New Algorithms for the Disk Scheduling Problem

Matthew Andrews*
Laboratory for Computer Science
MIT
andrews@theory.lcs.mit.edu

Michael A. Bender†
Aiken Computation Laboratory
Harvard University
bender@das.harvard.edu

Lisa Zhang‡
Laboratory for Computer Science
MIT
ylz@theory.lcs.mit.edu

Abstract

Processor speed and memory capacity are increasing several times faster than disk speed. This disparity suggests that disk I/O performance will become an important bottleneck. Methods are needed for using disks more efficiently. Past analysis of disk scheduling algorithms has largely been experimental and little attempt has been made to develop algorithms with provable performance guarantees.

We consider the following disk scheduling problem. Given a set of requests on a computer disk and a convex reachability function which determines how fast the disk head travels between tracks, our goal is to schedule the disk head so that it services all the requests in the shortest time possible. We present a 3/2-approximation algorithm (with a constant additive term). For the special case in which the reachability function is linear we present an optimal polynomial-time solution.

*The disk scheduling problem is related to the special case of the asymmetric Traveling Salesman Problem with the triangle inequality (*ATSP-Δ*) in which all distances are either 0 or some constant α. We show how to find the optimal tour in polynomial time and describe how this gives another approximation algorithm for the disk scheduling problem. Finally we consider the on-line version of the problem in which uniformly-distributed requests arrive over time. We present an algorithm (related to the above* ATSP-Δ*) that appears to give higher throughput than previously existing head scheduling algorithms.*

*Supported by NSF contract 9302476-CCR, ARMY grant DAAH04-95-1-0607 and ARPA contract N00014-95-1-1246.

†Supported by NSF contract 9313775-CCR.

‡Supported by an NSF graduate fellowship, ARMY grant DAAH04-95-1-0607 and ARPA contract N00014-95-1-1246.

1. Introduction

Processor speed and disk and memory capacity are increasing by over 40% per year. In contrast, disk speed is increasing more gradually, growing by only 7% per year [19]. Since this rate is unlikely to change substantially in the near future, I/O performance may become the system bottleneck. However, despite the difficulty of improving mechanical components, we can still aim to *use* the disks more efficiently.

For example, disks generally operate at a small fraction of their maximum bandwidth. Researchers have demonstrated experimentally that sophisticated disk head scheduling algorithms can deliver higher throughput [20, 12, 23]. This past research has focused almost exclusively on two types of work loads: synthetic work loads, where disk requests are randomly and uniformly distributed across the disk, and more recently, traces, where the requests to an actual disk are recorded and used as a testing ground for algorithms. However, for these or for general work loads, researchers have made little attempt to develop algorithms with provable performance *guarantees*. In addition, no one has determined the computational complexity of the disk scheduling problem. There is a risk that synthetic work loads and traces from a few environments may not represent all possible situations.

In this paper we propose several disk-scheduling algorithms with performance guarantees and we state a hardness result. The research has provided additional payoffs. The first, of practical interest, is an algorithm that seems to perform better than previously-existing head scheduling algorithms when servicing random uniformly-distributed requests that arrive on-line. The second payoff is of theoretical interest: the disk problem suggests algorithms for a special case of the asymmetric traveling salesman problem with the triangle inequality (ATSP-Δ). Before defining our problem formally we describe the structure of a modern disk.

The Disk A computer disk is composed of several concentric, rapidly-rotating *platters*, where data may be written to both sides of each platter. Platters are logically divided into circular *tracks*. A *cylinder* is composed of all the circular tracks having the same radius. The smallest unit that can be written to disk is called a *sector*, which typically holds 512 bytes of data. Modern disks have approximately 2000 cylinders and 100 sectors per track. The data is transferred to and from the disk by a set of read/write heads (usually one per surface). The disk arm moves the heads in concert, so that all of the heads are contained in one cylinder. For this reason we can restrict our attention to one disk platter and one disk head.

When a head accesses a particular sector, it suffers two kinds of delays. The *seek time* is the time required to move the head to the correct track; the *rotational latency* is the time necessary, once the head is in the correct track, for the requested sector to pass underneath the head. Modern disks rotate at a speed of 3600-7200 rpm (implying that one rotation takes 8-16 msec). With today's technology, the time for a *track-to-track* seek (one track to a neighboring track) is typically 1 msec; the time for a *full-seek* (the innermost to the outermost track) is typically 20 msec. Small seeks are dominated by a constant start-up time, medium-length seeks by a period of acceleration and deceleration, and long seeks by a period of constant speed. In the appendix we give the specifications from [19] for the Hewlett-Packard 97560 disk.

The Problem In this paper we chiefly consider the off-line version of the disk scheduling problem. The input consists of a set of points on the disk (which we call *requests*) and a convex *reachability function* which determines how long it takes the disk head to move between tracks. Our goal is to schedule the disk head so that it services (i.e. visits) all of the requests in the shortest possible time. Note that if we consider the motion of the head relative to the disk then the problem becomes a special case of the Traveling Salesman Problem. We also consider an on-line version of the disk scheduling problem in which the requests arrive over time and are placed into a queue. The head is able to service any request that is currently in the queue. Our goal is to maximize the throughput.

Our Results The main result of this paper is an algorithm, HEADSCHEDULE, for the off-line problem; HEADSCHEDULE services all the requests in at most $\frac{3}{2}T_{\text{opt}} + a$ rotations, where T_{opt} is the number of rotations taken by an optimal algorithm and a is a term that depends solely on the reachability function. We can show that this problem is NP-hard and so in general it is not practical to look for the optimum algorithm. For the special case in which the reachability function is linear we present a polynomial-time algorithm, MONOTONE, and show that it is optimal. Algorithms HEADSCHEDULE and MONOTONE are presented in Sections 3 and 4 respectively.

In Section 5 we relate the disk scheduling problem to the special case of the asymmetric Traveling Salesman Problem with the triangle inequality (ATSP-Δ) in which all distances are either 0 or α for some constant α. We show how to find the optimal tour in polynomial time and describe how this gives another approximation algorithm for the disk scheduling problem. In Section 6 we consider the on-line problem. Since the requests in real systems are known to arrive in a "bursty" fashion [18] our off-line algorithms are still useful. When a burst of requests arrive we can schedule them using an off-line algorithm. We also consider the steady state model of disk arrivals of Seltzer, Chen and Ousterhout [20] in which the queue is always full. Whenever a request is serviced it is replaced in the queue by a new request chosen uniformly at random on the disk. We present an algorithm, CHAIN, which is related to our algorithm for the above ATSP-Δ and which appears to give higher throughput than the algorithms considered in [20].

Related Work One of the most famous disk head scheduling algorithms, called CSCAN, is used in the UNIX[1] operating system. In this algorithm, designed to reduce seek times, the head travels from the outermost to the innermost track, servicing all the requests in a track as the head passes over it. The scheduler does not take into account the rotational position of the request, only its track number.

Although useful in the past, this design principle currently makes less sense. In older disks the seek time was the dominating factor limiting performance. In modern disks, however, the rotational latency also plays a significant role since seek times have been decreasing at a higher rate than rotational latency. Furthermore, since processor speed is increasing faster than disk speed, a modern processor, dedicated to the task of disk head scheduling, can execute algorithms that are more computationally expensive. Finally, memory capacity has increased dramatically. Thus, it is possible to buffer a large number of requests to be serviced in a highly-efficient order.

Most previous scheduling algorithms have two often-conflicting goals: increasing throughput and preventing starvation (when a request languishes in the buffer without being serviced). Now, however, nonvolatile memory (NVRAM, i.e. memory that retains its stored values even without power) is emerging as a viable technology. (NVRAM is currently used, for example, in the products Legato Prestoserve™ and the Network Appliance Corporation's FAServer™ [10].) If the disk buffer consists of NVRAM, then for writes, only the first goal – increasing the throughput – is important, since the nonvolatile buffer is now part of storage[2] [1, 10].

Most early papers (e.g., [11, 3, 8, 21, 9, 15, 22]) consider the problem of on-line scheduling with small queue lengths (less than 50). They focus primarily on the algorithms first-come-first-serve (FCFS), shortest-seek-first (SSF), CSCAN, and

[1] UNIX is a registered trademark of X/Open Company Limited.

[2] Servicing read requests is not considered as much of a potential bottleneck, because many reads can be avoided as cache sizes increase.

modifications and generalizations of these algorithms. In [20] and [12], the authors examine algorithms that depend on rotational position. They test the algorithms with queues having many (up to 1000) random uniformly-distributed requests. In [23], the authors test disk scheduling algorithms on traces of actual disk executions. Algorithms for disk scheduling that are based on the Traveling Salesman Problem are presented in [6]. Other papers dedicated to modeling disks include [24, 19, 13].

2. The Model

The Disk Scheduling Problem A computer disk is in the shape of an annulus. We let the radial distance between the inner and outer circles be 1. The disk rotates at a constant rate. A movable *disk head* travels in and out radially in order to access locations on the disk. (In our presentation we consider the motion of the head relative to the disk and so the head not only moves radially but also moves around the disk at a constant rate.) An instance of the disk scheduling problem consists of a set of n locations on the disk. These n locations represent *requests* to be serviced by the disk head. To *service* a request, the disk head must be at the request and have no radial movement. A solution to the disk problem is a path of the disk head that services all n requests and that begins and ends at the same location on the disk. An optimal solution requires the minimum number of rotations to service all the requests.

The Reachability Function Associated with a disk drive is a function $f(\theta)$, which we call the *reachability function*. In a rotation of angle θ, function $f(\theta)$ represents the maximum radial distance the head can travel when it starts and ends with no radial movement. Since the annulus has thickness 1, we have $0 \leq f(\theta) \leq 1$. For convenience, we let $f(\theta) = 0$ for $\theta \leq 0$. Thus, from any starting-point the function $f(\theta)$ defines the reachable region in the θ-r plane; we call this region the *reachability cone*. The reachability function f has the following properties.

1. Function f is nondecreasing since given more time the disk head can visit a larger fraction of the disk. That is, $f' \geq 0$ where f' is the first derivative of f. (We can assume that the derivative, f', exists since the radial speed of the disk head is well-defined.)

2. Function f is convex, implying that the slope of f is nondecreasing. The intuition for convexity is as follows. The head accelerates as much as possible and stays at the maximum radial speed as long as possible (if the maximum speed is reached) before decelerating.

3. Properties 1 and 2 imply that $f(\theta + \theta') \geq f(\theta) + f(\theta')$ for $\theta, \theta' \geq 0$.

We define t_{fullseek} to be the minimum number of rotations (not necessarily integral) required for the head to travel the entire radial distance. That is, $2\pi t_{\text{fullseek}} = \text{argmin}_\theta f(\theta) = 1$. On modern disks $1 \leq t_{\text{fullseek}} < 3$, and usually $t_{\text{fullseek}} < 2$.

The Representation of the Disk and the Requests For ease of presentation we view the disk as a $2\pi \times 1$ rectangle. Each request R_i is specified by coordinates (θ_i, r_i), where $0 \leq r_i \leq 1$ and $0 \leq \theta_i < 2\pi$. The *distance* between two requests $R_i = (\theta_i, r_i)$ and $R_j = (\theta_j, r_j)$ is defined by,

$$\begin{aligned} d(R_i, R_j) &= d((\theta_i, r_i), (\theta_j, r_j)) \\ &= \min\{\text{integers } k : f(\theta_j - \theta_i + 2k\pi) \geq |r_j - r_i|\}. \end{aligned}$$

In other words, the distance from request R_i to R_j is equal to the number of times that the head must cross the line $\theta = 0$ when traveling from R_i to R_j.[3] Note that this distance is *asymmetric*. To reflect the "rotational nature" of the disk we also use $(\theta_i + 2k\pi, r_i)$ to denote request R_i, and we sometimes represent multiple copies of the disk by a $2k\pi \times 1$ rectangle.

The *disk graph* is a directed graph whose vertices are the requests and whose directed edges are the ordered pairs of vertices. The weight on the directed edge R_iR_j is $d(R_i, R_j)$.

3. A 3/2-Approximation Algorithm

In this section we present an algorithm that services all of the requests on the disk in at most $\frac{3}{2}T_{\text{opt}} + a$ rotations, where T_{opt} is the number of rotations required by an optimal algorithm and a is a term that depends solely on the reachability function, not on the number of requests. It is not practical to look for an optimal solution since we can show that the problem is NP-hard. The reduction is from the directed Hamiltonian cycle problem in which every vertex is incident to at most 3 arcs. This is known to be NP-complete. (See [7, 17].) The details of the reduction are left for the full version.

The Minimum-cost Cycle Cover and a Lower Bound

We first use a minimum-cost cycle cover of the disk graph to derive a lower bound LB for T_{opt}, and then present an algorithm that services all the requests in $\frac{3}{2}LB + a$ rotations, for a constant a. For a graph G, let $\mathcal{C}$ denote a collection of cycles in G. If every node of G is contained in exactly one cycle, then $\mathcal{C}$ is called a *cycle cover* of G. For edge-weighted graphs the cost of a cycle cover, $\mathcal{C}$, is the sum of the weights of the edges in $\mathcal{C}$. A *minimum-cost cycle cover* of G has the minimum cost among all the cycle covers of G. The problem of finding a minimum-cost cycle cover is equivalent to solving an assignment problem on the matrix of edge weights [5]. If G has n nodes then we can solve the assignment problem and

[3] The distance between R_i and R_j could be defined as the angular distance through which the head must travel in order to service R_i and then R_j. However, our integral definition of distance facilitates the proofs in Sections 3 and 5.

hence find a minimum-cost cycle cover in $O(n^3)$ time using the Hungarian method of Kuhn [14, 16].

Let $\mathcal{C}$ denote a minimum-cost cycle cover of the disk graph, and let $C^{(i)}$ denote the set of cycles in $\mathcal{C}$ with cost i. Let p be the maximum cost of a cycle in $\mathcal{C}$. Then $\mathcal{C} = C^{(1)} \cup C^{(2)} \cup \ldots \cup C^{(p)}$. Let K be the total cost of $\mathcal{C}$, i.e. $K = \sum_{i=1}^{p} i|C^{(i)}|$, where $|C^{(i)}|$ is the number of cycles in $C^{(i)}$. Note that K is a lower bound on T_{opt}, since an optimal solution to the disk scheduling problem is a cycle cover. As shown below, our algorithm finds an order in which to service the cycles in $\mathcal{C}$ such that the disk head can move between the cycles without using "too many" rotations. An approach based on finding a minimum-cost cycle cover was independently proposed by [6], but no performance guarantees were provided for the resulting algorithms.

The Virtual Trace

Before describing the algorithm in detail, we need to define a *virtual trace* to connect neighboring requests on a cycle. A virtual trace does not describe the actual trace of the disk head, but rather an imaginary path defined by the reachability function f. Consider a cycle $c \in C^{(i)}$. Let $R_j = (\theta_j, r_j)$, for $1 \leq j \leq m$, be the requests on c. Request $R_1 = (\theta_1, r_1)$ satisfies $\theta_1 = \min_{1 \leq j \leq m} \theta_j$, and $R_j R_{j+1}$ and $R_m R_1$ are directed edges in the cycle cover. For each $c \in C^{(i)}$ we shall view the disk as a $2i\pi \times 1$ rectangle, $T^{(i)}$, i.e. i copies of the disk are joined end to end. As demonstrated in Figure 1, we represent the requests on cycle c so that every request appears exactly once in rectangle $T^{(i)}$. Formally, R_1 appears at location (ϕ_1, r_1) for $\phi_1 = \theta_1$. Request R_j for $2 \leq j \leq m$ appears at (ϕ_j, r_j), where $\phi_j = \theta_j - \theta_{j-1} + \phi_{j-1} + 2k_j\pi$ and $k_j = d(R_{j-1}, R_j)$.

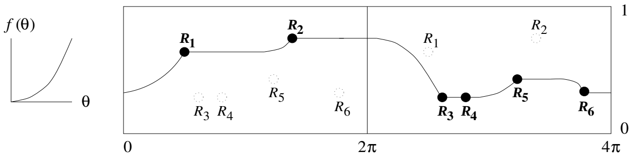

Figure 1. (Left) The reachability function f. (Right) A cycle $c \in C^{(2)}$, whose neighboring requests are connected by the virtual trace. The disk is viewed as a $4\pi \times 1$ rectangle $T^{(2)}$.

Consider two neighboring requests R_j and R_{j+1}, which appear at locations (ϕ_j, r_j) and (ϕ_{j+1}, r_{j+1}) respectively. The trace connecting them is composed of a horizontal line (of possibly zero length) followed by a curve defined by f or $-f$. (See Figure 1.) Formally, the virtual trace is defined as follows. By the definition of ϕ_j and ϕ_{j+1}, one can verify that there exists $\phi' \in [\phi_j, \phi_{j+1}]$ such that $f(\phi_{j+1} - \phi') = |r_j - r_{j+1}|$. For $r_{j+1} \geq r_j$ let,

$$g_c(\theta) = \begin{cases} r_j & \text{for } \phi_j \leq \theta \leq \phi' \\ r_j + f(\theta - \phi') & \text{for } \phi' < \theta \leq \phi_{j+1} \end{cases}.$$

The virtual trace between R_j and R_{j+1} is defined parametrically by $(\theta, g_c(\theta))$ for $\phi_j \leq \theta \leq \phi_{j+1}$. The case in which $r_{j+1} < r_j$ is analogous. The trace from R_m to R_1 is obtained from the trace connecting (ϕ_m, r_m) and $(2i\pi + \phi_1, r_1)$. If R_1 is the only request on c then the trace is the horizontal line $r = r_1$. The virtual trace of cycle $c \in C^{(i)}$ has the following properties. (See the appendix for the proofs.)

Lemma 1 *If the virtual trace can connect two requests within an angle θ then the disk head can service both of them within a rotation through angle θ.*

Lemma 2 *If R_j and R_k are two requests on cycle $c \in C^{(i)}$ and they appear at (ϕ_j, r_j) and (ϕ_k, r_k) respectively, then $|r_j - r_k| \leq f(i\pi)$.*

Lemma 3 *For a cycle $c \in C^{(i)}$, the slope of the virtual trace $(\theta, g_c(\theta))$ is between $-f'(i\pi)$ and $f'(i\pi)$ for $0 \leq \theta \leq 2i\pi$.*

Recall that $f(i\pi)$ is the radial distance that the head can travel after $i/2$ rotations of the disk, given that the head starts and ends at rest. Let $q_i = \lceil 1/f(i\pi) \rceil$. For each $C^{(i)}$ the rectangle $T^{(i)}$ is divided into smaller rectangles $T_1^{(i)}$, $T_2^{(i)}$, $\ldots$, $T_{q_i}^{(i)}$, each of size $2i\pi \times f(i\pi)$. A request $R = (\theta, r)$ is *in* rectangle $T_j^{(i)}$ if and only if $(j-1)f(i\pi) \leq r < j \cdot f(i\pi)$. We have,

Lemma 4 *If $(0, g_c(0))$ is in rectangle $T_j^{(i)}$ then the trace of cycle c either stays in rectangles $T_j^{(i)}$ and $T_{j+1}^{(i)}$ or else it stays in rectangles $T_j^{(i)}$ and $T_{j-1}^{(i)}$.*

For a cycle $c \in C^{(i)}$ let the *centerpoint* of cycle c, denoted by centerpoint(c), be the point $(i\pi, g_c(i\pi))$ on the virtual trace, and let the *leftpoint* of cycle c, denoted by leftpoint(c), be the point $(0, g_c(0)) = (2i\pi, g_c(2i\pi))$ on the trace. For an angle $\rho \in [0, 2i\pi)$, let α_c be the first request on the virtual trace of c that appears *after* the line $\theta = \rho$. We use the phrase *cycle c is serviced starting at angle ρ* to mean that α_c is the first request on c to be serviced and that the other requests on c are serviced in the order of the cycle.

The Algorithm

The algorithm HEADSCHEDULE is shown in Figure 2. It proceeds by finding a min-cost cycle cover of the disk graph and then determining a particular order in which to service the cycles. For each cycle, HEADSCHEDULE identifies the first request to visit and then travels around the cycle servicing all of the requests. Our goal is to show that the disk head can

HEADSCHEDULE

find a min-cost cycle cover $\mathcal{C} = C^{(1)} \cup \ldots \cup C^{(p)}$
service short cycles
for $i = 1$ to $2\lceil t_{\text{fullseek}} \rceil + 1$ **do**
  if i is odd **then**
    for $j = 1$ to q_i **do**
      CYCLECONNECT $(H_j^{(i)}, V_j^{(i)})$
  if i is even **then**
    for $j = q_i$ down to 1 **do**
      CYCLECONNECT $(H_j^{(i)}, V_j^{(i)})$
service long cycles
while there exist unserviced long cycles c
  service cycle c starting at angle 0

CYCLECONNECT $(H_j^{(i)}, V_j^{(i)})$

while there exist unserviced cycles in $H_j^{(i)}$, alternate between the following.
- Let c be the unserviced cycle in $H_j^{(i)}$ which has the highest centerpoint. Service c starting at angle 0.
- Let c be the unserviced cycle in $H_j^{(i)}$ which has the lowest leftpoint. Service c starting at angle $i\pi$.

while there exist unserviced cycles in $V_j^{(i)}$, alternate between the following.
- Let c be the unserviced cycle in $V_j^{(i)}$ which has the lowest centerpoint. Service c starting at angle 0.
- Let c be the unserviced cycle in $V_j^{(i)}$ which has the highest leftpoint. Service c starting at angle $i\pi$.

Figure 2. The HEADSCHEDULE algorithm.

connect the last request of a cycle to the first request of the next cycle without using "too many" rotations.

A cycle $c \in C^{(i)}$ is *long* if $i \geq 2L$ where $L = \lceil t_{\text{fullseek}} \rceil + 1$; otherwise c is *short*. Note that in L rotations, the head can travel from any request to any other request. The long cycles can therefore be serviced in any order and the number of rotations required is at most

$$\sum_{i=2L}^{p} i|C^{(i)}| + \sum_{i=2L}^{p} L|C^{(i)}| \leq \frac{3}{2} \sum_{i=2L}^{p} i|C^{(i)}|. \quad (1)$$

Hence, the approximation ratio of $3/2$ is achieved for servicing the long cycles.

We therefore focus on the order in which HEADSCHEDULE services the short cycles. The algorithm first services the cycles in $C^{(1)}$ and then the cycles in $C^{(2)}$, etc. By Lemma 4, cycles in $C^{(i)}$ can be divided into the following groups.

1. The *hill group* $H_j^{(i)}$ consists of cycles c whose leftpoint $(0, g_c(0))$ is in rectangle $T_j^{(i)}$ and whose centerpoint $(i\pi, g_c(i\pi))$ is in rectangle $T_j^{(i)}$ or $T_{j+1}^{(i)}$.
2. The *valley group* $V_j^{(i)}$ consists of cycles c whose leftpoint is in rectangle $T_j^{(i)}$ and whose centerpoint is in rectangle $T_{j-1}^{(i)}$.

To service the cycles in $C^{(i)}$, HEADSCHEDULE services the requests in rectangle $T^{(i)}$ from bottom to top when i is odd and services requests in $T^{(i)}$ from top to bottom when i is even. To be more precise, for odd i HEADSCHEDULE first services the cycles in $H_1^{(i)}$ and then the cycles in $V_1^{(i)}$, $H_2^{(i)}$ and $V_2^{(i)}$, etc; for even i HEADSCHEDULE first services cycles in $H_{q_i}^{(i)}$ and then cycles in $V_{q_i}^{(i)}$, $H_{q_{i-1}}^{(i-1)}$ and $V_{q_{i-1}}^{(i-1)}$, etc. The subroutine CYCLECONNECT specifies the order in which HEADSCHEDULE services the cycles in $H_j^{(i)}$ and $V_j^{(i)}$ and also identifies the first request to be serviced on each cycle. (See Figures 2 and 3.) Hence, the order in which HEADSCHEDULE services all the requests on the disk is fully determined.

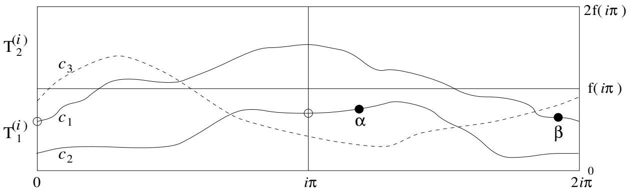

Figure 3. CYCLECONNECT **services the cycles in $H_1^{(i)}$ in the order c_1, c_2, c_3, since c_1 has the highest centerpoint and c_2 has the lowest leftpoint. To illustrate Lemma 6, β is serviced last on cycle c_1 and α is serviced first on c_2. Request α is reachable from leftpoint(c_1).** CYCLECONNECT **therefore uses $i/2$ rotations to travel from c_1 to c_2.**

To complete the analysis, we shall show that the approximation ratio of $3/2$ is achieved for servicing short cycles. In particular, we show in Lemma 6 that HEADSCHEDULE uses $\frac{3}{2}i|H_j^{(i)}|$ rotations (resp. $\frac{3}{2}i|V_j^{(i)}|$ rotations) to service all the cycles in $H_j^{(i)}$ (resp. $V_j^{(i)}$).

Let γ be a point of the form $(0, r_\gamma)$ and consider the reachability cone rooted at γ. Let functions $h_1(\theta)$ and $h_2(\theta)$ define the upper and lower boundaries of the cone, i.e. $h_1(\theta) = r_\gamma + f(\theta)$ and $h_2(\theta) = r_\gamma - f(\theta)$. For a cycle $c \in C^{(i)}$ let $\alpha_c = (\phi_c, r_c)$ be the first request on c that appears after the line $\theta = i\pi$. For simplicity we assume that $\phi_c > i\pi$. (For the case in which $\phi_c \leq i\pi$, the location of α_c can be taken at $(2i\pi + \phi_c, r_c)$ for the analysis.) Figure 4 illustrates Lemma 5 and its proof.

Lemma 5 *If the centerpoint of cycle c is in the reachability cone rooted at γ, then α_c is in the reachability cone rooted at γ. That is, if $h_1(i\pi) \leq g_c(i\pi) \leq h_2(i\pi)$, then $h_1(\phi_c) \leq g_c(\phi_c) \leq h_2(\phi_c)$.*

Proof: The definition of h_1 and h_2 implies that $h_1'(\theta) = f'(\theta)$ and $h_2'(\theta) = -f'(\theta)$. By Lemma 3, the virtual trace of cycle c never has a slope whose absolute value is greater than $f'(i\pi)$. Since f' is nondecreasing by Property 2 of f, we have $h_2'(\theta) \leq g_c'(\theta) \leq h_1'(\theta)$ for $\theta \geq i\pi$. Therefore, if $h_1(i\pi) \leq g_c(i\pi) \leq h_2(i\pi)$ then $h_1(\phi_c) \leq g_c(\phi_c) \leq h_2(\phi_c)$ for $\phi_c \geq i\pi$. Stated differently, if the centerpoint of the virtual trace is in the reachability cone, then the trace can never leave the cone, i.e. the point $(\theta, g_c(\theta))$ is in the cone for any $\theta \geq i\pi$. □

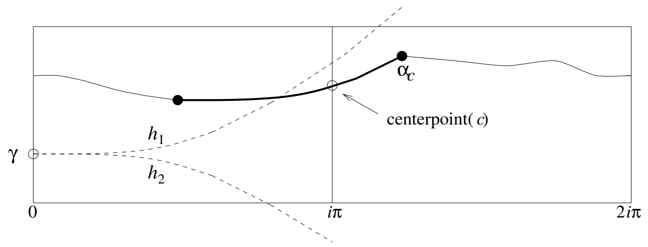

Figure 4. The two dashed curves represent functions h_1 and h_2, which define the upper and lower boundaries of the reachability cone rooted at the point γ. If centerpoint(c) is in the reachability cone then α_c, the first request on c after centerpoint(c), is also in the cone.

Lemma 6 *Subroutine* CYCLECONNECT *services all the cycles in $H_j^{(i)}$ (resp. $V_j^{(i)}$) in $\frac{3}{2}i|H_j^{(i)}|$ rotations (resp. $\frac{3}{2}i|V_j^{(i)}|$ rotations).*

Proof: Stated intuitively, we show that CYCLECONNECT uses $i/2$ rotations to travel to the next cycle and then uses i rotations to service all the requests on this cycle. Let $c_h \in H_j^{(i)}$ and $c_\ell \in H_j^{(i)}$ be the unserviced cycles which have the highest centerpoint and lowest leftpoint, respectively. Suppose that CYCLECONNECT services c_h followed by c_ℓ. Let β_h be the last request on cycle c_h before leftpoint(c_h) and let α_ℓ be the first request on cycle c_l after centerpoint(c_ℓ). (Note that β_h is serviced last on cycle c_h and α_ℓ is serviced first on c_ℓ. See Figure 3.) By the definition of the virtual trace it is clear that leftpoint(c_h) is *reachable* from β_h, i.e. leftpoint(c_h) is in the reachability cone rooted at β_h. The following two arguments show that centerpoint(c_ℓ) is in the reachability cone rooted at leftpoint(c_h).

Case 1: *Centerpoint(c_ℓ) is in $T_j^{(i)}$.* Since the height of $T_j^{(i)}$ is $f(i\pi)$, centerpoint(c_ℓ) is reachable from leftpoint(c_h).

Case 2: *Centerpoint(c_ℓ) is in $T_{j+1}^{(i)}$.* Centerpoint(c_h) is higher than centerpoint(c_ℓ) by the definition of c_h. Since centerpoint(c_h) is reachable form leftpoint(c_h), centerpoint(c_ℓ) is reachable from leftpoint(c_h).

Lemma 5 implies that α_ℓ is in this reachability cone and is hence reachable from β_h. Therefore the head services the requests in c_h, moves to cycle c_ℓ and services the requests in c_ℓ in $i + i/2 + i = 5i/2$ rotations. An analogous argument can be applied to show that after servicing any cycle in $H_j^{(i)}$ (resp. $V_j^{(i)}$) the next cycle can be serviced in $3i/2$ rotations. The result follows. □

Lemma 6 implies that CYCLECONNECT services all of the requests in $C^{(i)}$ in $\frac{3}{2}i|C^{(i)}| + 2iq_i$ rotations. The additive term is due to the rotations required for connecting hill groups and valley groups. It follows that CYCLECONNECT services all the short cycles in

$$\sum_{i=1}^{2L-1} \frac{3}{2}i|C^{(i)}| + \sum_{i=1}^{2L-1} 2iq_i \tag{2}$$

rotations, where $L = \lceil t_{\text{fullseek}} \rceil + 1$. (Note that the additive term $\sum_{i=1}^{2L-1} 2iq_i$ is also sufficient to cover the rotations required to connect the last cycle in $C^{(i)}$ and the first cycle in $C^{(i+1)}$.) Since $q_i = \lceil 1/f(i\pi) \rceil$ and $f(i\pi) \geq if(\pi)$ by Property 3 of f, we have $iq_i \leq i + q_1$. Hence, $\sum_{i=1}^{2L-1} 2iq_i \leq 2Lq_1 + 2L^2$. Combined with the analysis for long cycles (see inequality 1) we have,

Theorem 7 *The algorithm* HEADSCHEDULE *has a $3/2$-approximation ratio with an additive term of at most $2Lq_1 + 2L^2$, where $L = \lceil t_{\text{fullseek}} \rceil + 1$.*

4. An Optimal Algorithm for Linear Reachability Functions

Although the disk scheduling problem is NP-hard for general reachability functions, optimal solutions can be obtained for a special case. In this special case, the head either has no radial movement or else has full radial speed s. The reachability function is therefore linear, i.e. $f(\theta) = s\theta$. In addition, we require that the disk head starts at the point $(0, 0)$ and ends at $(0, 1)$.

One can verify that the linearity of f ensures the *reachability property*. That is, regardless of its current speed, the head can follow any head path passing through its current position. To be more precise, suppose that on one rotation the disk head goes from point A to point B to point C, and on another rotation the disk head goes from point D to point B to point E. Then the head can go from A to B to E or from D to B to C. (Note that A, B, etc. are any points on the head path, not necessarily requests.) The reachability property does not hold for general reachability functions.

Suppose that P is a head path that services all the requests. If P requires m rotations let $(\phi, g_P(\phi))$, $0 \leq \phi \leq 2m\pi$, be the location of the head after it rotates through an angle ϕ. Path P satisfies the *monotone property* if $g_P(\phi) \leq g_P(\phi+2k\pi)$ for any ϕ and positive integer k, where $0 \leq \phi \leq \phi+2k\pi \leq 2m\pi$.

Lemma 8 *Suppose that the disk head must start at* $(0,0)$ *and end at* $(0,1)$. *Then there exists an optimal solution to the disk scheduling problem such that the monotone property is satisfied.*

Proof: Consider any optimal solution and its corresponding path P. Suppose that P requires m rotations. We construct a new path Q with m rotations such that Q preserves monotonicity and can be followed by the disk head. In particular, for any angle $\theta \in [0, 2\pi)$ and integer $i \in [0, m-1]$, let $g_Q(\theta+2i\pi)$ be the ith smallest value from $g_P(\theta)$, $g_P(\theta+2\pi)$, ..., $g_P(\theta + 2(m-1)\pi)$. (See Figure 5.) By construction, the path Q that corresponds to g_Q takes m rotations and preserves monotonicity. The reachability property implies that Q is realizable by the disk head. □

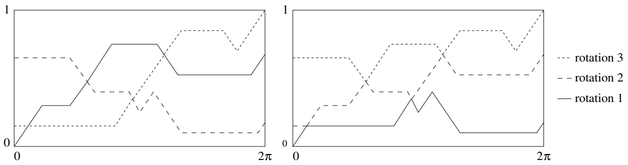

Figure 5. (Left) An optimal path P with $m = 3$ rotations. (Right) Path Q with 3 rotations which preserves monotonicity and is realizable by the disk head.

We now describe a situation in which monotonicity is violated. Consider the point (ϕ_0, r_0). The region *under* (ϕ_0, r_0) consists of points (ϕ, r), where,

$$\begin{cases} 0 \leq r < r_0 - f(\phi - \phi_0) & \text{for} \quad \phi > \phi_0 \\ 0 \leq r < r_0 - f(\phi_0 - \phi) & \text{for} \quad \phi \leq \phi_0. \end{cases}$$

If a request $R = (\theta, r)$, $0 \leq \theta < 2\pi$, is serviced on the $(k+1)$st rotation, then we say that R is serviced at angle $2k\pi + \theta$. We have the following.

Lemma 9 *Suppose request* $R = (\theta, r)$ *is serviced at angle* $2k\pi + \theta$. *If, when* R *is serviced, there are unserviced requests in the region under* $(2k\pi + \theta, r)$, *then monotonicity is violated.*

Proof: Let g be the function that describes the head path. Suppose that an unserviced request $U = (\theta_u, r_u)$, $0 \leq \theta_u < 2\pi$, is under the point $(2k\pi + \theta, r)$. We first assume $\theta_u < \theta$, then $g(2k\pi + \theta) = r$ and $g(2\ell\pi + \theta_u) = r_u$ for some $\ell > k$. Since U is unserviced and in the region under $(2k\pi+\theta, r)$, we must have $g(2\ell\pi + \theta) < r$ by definition of the region under a point, i.e. $g(2\ell\pi + \theta) < g(2k\pi + \theta)$ for some $\ell > k$. The case in which $\theta_u \geq \theta$ is similar. □

An optimal algorithm. We present an optimal algorithm MONOTONE that services the requests in the following order. The disk head starts at $(0,0)$. Let the current head position be (ϕ_0, r_0) where ϕ_0 is the actual angle through which the head has rotated. Suppose that $R = (\theta, r)$, where $0 \leq \theta < 2\pi$, is the next request to be serviced by MONOTONE, and suppose that R is serviced at angle $\phi = 2k\pi + \theta$. The following conditions are used to determine R.

1. The reachability cone rooted at (ϕ_0, r_0) contains (ϕ, r);
2. There are no unserviced requests in the region under (ϕ, r);
3. Request R is the first one that is in the cone rooted at (ϕ_0, r_0) and that satisfies condition 2. Stated differently, for any unserviced request $R' = (\theta', r')$, if there exists a k' such that $\phi' = 2k'\pi + \theta'$, $\phi_0 < \phi' < \phi$ and (ϕ', r') is in the cone rooted at (ϕ_0, r_0), then there are unserviced requests under the point (ϕ', r').

By Lemma 8 there exists an optimal solution OPT such that the monotone property is satisfied. The following theorem shows that if OPT serviced some request before MONOTONE does, then OPT would have to violate monotonicity. Hence, MONOTONE cannot perform worse than OPT.

Theorem 10 *Let* OPT *be any optimal algorithm that preserves monotonicity. Algorithms* OPT *and* MONOTONE *require the same number of rotations to service all the requests.*

Proof: To obtain a contradiction we assume MONOTONE requires more rotations than OPT. Let R_1, R_2, ... be the order in which MONOTONE services all the requests. Let $R_j = (\theta_j, r_j)$ be the first request that OPT services before MONOTONE. Suppose OPT services R_j at angle $\phi_j = 2k\pi + \theta_j$, which implies that MONOTONE services R_j at an angle greater than ϕ_j. Let $R_i = (\theta_i, r_i)$, $i < j$, be the last request MONOTONE services before angle ϕ_j. Suppose MONOTONE services R_i at angle $\phi_i = 2k'\pi + \theta_i$, where $\phi_i < \phi_j$. There are two cases to consider.

Case 1. *R_j is outside the reachability cone rooted at* (ϕ_i, r_i). By condition 2, (ϕ_j, r_j) is above the cone. This implies that OPT cannot service R_i at an angle $\phi \in [\phi_i, \phi_j]$, since OPT services R_j at ϕ_j. By the definition of R_j, OPT services R_i no earlier than ϕ_i. Hence, OPT services R_i after R_j. Lemma 9 implies that OPT violates monotonicity.

Case 2. *R_j is inside the reachability cone rooted at* (ϕ_i, r_i). By the definition of R_i, MONOTONE does not service any requests between angles ϕ_i and ϕ_j. The reason that MONOTONE does not service R_j after servicing R_i is that there is some request R_ℓ under the point (ϕ_j, r_j). This request R_ℓ is not serviced by MONOTONE by angle ϕ_j, but MONOTONE services R_ℓ before R_j by the construction of the algorithm (i.e. $i < \ell < j$). Hence, the definition of R_j implies that R_ℓ is also unserviced by OPT at angle ϕ_j. Therefore, OPT services R_j

before R_ℓ. Lemma 9 implies that OPT violates monotonicity. □

5. A Special Case of the Asymmetric Traveling Salesman Problem

In this section we view the disk scheduling problem as a special case of the asymmetric traveling salesman problem with the triangle inequality (ATSP-Δ). We present an approximation algorithm for this ATSP-Δ problem and then obtain another approximation algorithm for the disk scheduling problem.

In the disk graph, edge lengths are nonnegative integers given by the distance function defined in Section 2. If the disk head makes a full seek in t_{fullseek} rotations then all edge lengths are at most $L = \lceil t_{\text{fullseek}} \rceil + 1$. However, an ATSP-$\Delta$ problem that has integer edge lengths from $[0, L]$ is not necessarily a disk scheduling problem and so the algorithms of the previous sections may not apply.

The problem we consider in this section can be formally defined as follows. We are given a graph G with n nodes and a distance function δ on these nodes. The function δ is not necessarily symmetric but it satisfies the triangle inequality, i.e. $\delta(u, v) + \delta(v, w) \geq \delta(u, w)$ for all nodes u, v and w. We first assume that all distances are 0 or α for some $\alpha > 0$. We present an optimal algorithm for this case. Secondly we assume that all distances are either 0 or else lie between α and β where $0 < \alpha < \beta$. In this case we apply the previous result to obtain a β/α-approximation algorithm. The best known approximation ratio for a general ATSP-Δ problem is $\lceil \log_2 n \rceil$. (See [5].) We have,

Theorem 11 *Let $\alpha > 0$. If $\delta(u, v) \in \{0, \alpha\}$ for all nodes u and v then the resulting* ATSP-Δ *problem is polynomially solvable.*

Proof: We define a relation on the nodes. Let $u \sim v$ if and only if $\delta(u, v) = \delta(v, u) = 0$. By the triangle inequality this is an equivalence relation. Let $V_1, V_2, \ldots$ be the equivalence classes induced by this relation. Define the distance from equivalence class V_i to class V_j to be $\delta'(V_i, V_j) = \min\{\delta(u, v) : u \in V_i, v \in V_j\}$. One can verify that the triangle inequality holds for δ' and that if $\delta'(V_i, V_j) = 0$ then $\delta'(V_j, V_i) \neq 0$. Consider now a directed graph, H, whose nodes are the equivalence classes. A directed edge (V_i, V_j) exists if and only if $\delta'(V_i, V_j) = 0$. The graph H is acyclic, otherwise there would exist V_i and V_j such that $\delta'(V_i, V_j) = \delta'(V_j, V_i) = 0$. By the triangle inequality on δ' and the acyclicity of H, graph H induces a partial order, $(\mathcal{P}, \prec)$, on the equivalence classes. We say that $V_i \prec V_j$ in $\mathcal{P}$ if and only if there is a path from V_i to V_j in H. Two elements V_i and V_j in $\mathcal{P}$ are *comparable* if either $V_i \prec V_j$ or $V_j \prec V_i$, and they are *incomparable* otherwise. An *antichain* is a set of elements any two of which are incomparable. A *chain* is a set of elements any two of which are comparable.

Lemma 12 *Let A be the maximum cardinality of an antichain. The length of an optimal tour for our* ATSP-Δ *problem is at least αA.*

Proof: Let $\{V_1, V_2, \ldots, V_A\}$ be an antichain of size A. Since, for all i and j, $V_i \not\prec V_j$ and $V_j \not\prec V_i$, we have $\delta'(V_i, V_j) = \delta'(V_j, V_i) = \alpha$. For all i let v_i be any member of V_i. (Recall that V_i is an equivalence class of nodes in graph G.) The definition of δ' implies that $\delta(v_i, v_j) = \delta(v_j, v_i) = \alpha$ for $1 \leq i, j \leq A$. The optimal traveling salesman tour must visit all these nodes v_i. Hence the tour has length at least αA. □

It remains to show that we can find a tour that achieves this lower bound. Our algorithm is based on the following theorem. See [4, 2].

Dilworth's Theorem *If the largest antichain in a partial order $(\mathcal{P}, \prec)$ has cardinality A, then the partial order can be decomposed into exactly A chains. Moreover this decomposition can be obtained in polynomial time.*

It is clear that no decomposition can have fewer than A chains since every element of the antichain must be in a different chain. What is remarkable is that there always exist A chains that cover the whole partial order. (See [2] for a proof.)

An optimal tour is constructed from the chains in a minimum-size chain decomposition. Under the distance function δ', the total length of a chain is 0 and the distance from the end of any chain to the beginning of any other chain is at most α. Given that the size of the maximum antichain is A, we can therefore link the chains into a cycle of length at most αA. Note that this is a tour of graph H. To obtain a tour of G, we observe that once a tour has visited one node in an equivalence class it can visit all the other nodes in that class in any order without increasing its length. Hence we can construct a tour of G that has length at most αA. By Lemma 12 this tour is optimal. □

Corollary 13 *Let $\beta > \alpha > 0$. If either $\delta(u, v) = 0$ or $\alpha \leq \delta(u, v) \leq \beta$ for all nodes u and v then there exists a β/α-approximation algorithm for the resulting* ATSP-Δ *problem.*

If we assume that all nonzero distances are α and apply Theorem 11, then Corollary 13 follows. By the comments at the beginning of this section, if the disk head can make a full seek in t_{fullseek} rotations then Corollary 13 gives a $(\lceil t_{\text{fullseek}} \rceil + 1)$-approximation algorithm for the disk scheduling problem. (Typically $t_{\text{fullseek}} < 2$.)

6. The On-line Problem

In this section we consider the on-line model of the disk scheduling problem. Requests arrive over time and are placed into a queue. The disk head can only service requests that are in the queue. This situation may be viewed as an on-line problem in which we have limited look-ahead. In real systems the requests are known to arrive in a "bursty" fashion [18] and so the preceding analysis of the off-line problem is useful. Suppose a large group of requests arrive together and then there is a period in which no requests arrive. We can use an off-line algorithm to service these requests.

For the remainder of this section we focus on the steady state model of disk arrivals used in [20]. The disk is discretized so that there are finitely many points for which a request can be made. This is achieved by dividing the disk into t tracks and s sectors per track. A request must be a point that is on a track and at the beginning of a sector. At all times the queue contains a fixed number, q, of requests (not necessarily distinct). The disk head is only allowed to service one of the requests in the queue. Whenever a request is serviced it is replaced by a new request chosen *uniformly at random* from the allowable points on the disk. The *throughput* is defined to be the number of requests serviced divided by the number of sectors that have passed under the disk head. That is, if the disk has made r rotations and x requests have been serviced then the throughput is x/sr. Note that throughput is dependent on q as well as the algorithm used. If q is small then any algorithm will yield low throughput. In [20] the authors simulated a number of algorithms and measured their throughput. Among the algorithms they tested, SHORTEST-TIME-FIRST (STF) has the highest throughput. This algorithm always aims to service the request that it can reach with the smallest amount of rotation.

We describe an algorithm CHAIN for this on-line problem that is similar in spirit to the algorithms of Section 5. Unfortunately, we are currently unable to determine its throughput analytically. We believe that CHAIN is of interest however because preliminary simulations indicate that it gives a higher throughput than STF. For $s = 8$, $t = 40$ and $q = 40$ and using the reachability function from [20], the disk throughput of CHAIN is 25.0% whereas the throughput of STF is 23.7%. Hence CHAIN services 5% more requests than STF in a given time period. This contradicts the conjecture of [20] that STF yields the best throughput. More extensive simulation results will be given in the full version.

The algorithm Consider the q requests that are in the queue. We construct a partial order on $q + 1$ points, namely the q requests and the current position of the disk head, $P = (\theta_0, r_0)$. For simplicity assume $\theta_0 = 0$. We say that two points $R_i = (\theta_i, r_i)$ and $R_j = (\theta_j, r_j)$ (where $\theta_i, \theta_j \in [0, 2\pi)$) satisfy $R_i \prec R_j$ if and only if $d(R_i, R_j) = 0$. (Recall the distance function defined in Section 2.) It can be verified that this defines a partial order. Note that P is a minimal element in this partial order. Algorithm CHAIN proceeds by finding the longest chain whose minimum element is P. It then moves the disk head to the request that is directly above P in this chain. (If the longest chain contains P only then the algorithm moves the disk head to an arbitrary request.) This request is replaced in the queue with a new randomly-chosen request and the algorithm repeats, constructing a new partial order.

Acknowledgements

The authors wish to thank Michel Goemans, David Karger, Jon Kleinberg, Charles Leiserson, Margo Seltzer, Chris Small, Keith Smith, and John Wilkes for many helpful comments.

References

[1] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1992.

[2] K. Bogart. *Introductory Combinatorics*. Harcourt Brace Jovanovich, Orlando, Florida, 1990.

[3] E. G. Coffman, L. A. Klimko, and B. Ryan. Analysis of scanning policies for reducing disk seek times. *SIAM Journal of Computing*, 1(3), September 1972.

[4] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Ann. Math.*, 51(2):161–166, 1950.

[5] A. M. Frieze, G. Galbiati, and F. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12:23–39, 1982.

[6] G. Gallo, F. Malucelli, and M. Marrè. Hamiltonian paths algorithms for disk scheduling. Technical Report 20/94, Dipartimento di Informatica, Università di Pisa, 1994.

[7] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-Completeness*. Freeman, New York, 1979.

[8] R. Geist and S. Daniel. A continuum of disk scheduling algorithms. *ACM Transactions on Computer Systems*, 5(1):77–92, February 1987.

[9] C. C. Gotlieb and H. MacEwen. Performance of movable-head disk storage devices. *Journal of the ACM*, 20(4), October 1973.

[10] D. Hitz, J. Lau, and M. Malcolm. File system design for an NFS file server appliance. In *USENIX*, pages 235–245, Winter 1994.

[11] M. Hofri. Disk scheduling: FCFS vs SSTF revisited. *Communications of the ACM*, 23(11), November 1980.

[12] D. M. Jacobson and J. Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL–CSP–91–7rev1, Hewlett-Packard Company, 1991.

[13] D. Kotz, S. B. Toh, and S. Radhakrishnan. A detailed simulation model of the HP 97560 disk drive. Technical Report PCS–TR94–220, Dartmouth College, 1994.

[14] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83 – 97, 1955.

[15] W. C. Oney. Queuing analysis of the scan policy for moving-head disks. *Journal of the ACM*, 22(3), July 1975.

[16] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, New Jersey, 1982.

[17] J. Plesnik. The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two. Unpublished manuscript, 1978.

[18] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *USENIX*, pages 405–420, Winter 1993.

[19] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, March 1994.

[20] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *USENIX*, pages 313–324, Winter 1990.

[21] T. J. Teorey and T. B. Pinkerton. A comparative analysis of disk scheduling policies. *Communications of the ACM*, 15(3), March 1972.

[22] N. C. Wilhelm. An anomaly in disk scheduling: A comparison of FCFS and SSTF seek scheduling using an empirical model for disk accesses. *Communications of the ACM*, 9(1), January 1976.

[23] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling algorithms for modern disk drives. In *SIGMETRICS*, pages 241–251, 1994.

[24] B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes. On-line extraction of SCSI disk drive parameters. In *SIGMETRICS*, pages 146–156, 1995.

Appendix

The Proofs

Proof of Lemma 1: The result follows from the definitions of the virtual trace and the reachability function f. $\square$

Proof of Lemma 2: Without loss of generality, we assume $j \leq k$. Either $\phi_k - \phi_j \leq i\pi$ or $(2i\pi + \phi_j) - \phi_k \leq i\pi$. If the former case holds then,

$$\begin{aligned} |r_k - r_j| &\leq \sum_{\ell=j+1}^{k} f(\theta_\ell - \theta_{\ell-1}) \\ &\leq f(\sum_{\ell=j+1}^{k} (\theta_\ell - \theta_{\ell-1})) \leq f(i\pi). \end{aligned}$$

The first inequality follows the definition of the reachability function. The second and third inequalities follow from properties 3 and 1 of f respectively. A similar argument applies for the case in which $(2i\pi + \phi_j) - \phi_k \leq i\pi$. $\square$

Proof of Lemma 3: By construction, a virtual trace g_c for cycle c is composed of the curves defined by f and $-f$. In particular, $g_c(\theta) = r_k \pm f(\theta - \phi')$ for $\phi_k \leq \theta \leq \phi_{k+1}$ and $\phi' \in [\phi_k, \phi_{k+1}]$. (Recall that $f(\theta) = 0$ for $\theta < 0$.) Lemma 2 implies that $\phi_{k+1} - \phi' \leq i\pi$. Property 2 of f therefore implies the result. $\square$

Proof of Lemma 4: If (θ_k, r_k) is some request on $c \in C^{(i)}$ then by Lemma 2 the trace of c stays in the horizontal stripe defined by $r_k - f(i\pi) \leq r \leq r_k + f(i\pi)$. Since rectangles $T_j^{(i)}$ are of size $2i\pi \times f(i\pi)$, the result follows. $\square$

Disk Specifications

The following specifications are for the Hewlett-Packard 97560 disk. See [19].

sector size (bytes)	512
number of cylinders	1962
tracks per cylinder	19
data sectors per track	72
revolution speed (rpm)	4002
seek time (ms) for d tracks	
$d \leq 383$	$3.24 + 0.400\sqrt{d}$
$d > 383$	$8.00 + 0.008d$