Storage Management for Evolving Databases

J. Kleinberg* R. Motwani† P. Raghavan‡ S. Venkatasubramanian§

Abstract

The problem of maintaining data that arrives continuously over time is increasingly prevalent in databases and digital libraries. Building on a model for sliding-window indices *developed in [24], we devise efficient algorithms for some of the central problems that arise. We also show connections between the problems in this model and some fundamental problems in optimization and graph theory.*

1 Introduction

Large volumes of data arriving continuously over time add new complexity to problems of storage management in databases and digital libraries. Recent work in these communities has resulted in a framework for studying these issues (see e.g. [24] and the references therein). Among other results, we show here that one of the central problems within this model has a non-trivial polynomial-time solution, and we establish connections between some of these problems and a novel generalization of graph coloring that appears to be of interest in its own right.

We begin by highlighting three principal issues that motivate the framework.

Storage management costs: With the increasing volume of information that is stored in and retrieved from networks of computers, storage management is a growing concern. Indeed, it has been estimated that over the lifetime of storage hardware, the administration cost is about 100 times the hardware cost (disk drives are down to about 10 cents a megabyte, but administered storage costs \$3 to \$6 per megabyte per year, commercially). The trend is to automate and simplify (human-intensive) administrative functions through the use of sophisticated algorithms.

Continuously arriving data: There are many settings in which large databases must be maintained subject to the frequent insertion of large quantities of new information: these include data warehousing [26, 28]; web search engines and indices of Netnews articles (e.g. [14, 18, 15]); and automated fraud detection mechanisms such as SCAM (Stanford Copy Analysis Mechanism) [22, 23].

Sliding-window indices: Minimizing the storage required is a major objective in such settings; this is due both to storage costs and the requirements of fast query performance. One paradigm that has emerged is the use of *sliding-window indices*: for some parameter w, the index only maintains the data from the last w days. This limits the storage needed; and in many of the examples above, it can be justified on the grounds that most queries are to the most recent data. In this way, a sliding-window index can be used as a fast cache of the most recent data; users interested in older data must tolerate a slower search process.

1.1 The problem statement

In a *sliding-window index* [24], we have k *compartments* $P_1, \ldots, P_k$, and a *window size* w. Time progresses continuously, and a sequence of *items* $X_1, X_2, \ldots$ arrive at arbitrary times; unless otherwise specified, each item has an arbitrary *size.* Each item must be placed in a compartment as it arrives and reside there for at least w units of time (hereafter referred to as "days"). An item is said to be *live* during the time period extending from its arrival for exactly w days; thereafter, it is said to be *expired.*

Reclaiming the space left by expired items turns out to be a formidable problem. Query access to the items in the above scenarios is provided by an enormous inverted index (comparable to the size of the archive itself), and the work required to update the index structure after the deletion of a small number of items often outweighs the savings obtained by reclaiming the freed storage space. Thus for many of the applications of sliding-window indices (including those exemplified

*Department of Computer Science, Cornell University, Ithaca NY 14853. **Email:** kleinber@cs.cornell.edu. This work was done while on leave at IBM Almaden Research Center, San Jose, CA 95120.

†Department of Computer Science, Stanford University, Stanford, CA 94305. **Email:** rajeev@cs.stanford.edu. Partially supported by an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership Award, an ARO MURI Grant DAAH04-96-1-0007, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

‡IBM Almaden Research Center, San Jose, CA 95120. **Email:** pragh@almaden.ibm.com.

§Department of Computer Science, Stanford University, Stanford, CA 94305. **Email:** suresh@cs.stanford.edu.

by the TPC-D benchmark from the Transaction Processing Council [26]), Shivakumar and Garcia-Molina report that the following approach achieves the best performance [24]:

> *Wait and Throw Away (WATA)*: Only delete items from a given compartment when *all* the items in that compartment have expired.

The use of this indexing scheme amortizes the cost of reclaiming space by allowing expired items to be cleared out only in increments of whole compartments. However the latter constraint, that an index must be preserved so long as one of items in its compartment is active, is the main source of difficulty in implementing this scheme.

We study the quality of an *assignment*, which is a function ϕ from items to compartments. At any given time, a compartment P_i is *active* if it contains at least one live item; its *load* $L(P_i)$ is defined to be the total size of all items (live or expired) that were assigned to it since it was last inactive. This captures the notion that the load of P_i is the *sum* of the accumulated live and expired items that have built up in it since it was last "flushed out." A compartment gets *flushed out* the moment it is inactive.

There are two natural objective functions within this framework.

• **Total Load.** The *weight* at time t of an assignment ϕ, denoted $\mathcal{W}_t(\phi)$, is the sum of the loads of all active compartments at that time. The weight of ϕ, denoted $\mathcal{W}(\phi)$, is its maximum weight over all time. We seek an assignment of minimum weight; this captures the notion that we want to minimize the space taken up by expired items that are still in the system. Note that total load is a better measure of system performance than total "expired load," although minimizing either is equivalent from the point of view of optimal solutions.

Of course, in a simple model of items that vanish as soon as they expire, the total weight of an assignment at any time would be just the sum of the sizes of all live items, and hence the total load problem would be trivial. In our model, the total weight is the *sum* of the sizes of the live items *and* of the expired items that have not yet been flushed out. Thus, minimizing total load becomes a non-trivial optimization problem, as it is in practice. In [24], Shivakumar and Garcia-Molina propose a simple "round-robin" heuristic for minimizing total load which is always within a factor of 2 of optimal.

• **Maximum Load.** Here, we wish to find an assignment that minimizes the *maximum load* of any compartment, over all time. This is motivated by the problem of balancing the indexing requirements evenly across all compartments, thereby improving query performance.

1.2 The input model

Finally, we discuss how best to model the way in which input is presented to an indexing algorithm in this setting. The standard on-line model is unrealistically pessimistic, in that it assumes the indexing algorithm is designed with no knowledge of the item sizes that will arrive. In reality, trace information generally allows one to design algorithms that exploit known cyclical (e.g., weekly) patterns in data arrival. Moreover, such algorithms are also evaluated and tuned on this type of trace information [24].

Thus, the most reasonable input model is the following *periodic model*: we take a finite sequence of item arrivals and repeat it indefinitely, shifted periodically by a fixed increment of time. That is, for a number T and $i = 1, 2, \ldots, m$, an item of size s_i arrives at times $t_i, t_i + T, t_i + 2T, t_i + 3T, \ldots$. We note that the pure on-line model also makes sense in some scenarios, such as the case in which one must operate with no *a priori* profile information about the underlying arrivals; thus we study on-line algorithms for the problem as well.

1.3 Summary of results

Our first main result is a polynomial-time algorithm for the problem of minimizing total load. This improves on the 2-approximation heuristic proposed in [24]. We do this by first developing a polynomial-time algorithm for the *finite off-line model* (i.e. a finite set of items is presented to the algorithm), and then extending the techniques to apply to the periodic model as well.

In the pure on-line model, we show that no algorithm can provide an optimal solution for total load. We propose a simple on-line algorithm, BALANCED-PARTITION, that achieves an approximation ratio of $k/(k-1)$.

The problem of minimizing the *maximum load* is NP-complete, even with item sizes that are polynomially bounded integers, since it contains the bin-packing problem as a special case. We show how an adaptation of our techniques for the total load problem provides a 2-approximation algorithm for maximum load.

Coping with expiration raises some new combinatorial issues; in particular, for the maximum load problem, it is not obvious how to handle even the case of *unit-size* items. (For example, no on-line algorithm can produce the optimal load.) It turns out that this case leads to a novel generalization of graph coloring, as follows. We can represent the lifetime of each item as a (length w) interval on the line, and thereby obtain a unit-interval graph G. Now, observe that a maximum load of c can be maintained if and only if there is an assignment of intervals to compartments in such a way that that the intervals assigned to any one compartment consist of a number of connected components, each of size at most c. This has the following fundamental graph-theoretic interpretation, which we call the *fragmented coloring problem*:

Definition 1.1 *A* (k, c)-fragmented coloring *of a graph* G *is a partition of its vertex set into* k *sets* $V_1, \ldots, V_k$*, such that for* $i = 1, \ldots, k$*,* $G[V_i]$ *has no connected component of size greater than* c*.*

We will sometimes say that G is (k, c)-*fragmentable*. Clearly, a $(k, 1)$-fragmented coloring is just a k-coloring. Minimizing the maximum load with unit-size items is just the fragmented coloring problem for *unit-interval graphs*. In Section 3, we show how to decide the (k, c)-fragmentability of a unit interval graph in polynomial time; this gives us an optimal algorithm for the maximum load with unit-size items.

We have not been able to find any previous reference to the fragmented coloring problem in the literature (survey books such as [19] do not contain it). In its statement, it is similar to the *defective coloring problem* studied recently by Cowen, Goddard, and Jesurum [12]. In particular, a $(k, 1)$-defective coloring in the sense of [12] is the same as a $(k, 2)$-fragmented coloring in the present context; but for larger values of c the two notions become very different. The fragmented coloring problem appears to be remarkably intricate; in Section 4 we indicate some basic results that we can prove about it, and some interesting open questions.

2 Total load

We begin by presenting a strongly polynomial-time algorithm for the *finite off-line model*, in which items $\mathcal{X} = \{X_1, \ldots, X_n\}$ arrive at times $\tau_1, \ldots, \tau_n$. The techniques developed here will be crucial for the polynomial-time algorithm in the periodic model.

We assume throughout that there is no i such that $\tau_{i+1} - \tau_i \geq w$; if there is, we simply decompose the initial problem into independent sub-problems at all such gaps.

An assignment ϕ induces a canonical decomposition of the sequence $\mathcal{X}$ into *blocks* $U_1, \ldots, U_m$, such that each U_i is a maximal interval over which the same compartment is receiving items. We use $\phi(U_i)$ to denote the compartment receiving items in block U_i. A *compression* in ϕ is a pair of indices (t, t') such that:

- $t < t' - 1$,
- the same compartment P_i was assigned X_t and $X_{t'}$, and
- P_i was active but assigned no items in the interval of indices $\{t'' : t < t'' < t'\}$.

Observe that if (t, t') is a compression, then t must come at the end of a block and t' must come at the start of a block. Essentially, a compression is an interval of time in which the assignment "returns" to a compartment still containing live items.

The following lemma is a crucial ingredient in the development of the algorithm.

Lemma 2.1 *There is an optimal assignment that contains no compression.*

Proof. For an assignment ϕ that contains a compression, we define $c(\phi)$ to be the compression (t, t') which is maximal in lexicographic order. Suppose that the lemma does not hold; then every optimal assignment contains a compression. Let ϕ^* denote an optimal assignment for which $c(\phi^*) = (t, t')$ is minimal in lexicographic order. Let $U_1, \ldots, U_m$ denote the decomposition of $\mathcal{X}$ induced by ϕ^*. Suppose that U_i has X_t as its last element, and U_j has $X_{t'}$ as its first, where $i < j - 1$. Let $P^* = \phi^*(U_i) = \phi^*(U_j)$.

Let $\psi : \{P_1, \ldots, P_k\} \to \{P_1, \ldots, P_k\}$ denote the following function:

$$\psi(P) = \begin{cases} \phi^*(U_k) & \text{if } i \leq k < j \text{ and } \phi^*(U_{k+1}) = P \\ P & \text{otherwise} \end{cases}$$

Note that by the definition of $c(\cdot)$, the compartments $\phi^*(U_{i+1}), \ldots, \phi^*(U_{j-1})$ are all distinct and different from P^*; from this it follows that ψ is well-defined and a bijection.

We construct a new assignment ϕ' as follows:

- ϕ' agrees with ϕ^* up to the end of interval U_i,
- ϕ' assigns the items in U_{i+1} to P^*,
- for $k = i+2, i+3, \ldots$, ϕ' assigns the items in U_k to the compartment $\psi(\phi^*(U_k))$.

Thus, the block decomposition of $\mathcal{X}$ with respect to ϕ' consists of blocks $U'_1, \ldots, U'_{m-1}$, where $U'_k = U_k$ for $k < i$, $U'_i = (U_i \cup U_{i+1})$, and $U'_k = U_{k+1}$ for $k > i$. Let $\sigma(U_k)$ denote the block in $\{U'_1, \ldots, U'_{m-1}\}$ containing U_k. The following two facts will imply the result.

Fact A. *The compression* $c(\phi')$ *is lexicographically less than* $c(\phi^*)$*.*

For suppose that ϕ' has a compression (t_0, t_1), where $t_0 \geq t$. Then t_0 is at the end of a block U'_k, $k \geq i$, and t_1 is at the start of a block U'_ℓ. Now, if

$k > i$ then since $\phi'(U'_k) = \psi(\phi^*(U_{k+1}))$, $\phi'(U_\ell) = \psi(\phi^*(U_{\ell+1}))$, and ψ is a bijection, it follows that (t_0, t_1) is a compression in ϕ^* as well; this contradicts the assumption that $c(\phi^*) = (t, t')$. If, on the other hand, $k = i$, then $\phi'(U'_k) = \phi'(U'_\ell) = P^*$. But then $\phi^*(U_{i+1}) = \phi^*(U_{\ell+1}) = P_{i+1}$, and again (t_0, t_1) would be a compression in ϕ^*.

Fact B. *For all τ, $\mathcal{W}_\tau(\phi') \leq \mathcal{W}_\tau(\phi^*)$.*

To prove this, it is enough to show that if the items in U_k are erased from their compartment by time τ, then the items in $\sigma(U_k)$ are erased from their compartment by time τ. For $k < i$, this follows from the fact that in the subsequence $U_1 \cup \cdots \cup U_j$, the intervals between consecutive assignments to a compartment by ϕ' are supersets of the intervals between consecutive assignments by ϕ^*. For $k \geq i$, the items in $\sigma(U_k)$ are erased exactly w days after the end of $\sigma(U_k)$; otherwise we would have a compression in ϕ' that is reverse-lexicographically no less than (t, t'), in contradiction of Fact A.

By Fact B, ϕ' is an optimal assignment; but by Fact A, this contradicts the minimality of ϕ^*. ∎

We define a function $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ as follows: $f(i) = j$ if and only if X_j is the last item to arrive before the expiration of X_i. An *interval* is a set of contiguous indices; define the interval $I_j = \{j, j+1, \ldots, f(j-1)\}$.

Lemma 2.2 *If an assignment ϕ has no compression, then no k blocks have their start times in an interval of the form I_j.*

Proof. Let ϕ be an assignment, and suppose that k blocks have their start times in the interval I_j. The items $j-1, \ldots, f(j-1)$ are all active at the same time, and belong to $k+1$ different blocks; thus, some two of these blocks must induce a compression. ∎

Lemma 2.3 *Let ϕ be an assignment with no compression, and let $t_1, \ldots, t_m$ be the start indices of each block. Let $t_{m+1} = n + 1$. Then,*

$$\mathcal{W}(\phi) = \max_{1 \leq i \leq m} \sum_{k=t_i}^{f(t_{i+1}-1)} s_k.$$

Proof. The set of items $X_{t_i}, \ldots, X_{f(t_{i+1}-1)}$ simultaneously resides in active compartments; thus $\mathcal{W}(\phi)$ is at least as large as the right-hand side. To show the other inequality, note that if $k \neq f(t_i - 1)$ for some i, then $\mathcal{W}_{\tau_{k+1}}(\phi) \geq \mathcal{W}_{\tau_k}(\phi)$. ∎

We say that G is a *path graph* if G has vertex set $\{v_1, \ldots, v_n\}$, and for each $i \in \{1, \ldots, n-1\}$ there exists a $j > i$ such that v_i has out-edges to the nodes $v_{i+1}, \ldots, v_j$. We say that a path P in G is *spanning* if it contains v_1 and v_n. We say that an *interval* I in G is a set of nodes of G whose indices are contiguous; we say that an *interval system* in G is a set $\mathcal{I}$ of intervals in G. A path P in G is $(a, \mathcal{I})$*-sparse* if no interval in $\mathcal{I}$ contains a vertices of P.

Now, for a parameter N, we construct a path graph G_N with vertex set $\{v_1, \ldots, v_{n+1}\}$, such that there is a directed edge from v_i to v_j if $i < j$ and

$$\sum_{k=i}^{f(j-1)} s_k \leq N. \tag{1}$$

Observe that all paths in G_N have monotonically increasing indices. We use $\mathcal{I}$ to denote the interval system in G_N consisting of the intervals $I_1, \ldots, I_n$ defined above.

Lemma 2.4 *There is an assignment of weight at most N iff there is a $(k, \mathcal{I})$-sparse path in G_N.*

Proof. Let ϕ be a compression-free assignment with $\mathcal{W}(\phi) \leq N$, and let $t_1, \ldots, t_m$ be the start indices of each block. By Lemma 2.2, the path path $P = \{v_{t_1}, \ldots, v_{t_m}, v_{n+1}\}$ is $(k, \mathcal{I})$-sparse. Moreover, for each i, the set of items $X_{t_i}, \ldots, X_{f(t_{i+1}-1)}$ simultaneously resides in active compartments; hence $\sum_{k=t_i}^{f(t_{i+1}-1)} s_k \leq N$ and so P is a path in G_N.

Conversely, we can construct an assignment ϕ from any $(k, \mathcal{I})$-sparse spanning path P in G_N by using the indices in P to begin the blocks. If we assign the compartments in round-robin order, the resulting assignment will be compression-free, and hence its weight will be at most N by Lemma 2.3. ∎

In order to find an optimal assignment, we study the problem of finding sparse paths.

Lemma 2.5 *Let G be a path graph, a a natural number, and $\mathcal{I}$ an interval system in G. In polynomial time we can decide whether G contains an $(a, \mathcal{I})$-sparse spanning path.*

Proof. First, given the interval system $\mathcal{I}$, let us define a new interval system $\mathcal{I}'$ such that G has an $(a, \mathcal{I})$-sparse spanning path if and only if it has an $(a, \mathcal{I}')$-sparse spanning path. For each $v_i \in G$, let v_j be the the maximal right endpoint of any interval in $\mathcal{I}$

containing v_i. We define $I_{v_i} = \{v_i, v_{i+1}, \ldots, v_j\}$, and set $\mathcal{I}' = \{I_{v_1}, \ldots, I_{v_n}\}$. Note that no two intervals in $\mathcal{I}'$ are nested. It is easy to verify that there is an $(a, \mathcal{I})$-sparse path if and only if there is an $(a, \mathcal{I}')$-sparse one.

For $v \in G$, let $\pi(v)$ denote the path obtained by starting at v and iteratively following longest out-edges. If $Z \subset V(G)$, let $\pi_Z(v)$ denote the path obtained by starting at v and iteratively following longest out-edges, subject to the condition that no vertex in Z is visited. (If no out-edges can be followed, then the current vertex is simply repeated.) Finally, let $q_Z(v,k)$ denote the vertex visited on the k^{th} step of $\pi_Z(v)$.

Our algorithm is as follows:

(1) Initialize $Z = \phi$.

(2) Until Z does not change:

 (a) For $u = v_n, v_{n-1}, \ldots, v_1$: if $q_Z(u, a-1) \in I_u$, then put u in Z.

(3) If $v_1 \in Z$ then no $(a, \mathcal{I}')$-sparse spanning path exists. Otherwise, $\pi_Z(v_1)$ is such a path.

It is clear that the algorithm terminates. Let Z^* denote the value of Z at termination. By induction on the order in which vertices are placed in Z^*, it is easy to show that if $v_i \in Z^*$, then there is no $(a, \mathcal{I}')$-sparse path from v_i to v_{n+1}.

Finally, we claim that $\pi_{Z^*}(v_i)$ is an $(a, \mathcal{I}')$-sparse spanning path. For if not, then there is a vertex u on this path for which $q_{Z^*}(u, a-1) \in I_u$. But in this case, u would have been added to Z^* in the final iteration (when Z^* supposedly did not change), a contradiction. ■

Using binary search (over $O(n^2)$ possible values) to find the minimum N for which G_N has a sparse spanning path, we obtain the following result.

Theorem 2.6 *It is possible to find an optimal assignment in strongly polynomial time.*

The Periodic Model. We now develop an algorithm for the periodic model, using many of the definitions and techniques from the finite case. In the periodic version of the problem, we are given a basic *period*, consisting of item sizes $\{s_1, \ldots, s_m\}$ and times $0 \le \tau_1 \le \cdots \le \tau_m < T$. These quantities generate an infinite sequence of arrivals as follows: for $j = 0, 1, 2, \ldots$, an item of size s_i arrives at time $jT + \tau_i$. We will assume here that T is a positive integer multiple of w. (Assuming that T and w are integers, this is essentially no loss of generality: in the event that T is not a multiple of w, we can repeat the basic period several times, and replace T by the least common multiple of T and w.)

An assignment is now a function from this infinite list of items to the k compartments. It is easily observed that there exists an assignment whose weight is finite, and hence it makes sense to speak of an *optimal* assignment.

Lemma 2.7 *There is an optimal assignment that contains no compression.*

Proof. Given that we have proven the analogous fact for the finite case, we can invoke a type of compactness argument. Suppose that the value of an optimal assignment is W^*. For $j = 1, 2, \ldots$, let ϕ_j denote an assignment for the first j items that has no compression and maintains a weight of at most W^*. Let S_1 denote the set of all these assignments. We now construct an assignment ϕ^* for the periodic problem via a countable sequence of stages.

> Stage i: Of the assignments in S_i, an infinite subset S_i' agree on a compartment P_{k_i} in which to place the i^{th} item. Define ϕ^* to place the i^{th} item in P_{k_i}, and set $S_{i+1} = S_i'$.

Since for any finite subset of the items, the assignment ϕ^* agrees with some assignment in S_1, it follows that ϕ^* has no compression and maintains a weight of at most W^*. ■

It will follow from the analysis to come that there is an optimal assignment that is periodic. Indeed, our algorithm will produce an optimal assignment of a particularly simple form; we will discuss this more below.

We define the function f as before, except that it now ranges over the non-negative integers. The analogues of Lemmas 2.2 and 2.3 hold as before. The main idea in developing an optimal algorithm for the periodic case is to establish that there is an optimal periodic assignment; and we do this by reducing the problem to the following variant of the sparse path problem.

We first define the notion of a *ρ-graph*. Let G be a graph with vertex set $\{v_1, \ldots, v_m\} \cup \{w_1, \ldots, w_m\}$. For $1 \le r \le m$, we define $v_{i+r} = w_r$. We say that a set I of nodes in G is an *interval* if one of the following holds:

(i) $I = \{v_i, v_{i+1}, \ldots, v_{i+r}\}$,

(ii) $I = \{w_i, w_{i+1}, \ldots, w_{i+r}\}$, with indices in this case interpreted modulo m.

In both cases, the node listed first in I is called the *start*, and the node listed last the *end*. We say

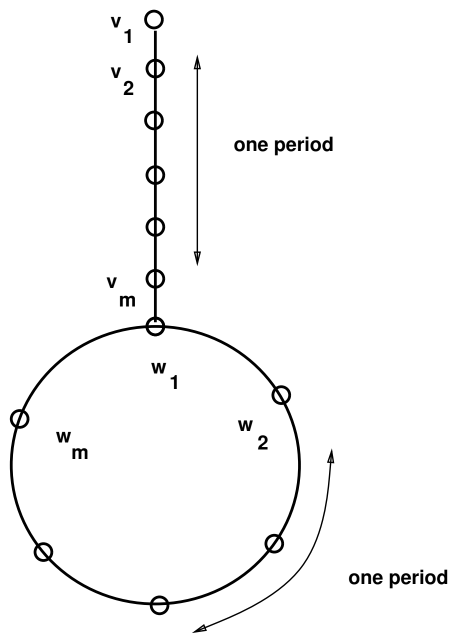

Figure 1: A periodic sparse path problem.

that G is a ρ-graph if each node v_i (resp. w_i) has directed edges precisely to nodes that constitute an interval whose start is v_{i+1} (resp. w_{i+1}). See Figure 1 for the basic shape of a ρ-graph. Finally, if $\mathcal{I}$ is a set of intervals, we say that a path P in G is $(a, \mathcal{I})$-*sparse* if no interval in $\mathcal{I}$ contains a vertices of P.

Since we are assuming that T is a multiple of w, it follows that if $j > i \geq 0$ and $1 \leq r \leq m$, then $f(im+r)$ is congruent to $f(jm+r)$ modulo m. Moreover, $f(im+r) \leq (i+1)m + r$. We therefore define $g : \{1, \ldots, m\} \rightarrow \{1, \ldots, m\}$ as follows: $g(r) = r'$ if $f(im+r)$ is congruent to r' modulo m, for all i.

For a parameter N, we construct a ρ-graph G_N with vertex set $\{v_1, \ldots, v_m\} \cup \{w_1, \ldots, w_m\}$ and edges as follows. For each node v_i (resp. w_i) in G_N, set $s(v_i)$ (resp. $s(w_i)$) equal to s_i. Now, each node v_i has a directed edge to v_j if $\sum_{k=i}^{g(j-1)} s(v_k) \leq N$. The edges from w_i are defined analogously. Also, we define $I_j = \{v_j, v_{j+1}, \ldots, v_{g(j-1)}\}$, $I'_j = \{w_j, w_{j+1}, \ldots, w_{g(j-1)}\}$, and the set $\mathcal{I} = \{I_1, \ldots, I_m, I'_1, \ldots, I'_m\}$.

The following can now be shown by analogy with the proof of Lemma 2.4. Note that although G_N is a finite graph, it still makes sense to speak of infinite (non-simple) paths in G_N.

Lemma 2.8 *There is an assignment of weight at most N iff there is an infinite $(k, \mathcal{I})$-sparse path in G_N containing v_1.*

By an algorithm very similar to that in the proof of Lemma 2.5, we show

Lemma 2.9 *Let G be a ρ-graph, a a natural number, and $\mathcal{I}$ a set of intervals in G. In polynomial time we can decide whether G contains an infinite $(a, \mathcal{I})$-sparse spanning path containing v_1.*

Now, if we set $N = \sum_{i=1}^{m} s_i$, then there is clearly an assignment that maintains a total size of at most N (using two compartments). And as N ranges from 0 to $\sum_{i=1}^{m} s_i$, we obtain only $O(m^2)$ different ρ-graphs G_N. Using binary search to find the minimum N for which one of these G_N has an infinite sparse path from v_1, we obtain

Theorem 2.10 *It is possible to find an optimal assignment in strongly polynomial time.*

It is also worth noting that the optimal assignment produced by the algorithm in Lemma 2.9 has a very simple form. Let us say that an infinite path P in the ρ-graph G_N is *homogeneous* if there is a function $h : V(G_N) \rightarrow V(G_N)$ such that if u is the i^{th} node of P, then $h(u)$ is the $(i+1)^{\text{st}}$ node of P. We have

Theorem 2.11 *There is an optimal assignment, with weight N, which corresponds to a homogeneous path in the graph G_N.*

Such an assignment has the nice property that it can represented very compactly: for each node of G_N, one simply has to record its "successor" in the block decomposition of the assignment.

3 Maximum load

We now turn to the problem of minimizing the maximum load in any compartment, over all time. We will focus here on the *finite* version of the problem; we must find an assignment for a finite set of items $\{X_1, \ldots, X_n\}$, where X_i arrives at time τ_i and has size s_i. We noted earlier that this problem is NP-complete, even when all weights are polynomially bounded integers, since it contains the bin-packing problem.

A major difficulty here, compared to the total load metric, is that the optimum assignment need not be compression-free. However, compression-free assignments have a number of practical advantages — particularly, temporal locality — and thus remain an interesting case to focus on. We show that the sparse path algorithm of the preceding section can be used to find an optimal compression-free assignment in polynomial time.

Theorem 3.1 *There exists a polynomial-time algorithm to find the compression-free assignment with minimum maximum load.*

Proof. We build a path graph H_N on nodes $w_1, \ldots, w_{n+1}$: (w_i, w_j) is a directed edge if and only if $\sum_{t=i}^{j-1} s_t \leq N$. We define the following interval system $\mathcal{I}$ on H: for $i < j$, we include an interval $\{w_i, \ldots, w_j\}$ if and only if the lifetimes of X_{i-1} and X_j overlap. One can now show that there is a compression-free assignment ϕ with maximum load at most N if and only if H_N has a $(k, \mathcal{I})$-sparse spanning path. The result follow from this fact, combined with Lemma 2.5. ■

We use Theorem 3.1 to obtain two main consequences. First, it implies that the case of unit-size items can be solved optimally in polynomial time, in view of the following lemma. The proof is similar to that of Lemma 2.1.

Lemma 3.2 *Let all items have unit size. If a maximum load c can be maintained, then there is a compression-free assignment with maximum load c.*

Proof. As in the proof of Lemma 2.1, if the lemma does not hold then every optimal assignment has a compression. Choose the assignment ϕ^* for which $c(\phi^*)$ is minimal in lexicographic order. Let $U_1, \ldots, U_m$ denote the decomposition of $\mathcal{X}$ induced by ϕ^*. Suppose that U_i has X_t as its last element, and U_j has $X_{t'}$ as its first, where $i < j - 1$. Let $P^* = \phi^*(U_i) = \phi^*(U_j)$. Define the function

$$\psi(P) = \begin{cases} \phi^*(U_k) & \text{if } i \leq k < j \text{ and } \phi^*(U_{k+1}) = P \\ P & \text{otherwise} \end{cases}$$

Let $X_r, X_{r+1}, \ldots, X_s$ denote the items in the set $U_{i+1} \cup \cdots \cup U_j$. We construct a new assignment ϕ':

- Assignment ϕ' agrees with ϕ^* up to the end of interval U_i.
- If $|U_j| \geq |U_{i+1}|$, then we simply assign the items in U_{i+1} to P^* and the items in U_k $(k > i+1)$ to $\psi(\phi^*(U_k))$. Otherwise, let q be the minimal index among $i+2, \ldots, j$ for which $|U_j| \geq |U_q|$, and let X_ℓ be the last item in U_q. Among the items in $\{X_r, \ldots X_\ell\}$, we assign the first $|U_j|$ to P^*, the next $|U_{i+1}|$ to P_{i+1}, the next $|U_{i+2}|$ to P_{i+2}, and so on, assigning the final $|U_{q-1}| - |U_q|$ items to P_{q-1}. For $k > q$, we assign the items in U_k to $\psi(\phi^*(U_k))$.

As in the proof of Lemma 2.1, we have the following two facts, which contradict the minimality of ϕ^*.

Fact A′. *$c(\phi')$ is lexicographically less than $c(\phi^*)$.*

Fact B′. *The maximum load of ϕ' is at most that of ϕ.*

■

Second, Theorem 3.1 provides a 2-approximation for the optimum load of an arbitrary assignment. This is a consequence of the following. (We assume below that item sizes are integers.)

Theorem 3.3 *The maximum load of the optimal compression-free assignment is at most twice the maximum load of the optimal assignment.*

Proof. Given a problem with items $\mathcal{X} = \{X_1, \ldots, X_n\}$, each of whose sizes is an integer, we create a new problem as follows. We represent item X_i, whose size is s_i, as s_i items $Y_{i1}, \ldots, Y_{is_i}$ of unit size, each of which arrives at time τ_i. Let $\mathcal{Y} = \{Y_{ij}\}$. Note that the optimal maximum load for the $\mathcal{Y}$ problem is at most that for the $\mathcal{X}$ problem; and by Lemma 3.2, there is a compression-free assignment ϕ^* that optimizes the maximum load for the $\mathcal{Y}$ problem. We use ϕ^* to guide the construction of a compression-free assignment ϕ for the $\mathcal{X}$ problem.

Consider the set $\mathcal{Y}_i$ of items in $\mathcal{Y}$ corresponding to item X_i. If all these items are assigned to a single compartment P, then ϕ assigns X_i to P. If the items in $\mathcal{Y}_i$ are assigned to precisely two compartments P and P', then ϕ assigns X_i to the one which receives more than half of $\mathcal{Y}_i$ under ϕ^*. Finally, suppose that ϕ^* assigns the items in $\mathcal{Y}_i$ to more than two compartments. Then in the block decomposition of ϕ^*, there is a block consisting exclusively of assignments of items in $\mathcal{Y}_i$ to a single compartment P. ϕ assigns the item X_i to such a compartment P.

We now observe two facts. First, ϕ is compression-free. Second, the maximum load of ϕ exceeds the maximum load of ϕ^* by at most the maximum item size; since the maximum load of ϕ^* and the maximum item size are both lower bounds on the optimum for the $\mathcal{X}$ problem, the load of ϕ is at most twice this optimum. ■

We note that the ratio of 2 in the above theorem is asymptotically tight.

4 Fragmented coloring

Recall the definition of the *fragmented coloring problem* from the introduction. As noted there, deciding whether a maximum load of c can be maintained for a set of unit-size items is equivalent to the (k, c)-fragmented coloring problem for a unit-interval graph: one represents the lifetime of each item as a unit interval on a line, views the assignment of items to compartments as a k-coloring, and requires that there be no monochromatic connected component of size greater than c.

We feel that (k, c)-fragmented coloring is an interesting combinatorial notion in its own right, and we discuss some basic results and questions here. First of all, let us compare this notion with that of *defective coloring*, as studied in Cowen, Goddard, and Jesurum [12]. There, a graph is said to have a (k, d)-defective coloring if it can be k-colored so that each vertex is adjacent to at most d other vertices of the same color. Thus, a graph has a $(k, 2)$-fragmented coloring if and only if it has a $(k, 1)$-defective coloring. Hence, by a result of [12], it is NP-hard to decide whether an arbitrary graph has a $(k, 2)$-fragmented coloring. However, for larger values of c and d, the notions of fragmented coloring and defective coloring are very different. (In particular, once $d > 1$, there is no way to bound the size of the monochromatic components in a (k, d)-defective coloring.)

It is clear that for each k and c, there exists a graph that has no (k, c)-fragmented coloring; take for example the complete graph on $kc + 1$ vertices. It is much harder to find counterexamples of bounded-degree; for example, a natural question is whether there exists an absolute constant k_0 and a function $f(\cdot)$ so that every graph of maximum degree Δ has a $(k_0, f(\Delta))$-fragmented coloring. Using properties of strong expander graphs — in particular, a construction due to Alon and Chung [1] — we can show that the answer to this question is negative.

Theorem 4.1 *There is an absolute constant ε such that the following holds. For every Δ, there is an infinite family $\{G_n\}$ of graphs of maximum degree Δ, none of which has an $(\varepsilon\sqrt{\Delta}, \varepsilon|G_n|)$-fragmented coloring.*

It would be interesting to see whether one obtains qualitatively different behavior when k is asymptotically much smaller than $\sqrt{\Delta}$.

Another set of questions concerns planar graphs. The four-color theorem is precisely the statement that planar graphs have $(4, 1)$-fragmented colorings. We can show

Theorem 4.2 *For each c, there exists a planar graph that has no $(3, c)$-fragmented coloring.*

Proof. We construct a planar graph G_c as follows. G_c has vertex set

$$\begin{aligned} V &= \{v_{i,j} : 1 \le i \le c^2 + c, 1 \le j \le c\} \\ &\quad \cup \{u_i : 1 \le i \le c\} \cup \{w\}. \end{aligned}$$

$v_{i,j}$ is joined to $v_{i,j-1}$ and $v_{i,j+1}$ (provided these nodes are defined); u_i is joined to $v_{i,j}$ for all j, and w is joined to all other nodes. Let $S_i \subset V$ denote the set consisting of u_i and all its neighbors.

Consider any 3-coloring of G_c, and suppose it has no monochromatic component of size greater than c. We first note that there is some set S_i that receives only two colors; for otherwise, the monochromatic component containing w would have more than c vertices. Now, in this set S_i, at most $c - 1$ of the $v_{i,j}$ can have the same color as u_i. But this means that in S_i, there is a monochromatic path of the form $v_{i,j}, v_{i,j+1}, \ldots, v_{i,j+c}$, for some index j; this contradicts our assumption that G_c has a $(3, c)$-coloring. ∎

Note that the planar graphs constructed in this theorem have unbounded degree. In view of this, and the fact that we only know how to prove Theorem 4.1 via the use of strong expander graphs, an interesting open question is the following:

Question 4.3 *Is there a function $f(\cdot)$ such that all planar graphs of maximum degree Δ have a $(3, f(\Delta))$-fragmented coloring?*

5 On-Line algorithms

On-line Minimization of Total Load. Consider now the on-line version of the total load problem, in which items arrive over time and must be assigned to compartments immediately.

Theorem 5.1 *Any deterministic on-line algorithm has competitive ratio at least $1 + 1/k$.*

If we assume that at most one item arrives per day, then a simple on-line algorithm is provided by the EQUIPARTITION heuristic of Shivakumar and Garcia-Molina [24] which places $(w-1)/(k-1)$ files in each compartment before moving on to the next one.

Theorem 5.2 *EQUIPARTITION is 2-competitive but not c-competitive for any $c < 2$.*

We now give an on-line algorithm that nearly matches the on-line lower bound contained in Theorem 5.1. We assume here that we are given T, the maximum total size of active items at any point in time. Define $L = T/(k-1)$. Our heuristic, BALANCED-PARTITION, works as follows: for each compartment, stack up items until the total size first exceeds L; then, move on to the next compartment. We establish the following properties of BALANCED-PARTITION.

Lemma 5.3 *The assignment that is produced by BALANCED-PARTITION has no compressions.*

Proof. We prove by induction on time that BALANCED-PARTITION maintains the following invariant on the compartments: if P_j is its current active compartment, then there is an index i so that the compartments $P_{j-1}, P_{j-2}, \ldots, P_i$ (indices modulo k) each contain weight at least L, and the only expired items in the system reside in P_i. This invariant will be maintained provided that when BALANCED-PARTITION moves to P_{j+1}, it is empty. But if this were not the case, then the $k-1$ compartments other than P_{j+1} would each contain a weight of at least L in live items, and P_{j+1} (since it is non-empty) would contain at least one live item. But then the set of live items would have total size more than $(k-1)L = T$, contradicting the definition of T. ∎

Lemma 5.4 *At any time, the total load of expired items that are still in the system is at most L.*

Proof. Since the assignment produced by BALANCED-PARTITION has no compression, there is at most one active compartment P^* at any time that contains expired items. We observe that by the definition of BALANCED-PARTITION, only the last item added to P^* can have size greater than L, and since P^* is active, this must be a live item. Hence, the total size of expired items is at most L. ∎

Theorem 5.5 *BALANCED-PARTITION maintains total load at most $\frac{kT}{k-1}$, and hence is $\frac{k}{k-1}$-competitive.*

Proof. We obtain that the total load on the system is the weight of the currently live items (at most T) and the currently expired items (at most L, by Lemma 5.4). This establishes the theorem. ∎

On-Line Minimization of Maximum Load. As before, it is reasonable to assume that we are given the value T of the maximum total size of live items at any point in time. Let $s_{\max}$ be the maximum size of a single item (we do not need to be told the value of $s_{\max}$). Define $L = T/(k-1)$.

The BALANCED-PARTITION algorithm for the total load problem, with parameter L, provides a good approximation in this setting as well. Recall that, by Lemma 5.3, it produces a compression-free schedule, and hence it maintains a maximum load of $L + s_{\max}$. If we let OPT denote the optimum maximum load obtainable, we have $s_{\max} \leq \text{OPT}$, and $L \leq k \times \text{OPT}/(k-1)$. We obtain the following result.

Theorem 5.6 *For the maximum load problem, BALANCED-PARTITION is $\frac{2k-1}{k-1}$-competitive*

For two bins, we can provide an improved algorithm in the following manner: Proceed as in BALANCED-PARTITION, but never exceed L. For this on-line algorithm, called AGGRESSIVE-BALANCED-PARTITION, we can prove the following:

Theorem 5.7 *AGGRESSIVE-BALANCED-PARTITION is 2-competitive for $k = 2$.*

One can show that for $k > 2$, AGGRESSIVE-BALANCED-PARTITION is not 2-competitive.

Acknowledgements

We are grateful to Hector Garcia-Molina and N. Shivakumar for introducing us to this problem and for their help with the model. We also thank Soumen Chakrabarti and Piotr Indyk for valuable discussions.

References

[1] N. Alon, F.R.K. Chung. Explicit construction of linear-size tolerant networks. *Discrete Math*, 72(1988).

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line machine scheduling with applications to load balancing and virtual circuit routing. In *Proceedings of 25th Annual ACM Symposium on Theory of Computing*, 1993, pp. 623–631.

[3] B. Awerbuch, Y. Azar, E.F. Grove, M-Y. Kao, P. Krishnan, and J.S. Vitter. Load Balancing in the L_p Norm. In *Proceedings of 36th Annual IEEE Symposium on Foundations of Computer Science*, 1995, pp. 383–391.

[4] Y. Azar, A.Z. Broder, and A.R. Karlin. On-line load balancing. *Theoretical Computer Science*, 130(1994):73–84.

[5] Y. Azar, B. Kalayanasundaram, S. Plotkin, K. R. Pruhs, and O. Waarts. On-line Load Balancing of Temporary Tasks. *Proceedings of the 1993 Workshop on Algorithms and Data Structures*, 1993, pp. 119–130.

[6] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New Algorithms for an Ancient Scheduling Problem. In *Proceedings of 24th Annual ACM Symposium on Theory of Computing*, 1992, pp. 51–58.

[7] Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50(1994):113–116.

[8] E.W. Brown, J.P. Callan, and W.B. Croft. Fast incremental indexing for full-text information retrieval. In *Proceedings of the 20th International Conference on Very Large Databases*, 1994, pp. 192–202.

[9] M. Charikar and P. Indyk. *Personal communication*, 1995.

[10] Bo Chen, A. van Vliet, and G.J. Woeginger. New lower and upper bounds for on-line scheduling. *Operations Research Letters*, 16(1994):221–230.

[11] Bo Chen, A. van Vliet, and G.J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Research Letters*, 51(1994):219–222.

[12] L. Cowen, W. Goddard, and E. Jesurum. Coloring with defect. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[13] D. Cutting and J. Pedersen. Optimizations for dynamic inverted index maintenance. In *Proceedings of the International Conference on Information Retrieval.* 1990, pp. 405–411.

[14] DEC Corporation. AltaVista search engine. `http://altavista.digital.com`.

[15] Deja News Research Service. Dejanews news research service. `http://www.dejanews.com`.

[16] U. Faigle, W. Kern, and G. Turán. On the Performance of On-Line Algorithms for Partition Problems. *Acta Cybernetica*, 9(1989):107–119.

[17] R.L. Graham. Bounds for Certain Multiprocessor Anomalies. *Bell System Technical Journal,* 45(1966):1563–1581.

[18] Infoseek Corporation. Infoseek search engine. `http://www.infoseek.com`.

[19] T. Jensen and B. Toft. *Graph Coloring Problems.* Wiley Interscience, 1995.

[20] D. Karger, S.J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[21] F.T. Leighton and E.J. Schwabe. Efficient algorithm for dynamic allocation of distributed memory. In *Proceedings of 32nd Annual IEEE Symposium on Foundations of Computer Science*, 1991, pp. 470–479.

[22] P.E. Ross. Cops versus robbers in cyberspace. *Forbes Magazine*, September 9, 1996, pp. 134–139.

[23] N. Shivakumar and H. Garcia-Molina. Building a scalable and accurate copy detection mechanism. In *Proceedings of the 1st ACM Conference on Digital Libraries*, 1996.

[24] N. Shivakumar and H. Garcia-Molina. Wave Indices: Indexing Evolving Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997.

[25] A. Tomasic, H. Garcia-Molina, and K. Shoens. Incremental updates of inverted lists for text document retrieval. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1994.

[26] TPC Committee. Transaction Processing Council. `http://www.tpc.org`.

[27] J. Westbrook. Load Balancing for Response Time. Manuscript, 1996.

[28] J. Widom. Research problems in data warehousing. In *Proceedings of the 4th Conference on Information and Knowledge Management*, November 1995.

[29] T. Yan and H. Garcia-Molina. Sift – a tool for wide-area information dissemination. In *Proceedings of USENIX*, 1995.