Discrete Mathematics and Theoretical Computer Science DMTCS vol. **8**, 2006, 97–120

The Online Specialization Problem

Edwin S. Hong [1]

[1] *University of Washington, Tacoma, Computing and Software Systems, Institute of Technology, 1900 Commerce Street, Box 358426, Tacoma, WA 98402 (253) 692-4659. Email: edhong@u.washington.edu.*

received Sep 2004, *revised Apr 2005*, *accepted Mar 2006*.

We study the online specialization problem, where items arrive in an online fashion for processing by one of n different methods. Each method has two costs: a processing cost (paid once for each item processed), and a set-up cost (paid only once, on the method's first use). There are n possible types of items; an item's type determines the set of methods available to process it. Each method has a different degree of specialization. Highly specialized methods can process few item types while generic methods may process all item types. This is a generalization of ski-rental and closely related to the capital investment problem of Azar et al. (1999). We primarily study the case where method $i + 1$ is always more specialized than method i and the set-up cost for a more specialized method is always higher than that of a less specialized method. We describe an algorithm with competitive ratio $O(\log n)$, and also show an $\Omega(\log n)$ lower bound on the competitive ratio for this problem; this shows our ratio is tight up to constant factors.

Keywords: online algorithms, competitive analysis, specializations

1 Introduction

To motivate the online specialization problem, consider the scenario of hosting an online data archival service. Customers are expected to store many data files into the archive regularly but rarely read data from the archive. To minimize the cost of operating the archive, the host could automatically compress the data files before storing them in archive. Since the incoming files could represent text, sound, or any number of other possible types, different compression algorithms are needed for an efficient system.

As a simple example, suppose there are four different methods for processing data: method f_1 denoting no compression at all, f_2 denoting a standard dictionary coding technique good for generic unicode text, f_3 denoting a specialized encoding scheme for English prose, and f_4 an efficient compressor for sound. An English novel could be compressed very efficiently with f_3, and less efficiently with f_2, and not at all with f_4. We say that f_3 is more specialized than f_2 (denoted $f_2 \prec f_3$) because f_2 can compress anything that f_3 can compress. We also know $f_1 \prec f_2$, $f_1 \prec f_4$, and f_2 is incomparable to f_4.

A simple model for the costs of operating the archive would be to assume each method has two costs: a set-up cost c_i representing the cost of creating (or purchasing) method f_i, as well as a "processing cost" p_i reflecting the cost of maintaining the storage space of any compressed file produced by f_i. This is an extremely oversimplified model for this scenario that assumes several things: 1) The cost of computer time for encoding and decoding is insignificant compared to the costs for creating the methods and for physical storage of the data; 2) All input files are the same size; and 3) Each method reduces the size of input files of the appropriate type by a fixed amount. Assumptions 2 and 3 imply a predetermined final size of any input file processed by method f_i; when the processing cost p_i represents the cost for storing the file, it is proportional to the final compressed file size. Note that assumption 2 may not be completely unrealistic as any large file can be viewed as a sequence of many uniformly-sized smaller files.

As the input files of different types arrive in an online fashion, we need to choose between the available compression methods, incurring the processing costs for each input, as well as set-up costs during the first use of each method. Using the standard competitive analysis framework (see Borodin and El-Yaniv (1998)), our goal is to find an algorithm with low competitive ratio. This means we wish to minimize the cost that our online algorithm incurs compared to the cost for the optimal algorithm that knows all inputs in advance. We assume that the methods, their set-up and processing costs, and the $\prec$-relation between them are known in advance.

The original motivation for studying this problem came from the dynamic compilation problem. Dynamic compilation attempts to decrease a program's execution time by using run-time information to perform additional optimization techniques. For example, consider a program P that has a function $f(x, y)$, and suppose P calls $f(3, 4)$ many times. A dynamic compilation system could speed up program P by first detecting that $f(3, 4)$ is called many times and then, next computing and storing the value, and finally performing a look-up of the value instead of recomputing for all future calls with the same parameters. This is called specializing f for $\mathtt{x} = 3$ and $\mathtt{y} = 4$.

Let f_1, f_2, and f_3 represent the unspecialized version of f, f specialized for $\mathtt{x} = 3$, and f specialized for $\mathtt{x} = 3$ and $\mathtt{y} = 4$, respectively. Specialized version f_2 could be created by applying any standard compiler optimization (such as constant folding and loop unrolling) to the function f while assuming $\mathtt{x} = 3$.

In the general problem, there are n different methods to process the inputs, and n different types of input. Furthermore, there is a *specialization hierarchy* defined by a partial order that represents the degree of specialization of each method. Our online algorithm must decide which specialization (if any) it should create on every input. In this paper, we concentrate primarily on the case where more specialized methods have higher set-up costs, and on the case where the graph representing the specialization hierarchy is a line. For this case we define an online algorithm that makes these specialization decisions with competitive ratio of $O(\log n)$. We also give a lower bound proof showing that every online algorithm has competitive ratio $\Omega(\log n)$.

1.1 Related work

To our knowledge, no one has studied the online specialization problem before, although it is a generalization of previous work. The ski-rental problem is a simplification of our problem to the case where there are only two methods, and where all inputs can be processed with either method. It was initially proposed as an analogue to the competitive snoopy caching problem Karlin et al. (1988).

A generalization of ski-rental that is relevant to the problem we describe here is the capital investment problem studied by Azar et al. (1999). They considered the following manufacturing problem: Initially, there is a set of machines that can manufacture a certain item. Each machine costs a certain amount to buy and can produce the item at a certain cost. Furthermore, at any time, technological advances could occur which would make new machines available for purchase. These new machines have different purchase costs and lower per-item production costs. They studied the problem of designing an algorithm that minimizes the total cost of production (the sum of the cost of buying machines and the cost of producing items).

The online specialization problem is a generalization of the capital investment problem to the case where machines can be specialized, and some items which are manufactured can be produced by some machines and not others. However, the online specialization problem does not include the idea of technological advances, where the adversary could give the online algorithm completely new methods for processing the input at future points in time; in our problem all methods are assumed to be known at the beginning.

Many other ski-rental generalizations have a multi-level structure related to our problem. However, none of them adequately represent the specialization hierarchy. For instance, the online file allocation problem (see Awerbuch et al. (2003)) takes a weighted graph representing a network and files located at nodes in the network. The online input is a series of access requests to the files from particular nodes, and the algorithm is allowed to move data files from one node to another at some cost. With an appropriate network design, the processing costs in our algorithm could be modeled by access requests from one node for files located at many other nodes. However, the idea of generic specializations that can process many input types would force nodes modeling a generic specialization f_1 to always have all data files that represent methods more specialized than f_1. In other words, the algorithm cannot decide to migrate files on a strictly individual basis; some files would be grouped into hierarchical layers so that any node with a layer i file must always have all layer j files for all $j > i$.

Other multi-level structure problems studied using competitive analysis include caching strategies with multi-level memory hierarchies for minimizing access time (starting with Aggarwal et al. (1987)), and online strategies for minimizing power usage of a device by transitioning between multiple available power states when idle (see Irani et al. (2005)). See Grant et al. (2000) for an experimental dynamic compilation system that motivated this work.

1.2 Algorithmic design issues

First consider the ski-rental case Karlin et al. (1988) where there are two methods f_1 and f_2, and where $c_1 = 0$. An online algorithm with competitive ratio 2 is to wait until $c_2/(p_1 - p_2)$ inputs are seen before creating f_2.

Our problem is significantly more challenging because of how the many different methods interact with one another. Consider the case with three methods f_1, f_2, and f_3; $c_1 = 0$; $f_2 \prec f_3$; and $p_2 > p_3$. Every input that f_3 can process can also be processed by f_2. The defining question with specialized methods is choosing between f_2 and f_3 when the inputs can be processed with either. Creating f_3 first is better when many future inputs of type f_3 occur; however, if many future inputs can use f_2 but not f_3, then f_2 is a better choice. We are faced with a tradeoff between highly specialized methods with low processing costs and widely applicable methods with high processing costs.

Now consider the case with n increasingly specialized versions of f: $(f_1 \prec f_2 \prec \cdots \prec f_n)$. We say an input has *specialization level* i when it can be processed by any $f \preceq f_i$ but not by any $f \succ f_i$. One difficulty with this problem is in measuring the actual benefit of creating f_i with respect to a worst-case adversary. The following scenario illustrates that the apparent benefit of creating f_i can decrease as more inputs are seen.

Suppose f_1 is the only method created so far and let $p_1 > p_2 > \cdots > p_i$, and $c = c_{i-1} = c_i$. Suppose $k = c/(p_1 - p_i)$ inputs with specialization level i have been seen so far, and no inputs with lower specialization levels have yet been seen. The apparent benefit of creating f_i is $k(p_1 - p_i)$, representing the gain in benefit when using f_i from the beginning instead of f_1. Since this benefit exactly equals the cost c of creating f_i, perhaps f_i should be created. However, suppose we anticipate processing so many future inputs of specialization level $i-1$ that creating f_{i-1} is guaranteed to be a good decision. In this anticipated future scenario, the benefit of creating f_i for the k inputs of specialization level i is only $k(p_{i-1} - p_i)$, which may be much smaller than the apparent benefit of $k(p_1 - p_i)$. It is not obvious when f_i should be created in this scenario.

1.3 Naive extension

We end this section showing the poor performance of a simple extension of the ski-rental algorithm to the case of n increasingly specialized versions of f. This algorithm, when processing the next input I, computes what the optimal offline algorithm would do on all inputs seen so far since the beginning of the algorithm, including I. Then it processes I with the same specialization as the optimal, creating it if necessary.

Consider this algorithm's performance in the special case where the costs are $c_1 = 0, c_2 = c_3 = \cdots = c_n = 1$, $p_1 = 1$, and $p_i = \frac{1}{n} - \frac{i}{2n^2}$ for $i > 2$. This is designed so that the processing costs from p_2 to p_n decrease linearly from from just below $\frac{1}{n}$ down to $\frac{1}{2n}$. For this specific case with just one input, the online algorithm creates f_1 and uses it because $c_1 = 0$. Suppose two inputs with specialization level n are given. Clearly the optimal offline algorithm would use f_n on those inputs to get a minimal total cost of $1 + 1/n$, where 1 is the creation cost and $1/n$ is the processing cost for both inputs. This means the online algorithm would create and use f_n when it sees the second input. Now suppose two inputs of level n are given followed by one of level $n-1$. The optimal cost would use f_{n-1} for all three inputs; creating both f_{n-1} and f_n is not cost effective because the relatively high creation costs do not offset the reduced processing cost. This means the online algorithm would create and use f_{n-1} on the third input, after already having created f_n for the second input.

Now suppose the total input consists of k inputs with specialization level 2 or higher, where $2 \le k \le n$. The optimal offline strategy here is to just create and use f_i on all the inputs where i is the minimum specialization level among the k inputs. At least one specialization in $\{f_2, f_3, ..., f_n\}$ must be created because of the high processing cost of f_1 compared to all other processing costs. No more than one can be created because the added benefit of creating a second specialization in the set is at most a $\frac{1}{2n}$ reduction in processing cost per input, and the benefit must outweigh the creation cost of 1. This means that more than $2n$ total inputs are necessary before the optimal algorithm could even consider creating more than one specialization in $\{f_2, f_3, ..., f_n\}$. And since we know the optimal must use create just one specialization, the best one to create to minimize processing costs is clearly the most highly specialized one that can be applied to all inputs.

The behavior of the optimal offline algorithm above implies the following adversarial strategy for this problem. Give the online algorithm two inputs with specialization level n, followed by one input each at level from $n-1$ down to 2, in the specified order. Note that n inputs are given overall.

With this strategy, the online algorithm uses f_1 for the first input. For the second input it uses f_n because the optimal would have used f_n when seeing just the first two inputs. On the third input, the online algorithm uses f_{n-1} because the optimal would use f_{n-1} for all three inputs. Generalizing, we see that the online algorithm uses the specializations $f_1, f_n, f_{n-1}, f_{n-2}, ..., f_3, f_2$ in that order for the n inputs in the adversarial strategy. It is basically tracking the behavior of the optimal offline algorithm on a prefix of the entire input.

The total cost incurred by this online algorithm on these n inputs is thus more than $n-1$ because it paid to create all specializations of level 2 and higher. In contrast, the optimal algorithm uses f_2 exclusively and pays just $c_2 + np_2 = 1 + n\left(\frac{1}{n} - \frac{1}{n^2}\right) = 1 + 1 - 1/n$. This shows a ratio of $\frac{n-1}{2}$; this algorithm has a competitive ratio of at least $\Omega(n)$.

The lesson learned is to not aggressively create highly specialized methods, even if the optimal solution would create them based on inputs seen so far. Creating intermediate level specializations in anticipation of future inputs with specialization levels in the middle is a much better idea.

2 Problem definition and results

2.1 Online specialization problem definitions

- $\mathcal{F} = \{f_1, f_2, ..., f_n\}$ denotes the set of all specializations.
- $c : \mathcal{F} \to [0, +\infty)$ is a function representing the *set-up cost* or *creation cost* for each element of $\mathcal{F}$. The cost $c(f_i)$, abbreviated c_i, denotes the creation cost of f_i. The *standard ordering* for specializations in $\mathcal{F}$ is in non-decreasing creation cost order, so $c_1 \leq c_2 \leq \cdots c_n$.
- $p : \mathcal{F} \to (0, +\infty)$ represents the *processing cost* for each f_i; $p(f_i)$ is abbreviated p_i.
- $\prec$ is the partial order over $\mathcal{F}$ that determines the specialization hierarchy.
- $\sigma = (I_1, I_2, ..., I_t)$ represents the input sequence.
- $\mathsf{lev}(I)$ denotes the *specialization level* of input I, defined so if $\mathsf{lev}(I) = f_i$ then any method f_j such that $f_j \preceq f_i$ can be used to process I.

We assume the set $\mathcal{F}$, the cost functions p and c, as well as the partial order $\prec$ are known before the online algorithm starts. The online algorithm outputs decisions about when to create each f_i as it sees the input sequence. We assume that once f_i has been created, it persists and is available for use by any future input. We study the following two special cases of the online specialization problem.

- The *monotone linearly-ordered specialization* problem, denoted $\mathrm{MLS}(\mathcal{F}, c, p, \prec)$, assumes $\prec$ is a total ordering on $\mathcal{F}$ and the following *monotonicity constraint* applies.

  $$\text{For } i \neq j, \text{ if } f_j \prec f_i, \text{ then } c(f_i) \geq c(f_j).$$

  The monotonicity constraint says that more specialized methods have higher set-up costs. Note that if $f_j \prec f_i$, $c_i \geq c_j$, and $p_i \geq p_j$, then no reasonable algorithm would ever use f_i. This is because f_j is more widely applicable than f_i and it also costs less. Thus without loss of generality, in this monotone setting we assume that if $f_j \prec f_i$, then $p_i < p_j$. The standard ordering for $\mathcal{F}$ based on creation costs also orders the methods by specialization level and by processing cost, so that $f_1 \prec f_2 \prec \cdots \prec f_n$, and $p_1 > p_2 > \cdots > p_n$.

- The *equally specialized* problem, denoted $\mathrm{ES}(\mathcal{F}, c, p)$ assumes that all methods can be used to process any input. Furthermore, we also assume that $c_i > c_j$ implies $p_i < p_j$. Note that an algorithm solving this special case is given in Azar et al. (1999), as well as in section 3.1.

We now define the costs for online algorithm $\mathcal{A}$ running on problem $\mathcal{P}$ with input σ.

- $\mathsf{CRE}^{\mathcal{P}}_{\mathcal{A}}(\sigma)$ denotes the cost for creating specializations.
- $\mathsf{PROC}^{\mathcal{P}}_{\mathcal{A}}(\sigma)$ denotes the processing cost.
- $\mathsf{TOT}^{\mathcal{P}}_{\mathcal{A}}(\sigma) = \mathsf{PROC}^{\mathcal{P}}_{\mathcal{A}}(\sigma) + \mathsf{CRE}^{\mathcal{P}}_{\mathcal{A}}(\sigma)$ denotes the total cost paid by $\mathcal{A}$ on σ.

$\mathsf{OPT}^{\mathcal{P}}(\sigma)$ denotes the cost of the optimal offline algorithm. Note that the optimal offline algorithm simply chooses the optimal set of methods to create at the beginning and then processes each input with the appropriate method.

In the above definitions, whenever problem $\mathcal{P}$, algorithm $\mathcal{A}$, or input σ is understood from context, we omit it. In the equally specialized setting, the online (and optimal) cost is determined solely by the number of inputs the adversary gives. Thus, given an ES problem, we replace the input parameter σ with an integer k representing the length of σ.

We derive performance bounds using standard competitive analysis. Let $\mathcal{P}(n)$ be the set of all problems with n specializations, and $\Sigma_{\mathcal{P}}$ denote the set of all possible inputs for $\mathcal{P}$. Then we say an algorithm $\mathcal{A}$ is $\rho(n)$-*competitive* if

$$\sup_{\substack{\mathcal{P}\in\mathcal{P}(n),\\ \sigma\in\Sigma_{\mathcal{P}}}} \frac{\mathsf{TOT}^{\mathcal{P}}_{\mathcal{A}}(\sigma)}{\mathsf{OPT}^{\mathcal{P}}(\sigma)} \leq \rho(n).$$

2.2 Results

For the MLS problem as described in section 2.1, we construct an algorithm MLSA that has competitive ratio $O(\log n)$, where n is the number of methods available. Our algorithm makes calls to ESA, a slightly modified version of the capital investment algorithm (CIA) from Azar et al. (1999). MLSA often creates the same specializations as ESA running on a related ES problem. The main idea is to partition $\mathcal{F}$ into "contiguous intervals" of the form $\{f_i, f_{i+1}, ...f_j\}$, and assign a worker to each interval. A worker's job is to process the inputs whose specialization level is inside its interval. Each worker runs completely independently of the other workers and makes its decisions based on the ESA algorithm running on the problem $\mathrm{ES}(\mathcal{F}', c, p)$, where $\mathcal{F}' \subset \mathcal{F}$. When a worker decides to quit, its interval is partitioned into smaller intervals, and more workers are created to handle the new intervals.

The two main theorems of this paper are as follows.

Theorem 2.1 *The algorithm* MLSA *on problem* $\mathrm{MLS}(\mathcal{F}, c, p, \prec)$ *is* $(1 + 13\log n)$*-competitive.*

Theorem 2.2 *Any online algorithm for the online monotone linearly-ordered specialization problem has competitive ratio at least* $(\log n)/4$.

Section 3 describes and analyzes the properties of ESA for the equally specialized setting. Section 4 describes and analyzes MLSA. Section 5 describes an adversary and a particular $\mathrm{MLS}(\mathcal{F}, c, p, \prec)$ problem that shows an $\Omega(\log n)$ lower bound on the competitive ratio for any online algorithm.

3 Design and analysis of ESA

3.1 Design of ESA

ESA is an online algorithm for solving $\mathrm{ES}(\mathcal{F}, c, p)$; it can be thought of as a simple modification to the capital investment algorithm (CIA) that slightly delays the creation of specializations. The overall idea for ESA is to use the best already-created specialization until the total processing costs so far is roughly equal to the total optimal cost. ESA improves upon CIA because it is 6-competitive (as opposed to 7-competitive); however, it cannot handle the technological advances present in the capital investment problem Azar et al. (1999). ESA behaves as follows.

For the first input, ESA creates and uses the method f' that minimizes $c(f) + p(f)$ over all $f \in \mathcal{F}$. For the kth input, $k > 1$, let f_b be the method ESA used for I_{k-1}. Then ESA first checks the condition

$$\mathsf{PROC}(k-1) + p(f_b) > \mathsf{OPT}(k). \tag{1}$$

If condition (1) is false then ESA uses f_b on the kth input. Otherwise, ESA creates and uses the method that minimizes $p(f)$ from the set

$$\{f : c(f) < 2\mathsf{PROC}(k-1) + p(f)\}. \tag{2}$$

3.2 *Analysis of* ESA

We first show that ESA is well defined, in particular, that whenever it chooses to create a new method, the set (2) above will contain a specialization that is more highly specialized than the one ESA had previously been using. We also state some lemmas about ESA that are required later in this paper for analyzing the performance of MLSA.

Lemma 3.1 *Suppose condition (1) is true in the* ESA *algorithm when processing input I_k. Let f^* be the method used by the optimal algorithm for k inputs. Let f_x be the new method from the set (2) above that* ESA *uses for input I_k. Then we know that $f^* \preceq f_x$.*

Proof: The truth of condition (1) implies that

$$c(f^*) < c(f^*) + kp(f^*) = \mathsf{OPT}(k) < \mathsf{PROC}(k-1) + p(f_b).$$

Let f' be the first method created by ESA. Applying $p(f_b) \le p(f') \le \mathsf{PROC}(k-1)$ to the previous statement shows $c(f^*) < 2\mathsf{PROC}(k-1)$. This means that f^* is available for ESA to pick for processing input I_k. Thus, the actual method f_x that ESA picks must be either f^* or another method with a lower processing cost than f^*; this shows $f^* \preceq f_x$. □

Lemma 3.2 *For any problem $\mathcal{P} = \mathrm{ES}(\mathcal{F}, c, p)$, $\forall k, \mathsf{PROC}^{\mathcal{P}}_{\mathrm{ESA}}(k) \le \mathsf{OPT}^{\mathcal{P}}(k)$.*

Proof: Because of the way the first method f' is chosen, it is clear that $\mathsf{PROC}(1) \le \mathsf{TOT}(1) = \mathsf{OPT}(1)$. Assume by induction that $\mathsf{PROC}(k-1) \le \mathsf{OPT}(k-1)$. Let f_b be the method used on I_{k-1} and consider the kth input. If the condition 1 from section 3.1 is false, then we use f_b on I_k and know that $\mathsf{PROC}(k) \le \mathsf{OPT}(k)$. Otherwise condition (1) is true, and we pay cost $\mathsf{PROC}(k-1) + p(f_x)$ for k inputs, where f_x is the method chosen from set (2). Let f^* be the method used by the optimal algorithm for k inputs, so $\mathsf{OPT}(k) = c(f^*) + kp(f^*)$. From lemma 3.1, we know $f^* \preceq f_x$, so that $p(f_x) \le p(f^*)$. Then by our induction hypothesis and by the optimality of OPT, $\mathsf{PROC}(k) = \mathsf{PROC}(k-1) + p(f_x) \le \mathsf{OPT}(k-1) + p(f^*) \le c(f^*) + (k-1)p(f^*) + p(f^*) = \mathsf{OPT}(k)$. □

Corollary 3.3 *Whenever condition (1) is true,* ESA *will create a new specialization that has a higher creation cost and lower processing cost than any specialization previously created.*

Proof: Since ESA's processing cost stays below the optimal total cost (lemma 3.2), we know that condition (1) is typically false, and that it becomes true only when the current specializiation f_b being used is less specialized (and has a higher processing cost) than the optimal one f^* for on k inputs. Thus, when the condition becomes true, it will always be possible to switch to a better specialization. If all specializations have been created, then the condition will never become true again. □

Lemma 3.4 *Given a problem* $\mathcal{P} = \mathrm{ES}(\mathcal{F}, c, p)$. *Then*

$$\forall k, j > 0, \mathsf{PROC}^{\mathcal{P}}_{\mathrm{ESA}}(k+j) \leq \mathsf{PROC}^{\mathcal{P}}_{\mathrm{ESA}}(k) + \mathsf{PROC}^{\mathcal{P}}_{\mathrm{ESA}}(j).$$

Proof: Clearly the processing cost per input decreases over time as ESA creates specializations with lower processing cost. Thus processing $k + j$ inputs together is better than processing k inputs and then restarting and processing j more. □

Lemma 3.5 *For any problem* $\mathcal{P} = \mathrm{ES}(\mathcal{F}, c, p)$*, let* k^* *be the number of inputs* ESA *must see before it creates its second specialization. Then* $\forall k \geq k^*, \mathsf{CRE}^{\mathcal{P}}_{\mathrm{ESA}}(k) \leq 5\mathsf{PROC}^{\mathcal{P}}_{\mathrm{ESA}}(k-1)$.

Proof: Suppose ESA creates m specializations when run for k inputs. Let g_i denote the ith specialization created, and k_i denote the number of inputs seen when g_i was created. For $z > 1$, g_z was created because the processing cost was "about to surpass" $\mathsf{OPT}(k_z)$. This implies two facts (based on condition 1):

Fact 1. $\mathsf{OPT}(k_z) < \mathsf{PROC}(k_z - 1) + p(g_{z-1})$.

Fact 2. $p(g_{z-1}) > p(f^*)$, where f^* is the method used by the optimal on k_z inputs.

Applying $\mathsf{OPT}(k_z) = c(f^*) + k_z p(f^*)$ to Fact 1 shows

$$c(f^*) < \mathsf{PROC}(g_z - 1) + p(g_{z-1}). \tag{3}$$

Fact 2 says ESA uses a less-than-optimal method for inputs up through $k_z - 1$. For $z > 2$, this implies f^* was not chosen at time k_{z-1} because its creation cost was too high at the time;

$$c(f^*) > \mathsf{PROC}(k_{z-1} - 1) + p(f^*). \tag{4}$$

Combining (3) and (4) and Fact 2 yields $2\mathsf{PROC}(k_{z-1} - 1) < \mathsf{PROC}(k_z - 1)$, for $z > 2$. Since the processing costs must at least double with each new specialization created, we know

$$\mathsf{PROC}(k_2 - 1) + \sum_{z=2}^{m} \mathsf{PROC}(k_z - 1) < 2\mathsf{PROC}(k_m - 1). \tag{5}$$

From the way that g_z is chosen at time k_z,

$$c(g_z) < 2\mathsf{PROC}(k_z - 1) + p(g_z). \tag{6}$$

Also $c(g_1) + p(g_1) = \mathsf{OPT}(1) < \mathsf{OPT}(k_2) \leq \mathsf{PROC}(k_2 - 1) + p(g_1)$ (using condition 1), so clearly

$$c(g_1) < \mathsf{PROC}(k_2 - 1). \tag{7}$$

Thus, our total creation costs can be bounded as follows:

$$\begin{array}{rcll} \mathsf{CRE}(k) & = & c(g_1) + \sum_{z=2}^{m} c(g_z) & \text{(defn. of } \mathsf{CRE}(k)) \\ & < & \mathsf{PROC}(k_2 - 1) + \sum_{z=2}^{m} (2\mathsf{PROC}(k_z - 1) + p(g_z)) & \text{(equations 7 and 6)} \\ & < & 4\mathsf{PROC}(k_m - 1) + \sum_{z=2}^{m} p(g_z) & \text{(equation 5)} \\ & < & 4\mathsf{PROC}(k_m - 1) + \mathsf{PROC}(k_m - 1) & \text{(each } g_z \text{ used at least once)} \\ & \leq & 5\mathsf{PROC}(k-1). & \end{array}$$

□

Corollary 3.6 ESA*is 6-competitive.*

Proof: By lemmas 3.2 and 3.5, we know

$$\mathsf{TOT}(k) = \mathsf{CRE}(k) + \mathsf{PROC}(k) \leq 5\mathsf{PROC}(k-1) + \mathsf{PROC}(k) \leq 6\mathsf{OPT}(k).$$

□

Lemma 3.7 *Let* $\mathrm{ES}(\mathcal{F}, c, p)$ *denote a problem where* $\mathcal{F}$ *has* n *specializations in the standard order, and let* $\mathcal{F}_\ell = \{f_1, f_2, ..., f_r\}$ *where* $r < n$*. Then*

$$\forall k > 0, \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(\mathcal{F},c,p)}(k) \leq \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(\mathcal{F}_\ell,c,p)}(k). \quad (8)$$

Furthermore, let $\mathcal{F}_r = \{f_r, f_{r+1}, ..., f_n\}$*, let* k^* *be the number of inputs that* ESA *must see on problem* $\mathrm{ES}(\mathcal{F}, c, p)$ *before it creates some specialization in* $\mathcal{F}_r$*, and let* k' *be the number of inputs that* ESA *must see on* $\mathrm{ES}(\mathcal{F}_\ell, c, p)$ *before it creates* f_r*. Then*

$$k' \geq k^*,$$

$$\forall k < k^*, \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(\mathcal{F},c,p)}(k) = \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(\mathcal{F}_\ell,c,p)}(k), \quad (9)$$

$$\forall k, k^* - 1 \leq k \leq k' - 1, \mathsf{CRE}_{\mathrm{ESA}}^{\mathrm{ES}(\mathcal{F}_\ell,c,p)}(k) = \mathsf{CRE}_{\mathrm{ESA}}^{\mathrm{ES}(\mathcal{F},c,p)}(k^* - 1). \quad (10)$$

Proof: Let A and A_ℓ denote the execution of ESA on $\mathrm{ES}(\mathcal{F}, c, p)$, and $\mathrm{ES}(\mathcal{F}_\ell, c, p)$, respectively. As long as the extra specializations available in execution A are too costly to be created by ESA, A and A_ℓ run in lock step and pay exactly the same costs. This is always true for $k < k^*$; thus equation (9) holds. The extra specializations in execution A may reduce the cost OPT as compared to execution A_ℓ; thus the condition (1) may become true earlier in execution A than in A_ℓ, but it can never come later. This shows that $k^* \leq k'$.

Once $k \geq k^*$, A creates and uses $f^* \in \mathcal{F}_r$ which may have lower processing cost than anything available to A_ℓ; this shows equation (8). By how f^* is chosen and equation (9),

$$c(f^*) < 2\mathsf{PROC}^{\mathrm{ES}(\mathcal{F},c,p)}(k^* - 1) + p(f^*) = 2\mathsf{PROC}^{\mathrm{ES}(\mathcal{F}_\ell,c,p)}(k^* - 1) + p(f^*). \quad (11)$$

From the ordering of $\mathcal{F}$, we know $c(f_r) \leq c(f^*)$ and $p(f^*) \leq p(f_r)$. Applying this to equation (11) yields $c(f_r) \leq 2\mathsf{PROC}^{\mathrm{ES}(\mathcal{F}_\ell,c,p)}(k^* - 1) + p(f_r)$. This implies that after the k^*th input in A_ℓ, f_r will be the next method created once method creation is allowed. Since this happens at input k', no additional methods are created by A_ℓ when running for up to $k' - 1$ inputs; this shows equation (10). □

4 Design and analysis of MLSA

4.1 *Overview of* MLSA

The two main ideas for solving MLS are partitioning $\mathcal{F}$ into subintervals, and using the behavior of ESA on a subset of the inputs to determine which specializations to create. MLSA consists of one manager, and many workers. The manager routes each input to a single worker who then processes the input. The manager also creates and destroys workers, as necessary. Each worker processes the inputs that it is given completely independently of all other workers.

Each worker is defined by a tuple of integers, (q, r, s), where $q \leq r \leq s$. The worker only knows about specializations in the set $\{f_i : q \leq i \leq s\}$ (abbreviated $[f_q, f_s]$); it cannot create any other specializations. The worker is only given inputs whose specialization level f_i satisfies $f_i \in [f_r, f_s]$. A worker's goal is to create f_r which is typically in the middle of its interval. On the way to creating f_r the worker runs ESA as a subroutine to figure out when it is necessary to create methods in $[f_q, f_{r-1}]$. Once f_r is created, the worker quits.

The manager maintains the partition invariant that if $\{(q_i, r_i, s_i)\}$ is the set of all current workers, then the sets $[f_{r_i}, f_{s_i}]$ form a partition of $\mathcal{F}$. With this invariant, it is a simple job to route the incoming inputs to the appropriate worker. Whenever a worker quits, the manager creates new workers in a manner that maintains the partition invariant.

Replacement workers are chosen to overcome the bad performance found in section 1.3. As an example, one of the initial workers is $(1, \lceil \frac{n+3}{2} \rceil, n)$. This implies $f_{\lceil \frac{n+3}{2} \rceil}$ is always created before any worker has a chance to create f_n. Our algorithm does not create a highly-specialized method right away even when the initial inputs are all of specialization level f_n.

4.2 MLSA *Manager*

The manager's two responsibilities are creating workers and routing inputs to the appropriate worker. The manager creates workers so that they partition $\{1, 2, ..., n\}$ into contiguous subsets; a worker (q, r, s) "covers" the interval from r to s, in the following sense: every input I_k is routed to the worker (q, r, s) such that $\mathsf{lev}(I_k) \in [f_r, f_s]$. Whenever a worker (q, r, s) quits, the manager replaces that worker with several new ones that cover the interval from r to s, as depicted in figure 1.

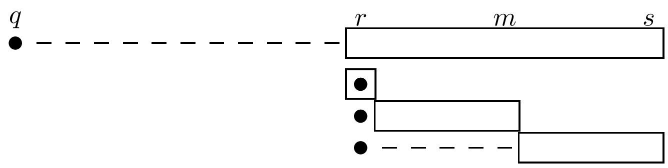
Suppose the worker $W = (q, r, s)$ has just quit, and let $m = \lceil (r + s)/2 \rceil$. The following three workers are created to replace it: (r, r, r), $(r, r + 1, m)$, $(r, m + 1, s)$. Note however, that the second and third workers listed are not created when the interval they cover is empty. We use the term *created worker* to refer to any worker created by MLSA when processing all the input (including the workers that have quit).

Initially, the manager creates f_1. It then follows the above procedure as if worker $(1, 1, n)$ had just quit.

Note that just before a worker (q, r, s) quits, it creates f_r. This fact and the way the manager creates workers imply the following invariants.

- f_q is created before worker (q, r, s) is created.
- Let $\mathscr{W}$ be the set of current workers that have not yet quit. For all $i, 1 \leq i \leq n$, there is only one worker (q, r, s) in $\mathscr{W}$ such that $r \leq i \leq s$.
- A created worker (q, r, s) always satisfies either $q = r = s$ or $q < r \leq s$.

Fig. 1: Worker (q, r, s) followed by its three replacement workers. Circles represent a specialization created before the worker starts. Boxes represent the set of specialization levels handled by the worker.



4.3 MLSA *Workers*

Each worker uses a private array N to track the number of inputs of each specialization level, initially set to all zeroes. This array is local to the worker and not shared between workers. Let (q, r, s) be a worker. An invariant maintained is that when a worker processes input I, $N[k]$ represents the number of inputs with specialization level f_k that the worker has seen, including the current input I.

We know either $(q = r = s)$ or $q < r \leq s$. A worker with $q = r = s$ uses f_q to process all its input and never quits. A worker with $q < r \leq s$ processes input I of specialization level f_j by first incrementing its private variable $N[j]$. It then performs two additional steps to process I: the *quit* step and the *process* step. The quit step checks to see whether or not the worker should quit; if it decides to quit, then f_r is created, and input I remains unprocessed (it is processed by one of its replacement workers). The process step decides the specialization f^* to be used on I; f^* is created if necessary. The decisions in both steps are made using calls to ESA with a subset of the methods available to the worker. The calls to ESA are all made with set-up costs c' defined so that $c'(f_q) = 0$ and $c'(f) = c(f)$ for all $f \neq f_q$.

In the quit step, the worker examines the behavior of ESA on many problems. Let

$$S = \{f : \text{ for some } i, r \leq i \leq s, f \text{ is the method that ESA uses on} \\ \text{ES}([f_q, f_i], c', p) \text{ when run for } \sum_{j=i}^{s} N[j] \text{ inputs}\}.$$

If there is a specialization $f \in S$ such that $f_r \preceq f$, then the worker creates f_r and then quits.

In the process step, the worker decides to use the specialization f^* that ESA uses on $\text{ES}([f_q, f_r], c', p)$ when run for $\sum_{j=r}^{s} N[j]$ inputs.

4.4 Analysis of MLSA

We separately bound the online processing and creation costs paid by MLSA relative to the total cost paid by the optimal offline algorithm. In particular, we prove the following theorems:

Theorem 4.1 *On problem* $\mathcal{P} = \text{MLS}(\mathcal{F}, c, p, \prec)$*, if* $n \geq 2$*, then*

$$\forall \sigma, \mathsf{PROC}^{\mathcal{P}}_{\text{MLSA}}(\sigma) \leq (2 \log n)\mathsf{OPT}^{\mathcal{P}}(\sigma).$$

Theorem 4.2 *On problem* $\mathcal{P} = \text{MLS}(\mathcal{F}, c, p, \prec)$*, if* $n \geq 2$*, then*

$$\forall \sigma, \mathsf{CRE}^{\mathcal{P}}_{\text{MLSA}}(\sigma) \leq (1 + 11 \log n)\mathsf{OPT}^{\mathcal{P}}(\sigma).$$

These two theorems prove theorem 2.1.

Our overall strategy for the processing cost bound charges the processing cost MLSA pays for each input I against the optimal cost for processing I. Let $\mathscr{I}$ be the set of all inputs processed with f_i in the optimal solution; $B = c(f_i) + |\mathscr{I}|p(f_i)$ represents the cost the optimal pays for processing $\mathscr{I}$. We show that the processing cost paid by MLSA for $\mathscr{I}$ is at most $(2\log n)B$. We derive this bound by separately bounding the total number of workers that could process inputs in $\mathscr{I}$, and the cost contributed by any one worker for inputs in $\mathscr{I}$. Our strategy for the creation cost bound is to bound the creation cost paid by each worker by a constant factor of the processing cost paid by the worker. We then reuse our bound of the processing cost relative to the optimal to get a bound of the creation cost relative to the optimal.

We now define terms for describing the subdivision of the costs involved, describe lemmas on the behavior of MLSA, and finally combine these lemmas in the proofs of the theorems.

4.5 Definitions

In describing the cost of MLSA, σ refers to the input sequence, and the $\mathcal{P} = \mathrm{MLS}(\mathcal{F}, c, p, \prec)$. Both σ and $\mathcal{P}$ are fixed for the rest of this section. The definitions below are abbreviated in that they implicitly depend on these two values.

- Let $V = [f_v, f_z] = \{f_v, f_{v+1}, ..., f_z\}$. We say V is an *optimal interval* if in the optimal algorithm, all inputs in σ whose specialization level is in V are processed by f_v.
- When V is an optimal interval, $\mathscr{I}(V)$ denotes the set of inputs from σ that are processed with f_v by the optimal offline algorithm.
- $\mathsf{OPTINT}(V) = c(f_v) + p(f_v)|\mathscr{I}(V)|$. By definition, summing $\mathsf{OPTINT}(V)$ over all optimal intervals yields OPT.
- If $\mathscr{I}$ is a subset of inputs from σ, then $\mathsf{PROCSUB}(\mathscr{I})$ denotes the processing cost that MLSA paid for inputs in $\mathscr{I}$, when MLSA was run on entire input sequence σ.
- Let W be a created worker. Then $\mathscr{I}(W) = \{I : I \text{ is in } \sigma \text{ and } W \text{ processed } I\}$.
- $\mathsf{CRESUB}(W)$ denotes the sum of the cost of the methods created by worker W. By definition, if $\mathscr{W}$ is the set of all created workers, then

$$\mathsf{CRE} = c(f_1) + \sum_{W \in \mathscr{W}} \mathsf{CRESUB}(W).$$

- Applying the previous definitions, if $\mathscr{W}$ and $\mathscr{V}$ are the set of all created workers and optimal intervals, respectively, then

$$\mathsf{PROC} = \sum_{V \in \mathscr{V}} \sum_{W \in \mathscr{W}} \mathsf{PROCSUB}(\mathscr{I}(W) \cap \mathscr{I}(V)).$$

We also use the following notation to describe the ES problems that MLSA simulates.

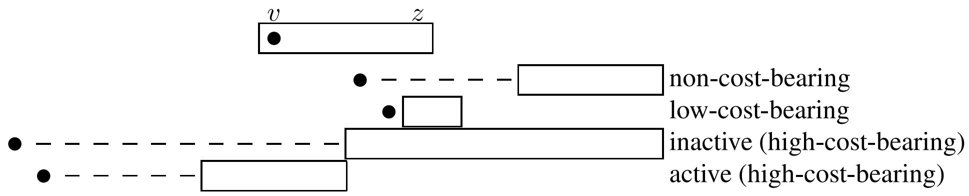
- Let $\mathrm{ES}(f_i, f_j)$ denote $\mathrm{ES}([f_i, f_j], c', p)$. Note that $c' = c$ except for $c'(f_i) = 0$. Workers simulate many problems of this form to determine when to quit.

- If $W = (q, r, s)$, then $\mathrm{ES}(W) = \mathrm{ES}(f_q, f_r)$; this is the problem W simulates in the process step.

We now define several different categories of workers as illustrated by figure 2. We show different cost bounds on each category in order to prove our main theorems. Let $V = [f_v, f_z]$ be an optimal interval, and $W = (q, r, s)$ be a created worker.

- We say W is a *cost-bearing* worker of V if $[f_r, f_s] \cap V \neq \emptyset$. Cost-bearing workers of V charge some portion of their processing cost onto V in our analysis.
- We say W is a *low-cost-bearing* worker of V if it is cost-bearing and $f_q \succeq f_v$. We say W is a *high-cost-bearing* worker of V if it is cost-bearing and $f_q \prec f_v$. Unlike high-cost-bearing workers, low-cost-bearing workers pay per-input processing costs no worse than that of the optimal algorithm.
- We say W is an *active* worker of V if it is cost-bearing and $f_q \prec f_r \prec f_v$. W is an *inactive* worker of V if it is cost-bearing and $f_q \prec f_v \preceq f_r$. We later show that inactive workers of V who quit are replaced by low-cost-bearing or non-cost-bearing workers, while active workers who quit can be replaced by high-cost-bearing workers. Note that active and inactive workers of V are by definition high-cost-bearing workers of V. Since our algorithm never creates workers where $q = r < s$, all high-cost-bearing workers of V are either active or inactive.

Fig. 2: Optimal Interval $[f_v, f_z]$ followed by four different categories of created workers.



4.6 Processing cost bound

Let V be an optimal interval and $W = (q, r, s)$ a cost-bearing worker of V. We bound $\mathsf{PROCSUB}(\mathscr{I}(W) \cap \mathscr{I}(V))$ by the corresponding cost incurred by ESA on $\mathrm{ES}(W)$. We then show a bound on the total cost charged against $\mathsf{OPTINT}(V)$.

Lemma 4.3 *Let W be a created worker and $V = \{f_v, f_{v+1}, ..., f_z\}$ be an optimal interval. Let $k = |\mathscr{I}(W) \cap \mathscr{I}(V)|$. Then we know the following about $\mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(k)$:*

1. *If W is a non-cost-bearing worker of V, then $\mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(k) = 0$.*
2. *If W is a low-cost-bearing worker of V, then $\mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(k) \leq kp(f_v)$.*
3. *If W is a high-cost-bearing worker of V, then $\mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(k) \leq c(f_v) + kp(f_v)$.*

Proof: Let $W = (q, r, s)$. If W is non-cost-bearing then $k = 0$; this shows statement 1. If W is low-cost-bearing then $f_v \preceq f_q$ implying $p(f_q) \leq p(f_v)$. Seeing that ESA pays at most $p(f_q)$ for each input yields statement 2.

For statement 3, we need to show that

$$\mathsf{PROC}^{\mathrm{ES}(f_q, f_r)}(k) \leq \mathsf{PROC}^{\mathrm{ES}(f_q, f_v)}(k). \tag{12}$$

Assuming equation (12) is true, we apply lemma 3.2 and the fact that the specific method of using f_v for k inputs is certainly no better than optimal method on k inputs; this yields

$$\mathsf{PROC}^{\mathrm{ES}(f_q, f_r)}(k) \leq \mathsf{PROC}^{\mathrm{ES}(f_q, f_v)}(k) \leq \mathsf{OPT}^{\mathrm{ES}(f_q, f_v)}(k) \leq c(f_v) + kp(f_v).$$

To show equation (12), we consider the cases where W is inactive and W is active separately:

Case I. If W is inactive, then $f_q \prec f_v \preceq f_r$. Equation (8) implies equation (12).

Case II. If W is active, then $f_q \preceq f_r \prec f_v$. Let k^* denote the number of inputs ESA must see on $\mathrm{ES}(f_q, f_v)$ before it creates some specialization in $[f_r, f_v]$. By the quitting condition of W, $\sum_{j=v}^{s} N[j] \leq k^*$, where N refers to the value of the local array in W. Note that equality can hold only if W quits after seeing (but not processing) its k^*th input. Since k counts only the inputs processed by W in $[f_v, f_{\min(s,z)}]$, $k < k^*$. Applying equation (9) of lemma 3.7 yields equation (12).

□

Lemma 4.4 *Let W be a created worker, and let $\mathcal{S}$ be an arbitrary subset of $\mathscr{I}(W)$. Then*

$$\mathsf{PROCSUB}(\mathcal{S}) \leq \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathcal{S}|).$$

Proof: Let $\mathcal{S}'$ be the first $|\mathcal{S}|$ inputs that processed by W. Since W imitates the behavior of ESA on $\mathrm{ES}(W)$, $\mathsf{PROCSUB}(\mathcal{S}') = \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathcal{S}|)$. We also know the per-input processing cost for worker W decreases over time just as it does in ESA. Thus $\mathsf{PROCSUB}(\mathcal{S}) \leq \mathsf{PROCSUB}(\mathcal{S}')$. □

Lemma 4.5 *Let V be an optimal interval. Let W be a non-active worker of V who quits, and let W' be a new worker created to replace it. Then W' is a low-cost-bearing or non-cost-bearing worker of V.*

Proof: Since any replacement worker for $W = (q, r, s)$ is within the interval from r to s, the replacement rules (figure 1) applied to the relevant worker types (figure 2) show the following:

- Non-cost-bearing workers are replaced with non-cost-bearing workers.
- Low-cost-bearing and inactive workers are replaced with one low-cost-bearing worker and up to two additional non-cost-bearing or low-cost-bearing workers.

□

Lemma 4.6 *Let V be an optimal interval. Let W be an active worker of V. Suppose W quits and is replaced by new workers. Then one of the following conditions holds:*

1. *There are 2 replacement workers W_1 and W_2, where W_1 is a non-cost-bearing worker of V, and W_2 is an inactive worker of V.*

2. *There are 3 replacement workers W_1, W_2, and W_3. W_1 is not a non-cost-bearing worker of V, and at most one of W_2 and W_3 is an active worker of V. Furthermore, let $|W|$ (resp. $|W_i|$) denote the number of input types W (W_i) is responsible for. Then $|W_2| \leq |W|/2$, and $|W_3| \leq |W|/2$.*

Proof: Let $W = (q, r, s)$ be the active worker who quit, and let $V = [f_v, f_z]$ be an optimal interval. Since W is an active worker of V,

$$f_q \prec f_r \prec f_v \preceq f_s. \tag{13}$$

This implies $r < s$. We now consider the manager's behavior on the remaining two cases:

Case I. Suppose $r + 1 = s$. Since the manager sets $m = \lceil (r + s)/2 \rceil = r + 1$, we know two workers $W_1 = (r, r, r)$ and $W_2 = (r, r + 1, r + 1)$ are created, and $f_v = f_{r+1}$ to satisfy equation (13). Thus W_2 is an inactive worker of V, and W_1 is a non-cost-bearing worker of V, satisfying condition 1 of the lemma.

Case II. Suppose $r + 1 < s$. Then $m = \lceil (r + s)/2 \rceil$, where $r < m < s$. Thus three workers $W_1 = (r, r, r)$, $W_2 = (r, r + 1, m)$, and $W_3 = (r, m + 1, s)$ are created, as illustrated in figure 3. We know W_1 is a non-cost-bearing worker of V. We now consider the following cases:

(a) If $v \leq m$, then W_2 is active unless $v = r + 1$, and W_3 is inactive or non-cost-bearing.

(b) If $v > m$, then W_2 non-cost-bearing and W_3 is active unless $v = m + 1$.

In both IIa and IIb, we have at most one of W_2 or W_3 is an active worker of V. Thus case II satisfies condition 2, where the method of choosing m ensures the size bound.

□

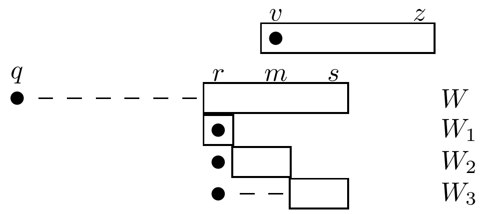
Corollary 4.7 *Let V be an optimal interval. Let W be an active worker of V, and let $|W|$ denote the number of input types W is responsible for. Then $|W| \geq 2$.*

Proof: For active worker (q, r, s), the proof of lemma 4.6 starts by showing $r < s$. □

Lemma 4.8 *If V is an optimal interval, then there are at most $2 \log n$ high-cost-bearing workers of V.*

Proof: Lemma 4.5 says only active workers of V that quit can be replaced with high-cost-bearing workers of V, and that other workers that quit are not replaced with high-cost-bearing workers. By lemma 4.6, the initial workers created by the manager include at most 2 high-cost-bearing workers of V, at most one of which is active. Every time an active worker quits, its replacement workers include at most one active and at most one inactive worker of V. Let k be the number of active workers of V created. Let W_1,W_2, ..., W_k be the active workers of V, in order that they are created. There are at most $k + 1$ inactive workers:

Fig. 3: Optimal Interval $[f_v, f_z]$, an active worker W, and its replacements.



one for each inactive worker of V that was created at the same time as W_i, and one for the inactive worker that could replace W_k. Thus there are at most $2k+1$ high-cost-bearing workers of V.

Now let $|W_i|$ denote the number of input types W_i is responsible for. By lemma 4.6, for $1 \le i \le k-1$, $|W_i|/2 \ge |W_{i+1}|$. Corollary 4.7 shows $|W_k| \ge 2$. Since $|W_1| \le n/2$, we conclude that $k \le (\log n) - 1$. This implies there are at most $2\log n$ high-cost-bearing workers of V. □

Lemma 4.9 *Let $\mathscr{W}$ be the set of all created workers, let V be an optimal interval. Then if $n \ge 2$,*

$$\begin{aligned}\sum_{W\in\mathscr{W}} \mathsf{PROCSUB}(\mathscr{I}(W)\cap\mathscr{I}(V)) &\le \sum_{W\in\mathscr{W}} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)\cap\mathscr{I}(V)|)\\ &\le (2\log n)\mathsf{OPTINT}(V).\end{aligned}$$

Proof: Lemma 4.4 implies the first inequality of this lemma. Let $\mathscr{W} = \mathscr{W}_1 \cup \mathscr{W}_2 \cup \mathscr{W}_3$, where $\mathscr{W}_1$ contains non-cost-bearing workers of V, $\mathscr{W}_2$ contains low-cost-bearing workers of V, and $\mathscr{W}_3$ contains high-cost-bearing workers of V. From lemma 4.3, we have a bound on $\mathsf{PROC}^{\mathrm{ES}(W)}(|\mathscr{I}(W)\cap\mathscr{I}(V)|)$ for each $W \in \mathscr{W}_i$. Lemma 4.8 tells us $|\mathscr{W}_3| \le 2\log n$. We conclude that

$$\begin{aligned}\sum_{W\in\mathscr{W}} \mathsf{PROCSUB}(\mathscr{I}(W)\cap\mathscr{I}(V)) &= \sum_{i=1}^{3}\sum_{W\in\mathscr{W}_i} \mathsf{PROCSUB}(\mathscr{I}(W)\cap\mathscr{I}(V))\\ &\le \sum_{i=1}^{3}\sum_{W\in\mathscr{W}_i} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)\cap\mathscr{I}(V)|)\\ &\le 0 + \sum_{W\in\mathscr{W}_2} p(f_v)|\mathscr{I}(W)\cap\mathscr{I}(V)| + \sum_{W\in\mathscr{W}_3}(c(f_v)+p(f_v)|\mathscr{I}(W)\cap\mathscr{I}(V)|)\\ &= \sum_{W\in\mathscr{W}_2\cup\mathscr{W}_3} p(f_v)|\mathscr{I}(W)\cap\mathscr{I}(V)| + \sum_{W\in\mathscr{W}_3} c(f_v)\\ &\le p_v|\mathscr{I}(V)| + (2\log n)c_v.\end{aligned}$$

If $n \ge 2$, then $p_v|\mathscr{I}(V)| \le (2\log n)p_v|\mathscr{I}(V)|$, and we get

$$p_v|\mathscr{I}(V)| + (2\log n)c_v \le (2\log n)(p_v|\mathscr{I}(V)| + c_v) = (2\log n)\mathsf{OPTINT}(V).$$

□

Theorem 4.10 (4.1) *On problem $\mathcal{P} = \mathrm{MLS}(\mathcal{F}, c, p, \prec)$, if $n \geq 2$, then*

$$\forall \sigma, \mathsf{PROC}^{\mathcal{P}}_{\mathrm{MLSA}}(\sigma) \leq (2\log n)\mathsf{OPT}^{\mathcal{P}}(\sigma).$$

Proof: Let $\mathcal{V}$ be the set of all the optimal intervals, and Let $\mathscr{W}$ be the set of all the created workers. Now simply sum over all optimal intervals and apply lemma 4.9, obtaining

$$\begin{aligned}
\mathsf{PROC}^{\mathcal{P}}_{\mathrm{MLSA}}(\sigma) &= \sum_{V \in \mathcal{V}} \sum_{W \in \mathscr{W}} \mathsf{PROCSUB}(\mathscr{I}(W) \cap \mathscr{I}(V)) \\
&\leq \sum_{V \in \mathcal{V}} (2\log n)\mathsf{OPTINT}(V) \\
&= (2\log n)\mathsf{OPT}^{\mathcal{P}}(\sigma).
\end{aligned}$$

$\square$

4.7 Creation cost bound

Our strategy is to bound our creation cost of any created worker by the processing cost of ESA on the corresponding simulated ES problem. These processing costs then be summed to get a total creation cost bound relative to the optimal cost for the MLS problem.

Lemma 4.11 *Let W be a created worker that quits. Then*

$$\mathsf{CRESUB}(W) \leq 5\mathsf{PROC}^{\mathrm{ES}(W)}_{\mathrm{ESA}}(|\mathscr{I}(W)|).$$

Proof: By the quitting condition of W, we can pick an i and an h that satisfy the following conditions: 1) $r \leq h \leq i \leq s$ and 2) ESA creates f_h when run on $\mathrm{ES}(f_q, f_i)$ for $\sum_{j=i}^{s} N[j]$ inputs. Let k^* be the number of inputs ESA needs on $\mathrm{ES}(f_q, f_i)$ to create some specialization in $[f_r, f_i]$; let k' be the number of inputs ESA needs on $\mathrm{ES}(f_q, f_r)$ to create f_r. We derive the following facts.

Fact 1.	$\sum_{j=i}^{s} N[j]$	$=$	k^*,	by how i is chosen and the quitting condition.
Fact 2.	$-1 + \sum_{j=r}^{s} N[j]$	$<$	k',	since the quitting condition was false before the k^*th input.
Fact 3.	$-1 + \sum_{j=r}^{s} N[j]$	$=$	$\lvert\mathscr{I}(W)\rvert$,	since $\lvert\mathscr{I}(W)\rvert$ only counts processed inputs.
Fact 4.	$\lvert\mathscr{I}(W)\rvert$	$<$	k',	combining facts 2 and 3.
Fact 5.	$k^* - 1$	$\leq$	$\lvert\mathscr{I}(W)\rvert$,	combining facts 1 and 3.

Then we know

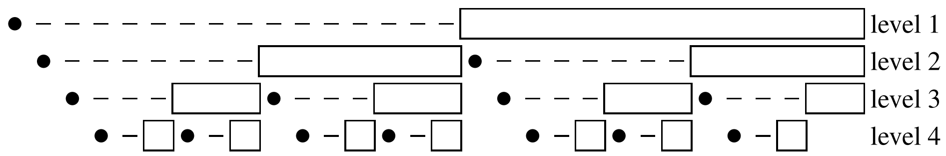
$$\begin{aligned}
\mathsf{CRESUB}(W) &= \mathsf{CRE}^{\mathrm{ES}(f_q,f_r)}(|\mathscr{I}(W)|) + c_r && \text{(since } W \text{ simulates ESA on ES}(W) \text{ and creates } f_r \text{ before quitting)} \\
&= \mathsf{CRE}^{\mathrm{ES}(f_q,f_i)}(k^* - 1) + c_r && \text{(by } |\mathscr{I}(W)| < k' \text{ and eq. 10 of lemma 3.7)} \\
&\leq \mathsf{CRE}^{\mathrm{ES}(f_q,f_i)}(k^* - 1) + c(f_h) && \text{(since } f_h \succeq f_r\text{)} \\
&= \mathsf{CRE}^{\mathrm{ES}(f_q,f_i)}(k^*) && \text{(by defn. of } k^*, i, \text{ and } h\text{)} \\
&\leq 5\mathsf{PROC}^{\mathrm{ES}(f_q,f_i)}(k^* - 1) && \text{(by lemma 3.5)} \\
&= 5\mathsf{PROC}^{\mathrm{ES}(f_q,f_r)}(k^* - 1) && \text{(by eq. 9 of lemma 3.7)} \\
&\leq 5\mathsf{PROC}^{\mathrm{ES}(f_q,f_r)}(|\mathscr{I}(W)|). && \text{(since } k^* - 1 \leq |\mathscr{I}(W)|\text{)}
\end{aligned}$$

□

Lemma 4.12 *For any fixed* $i, 1 \leq i \leq n$*, there are at most* $\log n$ *created workers (q,r,s) satisfying* $q < i < r$.

Proof: Consider the tree where each node is a created worker, and the children of node S are the workers created to replace the worker of node S. The root of this tree represents the manager and its children are the initial workers created. Let $\mathscr{W}_k$ denote all workers (q, r, s) at depth k in this tree that satisfy $q+1 < r$. In other words, $\mathscr{W}_k$ are all nodes at level k in the tree that are "relevant" in that they satisfy $q < i' < r$ for some integer i'. If $W = (q, r, s)$ and $W' = (q', r', s')$ are two workers in $\mathscr{W}$, we can see that the intervals $[q+1, r-1]$ and $[q'+1, r'-1]$ are disjoint due to the way manager replaces workers. We can also see that with each successive level, the interval size decreases by a factor of two. These two facts are illustrated in figure 4. This means that there is at most one created worker on each level satisfying the conditions of the lemma, and that there are at most $\log n$ total levels in the tree. □

Fig. 4: Relevant workers at many different levels of the tree.



Lemma 4.13 *There are at most* $\log n$ *created workers* W *that satisfy*

$$\mathsf{CRESUB}(W) > 5\mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)|). \tag{14}$$

Furthermore, each of these workers also satisfy $\mathsf{CRESUB}(W) < \mathsf{OPT}(\sigma)$.

Proof: Let $\mathscr{W}$ be the set of workers satisfying equation (14), and let $W \in \mathscr{W}$. From lemma 4.11, we know that W must be a created worker that does not quit. By the way the worker simulates ESA, $\mathsf{CRESUB}(W) = \mathsf{CRE}^{\mathrm{ES}(W)}(|\mathscr{I}(W)|)$. Lemma 3.5 shows that any worker which uses at least 2 specializations and does not quit cannot belong in $\mathscr{W}$; thus W must use only one specialization. Thus $\mathsf{CRESUB}(W) = c'(f_k)$, where f_k is the specialization that W uses for its first input. Let $W = (q, r, s)$. Then we know $k < r$, because otherwise W would have quit. We also know $k > q$, because if $k = q$, then $\mathsf{CRESUB}(W) = 0$, meaning $W \notin \mathscr{W}$.

Let f_i denote the first specialization where $p_i < c_i$. This means $p_j < c_j$ for all $j \geq i$, and $p_j \geq c_j$ for all $j < i$. We examine the possible workers in three cases.

Case I. $i \leq q$. This implies $p_q < c_q$. Since f_k was chosen to minimize the cost of processing one input, we know $c_k + p_k < p_q$; this results in the contradiction that $c_k < c_q$; thus this case cannot occur.

Case II. $q < i < r$. Since the optimal must pay at least $c_k + p_k$ to process one input in $[f_r, f_s]$, clearly $\mathsf{CRESUB}(W) < \mathsf{OPT}$. Lemma 4.12 bounds the number of workers in this case by $\log n$.

Case III. $i \geq r$. This implies $i > k$, which means that $p_k > c_k$, and that $\mathsf{CRESUB}(W) = c_k < p_k = \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(1)$, which would contradict $W \in \mathscr{W}$. Thus, this case cannot occur.

Thus there are a total of at most $\log n$ workers satisfying equation (14), and each one also satisfies $\mathsf{CRESUB}(W) < \mathsf{OPT}(\sigma)$. □

Lemma 4.14 *Let $\mathcal{P}$ denote* $\mathrm{MLS}(\mathcal{F}, c, p, \prec)$. *Let $\mathscr{W}$ be the set of all created workers. Then if $n \geq 2$,*

$$\sum_{W \in \mathscr{W}} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)|) \leq (2 \log n)\mathsf{OPT}^{\mathcal{P}}(\sigma).$$

Proof: Let $\mathcal{V}$ be the set of optimal intervals. For any given W we can apply lemma 3.4 to get

$$\mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)|) \leq \sum_{V \in \mathcal{V}} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W) \cap \mathscr{I}(V)|).$$

If we sum over all $W \in \mathscr{W}$ and then apply lemma 4.9, we get

$$\begin{aligned}
\sum_{W \in \mathscr{W}} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)|) &\leq \sum_{W \in \mathscr{W}} \sum_{V \in \mathcal{V}} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W) \cap \mathscr{I}(V)|) \\
&= \sum_{V \in \mathcal{V}} \sum_{W \in \mathscr{W}} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W) \cap \mathscr{I}(V)|) \\
&\leq \sum_{V \in \mathcal{V}} (2 \log n)\mathsf{OPTINT}(V) \\
&= (2 \log n)\mathsf{OPT}(\sigma).
\end{aligned}$$

□

Theorem 4.15 (4.2) *On problem $\mathcal{P} = \mathrm{MLS}(\mathcal{F}, c, p, \prec)$, if $n \geq 2$, then*

$$\forall \sigma, \mathsf{CRE}_{\mathrm{MLSA}}^{\mathcal{P}}(\sigma) \leq (1 + 11 \log n)\mathsf{OPT}^{\mathcal{P}}(\sigma).$$

Proof: Let $\mathscr{W}$ denote all created workers W satisfying $\mathsf{CRESUB}(W) < 5\mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)|)$ and $\mathscr{W}'$ denote all created workers not satisfying the previous relation. Then applying lemma 4.13 followed by lemma 4.14 yields

$$\begin{aligned}
\mathsf{CRE}(\sigma) &= c(f_1) + \sum_{W \in \mathscr{W}'} \mathsf{CRESUB}(W) + \sum_{W \in \mathscr{W}} \mathsf{CRESUB}(W) \\
&\leq \mathsf{OPT}(\sigma) + (\log n)\mathsf{OPT}(\sigma) + 5 \sum_{W \in \mathscr{W}} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)|) \\
&\leq (1 + \log n)\mathsf{OPT}(\sigma) + 5 \sum_{W \in \mathscr{W} \cup \mathscr{W}'} \mathsf{PROC}_{\mathrm{ESA}}^{\mathrm{ES}(W)}(|\mathscr{I}(W)|) \\
&\leq (1 + \log n)\mathsf{OPT}(\sigma) + (10 \log n)\mathsf{OPT}(\sigma) \\
&= (1 + 11 \log n)\mathsf{OPT}(\sigma).
\end{aligned}$$

□

5 Proof of lower bound

In this section, we describe an adversary that shows any online algorithm for the MLS problem has competitive ratio at least $\Omega(\log n)$. We consider the case where the following is true:

- $\prec$ is a total ordering on $\mathcal{F}$, so $\forall i > j, f_j \prec f_i$.
- $\forall i \geq 1, c_i = 1$.
- $\forall i \geq 1, p_i = 2^{-i}$.

A description of the adversary is as follows: Start with $S = \mathcal{F}$, and give inputs with specialization level f_n. This continues until the online algorithm decides to create a specialization f_k. f_k divides S into two sets: $\{f_1, f_2 ..., f_{k-1}\}$, and $\{f_k, f_{k+1}, ..., f_n\}$. Pick the larger set, and recursively apply this adversary to it. In other words, if $k - 1 > n/2$, then start giving inputs with specialization level f_{k-1}; otherwise continue giving inputs with specialization level f_n. Recursive calls continue while there are at least h methods in set S. We later choose h to maximize our lower bound.

This adversary is designed to maximize both the processing cost and the creation cost that the online algorithm pays in relation to the cost of the optimal solution. The key idea behind the adversary is that the online algorithm does not get to effectively use the specializations that it creates; any time it creates a specialization, the next input is chosen so that either that specialization does not get used, or it gets used but a more specialized version would have been better. The creation cost is maximized in that we expect the algorithm to create logarithmically many specializations.

In order to analyze the behavior of an online algorithm $\mathcal{A}$ on our adversary for the MLS problem, we provide the following definitions and invariants:

- I_k denotes the kth input given to $\mathcal{A}$ by the adversary.
- Define variables ℓ and r so that $\{f_{\ell+1}, f_{\ell+2}, \ldots, f_r\}$ is the contiguous set of methods that the adversary tracks as the online algorithm runs; these are methods which are not yet used by $\mathcal{A}$. We use ℓ_k and r_k to denote the values of ℓ and r, respectively, at the time just before the adversary has given the algorithm I_k. Initially, $\ell_1 = 0$ and $r_1 = n$. By definition, the adversary chooses I_k to have specialization level f_{r_k}.
- For all $k > 1, \ell_k \leq \ell_{k+1}$.
- For all $k > 1, \ell_k + h \leq r_k$.
- For all $k > 1, r_{k+1} \leq r_k$.
- Let m denote the total number of methods that $\mathcal{A}$ creates.

Lemma 5.1 *Let $\mathcal{A}$ be an online algorithm given t inputs from the adversary. Let $\ell = \ell_t$ and $r = r_t$ be the left and right boundaries for the uncreated methods of $\mathcal{A}$. Then on the t inputs,*

- *the optimal algorithm pays at most $1 + t2^{-r}$, and*
- *$\mathcal{A}$ pays at least $2^{-\ell}(t - m) + m$.*

Proof: We know from the basic facts that all t inputs have specialization level $\mathsf{lev}(I_t)$ or higher, meaning they can all be processed with $f_{r_t} = f_r$ at total cost $c_r + tp_r = 1 + t2^{-r}$; this bounds the amount the optimal pays. Now consider the costs for $\mathcal{A}$ as it processes input I_k. If it does not create a method, then by the basic fact invariants, it must use a method less specialized than f_{ℓ_k+1}. Thus it pays at least $2^{-\ell_k} \geq 2^{-\ell}$. If it does create a method, then it pays at least 1. Thus, the total cost is at least m for the methods created plus $(t-m)2^{-\ell}$ for processing the $t-m$ inputs that did not trigger method creation. □

Lemma 5.2 *Suppose that the adversary never stops giving the online algorithm $\mathcal{A}$ inputs. Then the competitive ratio for $\mathcal{A}$ must be at least 2^h.*

Proof: Because of the invariant $\ell + h \leq r$, we know that the adversary is paying at least 2^h times more than the optimal for processing each input. Since there are infinitely many inputs, creation costs are insignificant. □

Lemma 5.3 *Suppose that the adversary eventually stops giving $\mathcal{A}$ inputs. Then $m \geq \lfloor \log(n/h) \rfloor$.*

Proof: Define $s_k = r_k - l_k$ to be the size of the interval when producing input I_k. Each interval size change corresponds to the online algorithm creating at least one method. Every time the interval size changes, we know that the size of the new interval is at least half of the size of the old interval. This means it takes at least $\lfloor \log(n/h) \rfloor$ size changes before the interval size drops below h. Since the adversary does not stop until the size drops below h, we know $\mathcal{A}$ makes at least $\lfloor \log(n/h) \rfloor$ size changes; this is a lower bound on m. □

Lemma 5.4 *Suppose that the adversary eventually stops giving the online algorithm inputs. Then the competitive ratio for this algorithm must be at least* $\min(\log\lfloor n/h \rfloor, 2^h)/2$.

Proof: Lemma 5.1 gives us a lower bound of

$$\frac{m + (t-m)2^{-\ell}}{1 + t2^{-r}}$$

on the competitive ratio, where t is the total number of inputs given to the online algorithm. Since $\ell + h \leq r$ from our basic invariants, $t2^{-r} \leq t2^{-\ell-h}$, so we can rewrite our lower bound as

$$\frac{m + (t-m)2^{-\ell}}{1 + t2^{-\ell-h}}.$$

Case I. Suppose $t2^{-\ell-h} \leq 1$. Then certainly the denominator of our lower bound is less than 2, and the competitive ratio is therefore at least $m/2$.

Case II. Suppose $t2^{-\ell-h} > 1$. This implies the denominator of the lower bound is no more than $2t2^{-\ell-h}$. Define $\alpha = m/t$, so that $m = t\alpha$. Since $m \geq 0$, we know $\alpha \geq 0$. We can now rewrite the lower bound as follows:

$$\frac{m + (t-m)2^{-\ell}}{1 + t2^{-\ell-h}} \geq \frac{\alpha t + (t - \alpha t)2^{-\ell}}{2t2^{-\ell-h}} = \frac{\alpha + (1-\alpha)2^{-\ell}}{2^{-\ell-h+1}}$$

$$= \alpha 2^{\ell+h-1} + (1-\alpha)2^{h-1} = 2^{h-1}(\alpha 2^{\ell} + 1 - \alpha).$$

Since the online algorithm must create a specialization after seeing its first input, $\ell \geq 1$. Thus we have a lower bound of 2^{h-1}.

Combining both cases, the ratio is at least $\min(m/2, 2^h/2)$. By lemma 5.3, $m \geq \lfloor \log(n/h) \rfloor$. □

Theorem 5.5 (2.2) *Any online algorithm for the online monotone linearly-ordered specialization problem has competitive ratio at least* $(\log n)/4$.

Proof: Since the competitive ratio must be at least 1, clearly the statement is true for $n \leq 2^4$. Lemmas 5.2 and 5.4 taken together imply that the competitive ratio of any online algorithm must be at least $\min(\log\lfloor (n/h) \rfloor, 2^h)/2$. Assuming $n > 2^4$, choose h so that $2^h \leq \log n < 2^{h+1}$. This implies that $2^h > (\log n)/2$, and that $h = \lfloor \log\log n \rfloor$. Our competitive ratio is therefore at least

$$\min(\log \left\lfloor \frac{n}{\lfloor \log\log n \rfloor} \right\rfloor, \frac{\log n}{2})/2.$$

Using the fact that $n > 2^4$, we know

$$\log \left\lfloor \frac{n}{\lfloor \log\log n \rfloor} \right\rfloor \geq \log \left\lfloor \frac{n}{\log\log n} \right\rfloor \geq \log(\frac{n}{\log\log n} - 1) > \log(\frac{n}{2\log\log n})$$
$$= \log n - 1 - \log\log\log n > (\log n)/2.$$

Thus the competitive ratio is at least $(\log n)/4$. □

6 Conclusion

In conclusion, we present an online algorithm that decides between many different methods for processing input, where some inputs may be more specialized than others. Our algorithm is $O(\log n)$-competitive, and we also provide an $\Omega(\log n)$ lower bound on the competitive ratio for any online algorithm. We believe that our algorithm's design provides intuition for constructing online algorithms for a variety of practical problems, including the problem of dynamic compilation.

There are many ways to improve our current model. The most obvious one is to eliminate the restriction on the partial ordering. Since $\log n$ is a rather poor bound for use in practice, it may be better to change the model to a statistical one in order to derive better performance bounds that may be more practical. These new models could include information on the frequency of occurrence or probability of occurrence of various input types. The worst-case scenarios represented by the adversary may not actually happen in practice.

Another future direction to explore for this problem is to consider the model where methods can expire and must be created again to be used. This can model the problems where only a limited number of methods can be active at any one time (due to limited resources), or where the machinery used to process the inputs wears out over time. In the dynamic compilation application, having a limited number of methods active corresponds to limiting the amount of memory available for specialized versions of code. Exploring these alternative models would result in a better understanding of tradeoffs inherent in this problem, and could lead to better design of algorithms for practical specialization problems.

Acknowledgements

Thanks to some anonymous reviewers for proof reading, catching errors, and helpful suggestions on improving the structure and readability of this paper. Likewise, thanks to Richard Ladner, Anna Karlin, and Craig Chambers for their work on previous versions of this paper.

References

A. Aggarwal, B. Alpern, A. K. Chandra, and M. Snir. A model for hierarchical memory. In *ACM Symposium on Theory of Computing*, pages 305–314, May 1987.

B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. *Information and Computation*, 185(1):1–40, Aug. 2003.

Y. Azar, Y. Bartal, E. Feuerstein, A. Fiat, S. Leonardi, and A. Rosen. On capital investment. *Algorithmica*, 25(1):22–36, 1999.

A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

B. Grant, M. Mock, M. Philipose, C. Chambers, and S. J. Eggers. The benefits and costs of DyC's run-time optimizations. *ACM Transactions on Programming Languages and Systems*, 22(5):932–972, Sept. 2000.

S. Irani, G. Singh, S. Shukla, and R. Gupta. An overview of competitive and adversarial approaches to designing dynamic power management strategies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(12):1349–61, Dec. 2005.

A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.