95*.

[KrV] P. Krishnan and J. S. Vitter, "Optimal Prediction for Prefetching in the Worst Case," *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms* (January 1994), Also appears as Duke University Technical Report CS-1993-26..

[LKH] K. Li, R. Kumpf, P. Horton, and T. Anderson, "A Quantitative Analysis of Disk Drive Power Management in Portable Computers," *Proceedings of the 1994 Winter USENIX* (January 1994).

[LPR] C. Lund, S. Phillips, and N. Reingold, "IP over Connection-Oriented Networks and Distributed Paging," *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science* (November 1994), 424–434.

[MDK] B. Marsh, F. Douglis, and P. Krishnan, "Flash Memory File Caching for Mobile Computers," *Proceedings of the 27th IEEE Hawaii Conference on System Sciences* (January 1994).

[Pol] D. Pollard, *Convergence of Stochastic Processes*, Springer-Verlag, 1984.

[RuW] C. Ruemmler and J. Wilkes, "UNIX Disk Access Patterns," *Proceedings of the 1993 Winter USENIX Conference* (January 1993), 405–420.

[SaK] H. Saran and S. Keshav, "An Empirical Evaluation of Virtual Circuit Holding Times in IP-over-ATM networks," *Proceedings of INFOCOM '94* (June 1994).

[Vap] V. N. Vapnik, "Inductive principles of the search for empirical dependences (methods based on weak convergence of probability measures)," *Proceedings of the 1989 Workshop on Computational Learning Theory* (1989).

[VaC] V. N. Vapnik and A. Y. Chervonenkis, "On the Uniform Convergence of Relative Frequencies of Events to their Probabilities," *Theoretical Probability and Its Applications* 16 (1971), 264–280.

[ViK] J. S. Vitter and P. Krishnan, "Optimal Prefetching via Data Compression," *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science* (October 1991), 121–130, Also appears as Brown University Technical Report No. CS–91–46.