An Exact Algorithm for the Linear Tape Scheduling Problem

Valentin Honoré,[1] Bertrand Simon,[1] Frédéric Suter[1,2]

[1] IN2P3 Computing Center / CNRS, Lyon - Villeurbanne, France
[2] Oak Ridge National Laboratory, Oak Ridge, TN 37830
valentin.honore@cc.in2p3.fr, bertrand.simon@cc.in2p3.fr, frederic.suter@cc.in2p3.fr

Abstract

Magnetic tapes are often considered as an outdated storage technology, yet they are still used to store huge amounts of data. Their main interests are a large capacity and a low price per gigabyte, which come at the cost of a much larger file access time than on disks. With tapes, finding the right ordering of multiple file accesses is thus key to performance. Moving the reading head back and forth along a kilometer long tape has a non-negligible cost and unnecessary movements thus have to be avoided. However, the optimization of tape request ordering has rarely been studied in the scheduling literature, much less than I/O scheduling on disks. For instance, minimizing the average service time for several read requests on a linear tape remains an open question.

Therefore, in this paper, we aim at improving the quality of service experienced by users of tape storage systems, and not only the peak performance of such systems. To this end, we propose a reasonable polynomial-time exact algorithm while this problem and simpler variants have been conjectured NP-hard. We also refine the proposed model by considering U-turn penalty costs accounting for inherent mechanical accelerations. Then, we propose a low-cost variant of our optimal algorithm by restricting the solution space, yet still yielding an accurate suboptimal solution. Finally, we compare our algorithms to existing solutions from the literature on logs of the mass storage management system of a major datacenter. This allows us to assess the quality of previous solutions and the improvement achieved by our low-cost algorithm. Aiming for reproducibility, we make available the complete implementation of the algorithms used in our evaluation, alongside the dataset of tape requests that is, to the best of our knowledge, the first of its kind to be publicly released.

1 Introduction

Initially designed for media recording, the usage domain of magnetic tapes has broadened over the decades and remains a real competitor to disk storage even for scientific data. The main advantages of this storage medium are a large storage capacity for a reasonable price, a better data preservation, better security, and better energy efficiency. Indeed, it has been estimated that total costs are reduced by an average factor of 6 when archiving data on tape rather than disks (Reine and Kahn 2015).

Recent tape cartridges can store up to 20 terabytes of data on a one-kilometer-long physical storage medium, longitudinally divided into few bands which are each also longitudinally divided into dozens of wraps. Wraps are in turn divided into dozens of tracks. All tracks in a given wrap are read or written simultaneously. A tape is then composed of hundreds of parallel wraps which are logically linked together in a *linear serpentine*. Intuitively, the storage space can be seen as a single linear wrap coiled liked a serpent on the tape.

Thousands of such cartridges are usually stored on the shelves of robotic libraries, as books would be stored in an actual library. Then, when data on a given cartridge is not needed, its storage does not induce any power consumption, and it cannot be accessed by intruders. All these advantages of tape storage made it an unavoidable candidate for the storage of the exabytes of data produced at CERN by the Large Hadron Collider experiments (Davis et al. 2019) or data related to European weather forecast (Mäsker et al. 2016).

The huge amount of data stored in such tape libraries, typically hundreds of petabytes, is usually managed by a Mass Storage Management System (*e.g.,* IBM HPSS or HPE DMF) which keeps track of the exact location of the files stored on tapes and answers to users' requests. When a particular file is needed, the tape it is on will be fetched by a robotic arm, brought to a tape drive, and loaded. Then, the reading head of the tape drive is positioned to the beginning of the file to read, or to the first available space to write new data, and the I/O operation eventually occurs.

The main drawback of tape storage is the high latency to access a given file. Mounting a tape into a tape reader requires a delay of about a minute (Cano et al. 2021). Moreover, seeking from one file to another adds more delay to place the reading head on the correct wrap and adapt the longitudinal position of the tape in front of the head. When accesses to multiple files are requested, finding the right ordering of these accesses is thus key to performance. Moving the reading head back and forth along a kilometer long tape has a non-negligible cost and unnecessary movements thus have to be avoided. However, the optimization of tape request ordering has rarely been studied in the scheduling literature, much less than I/O scheduling on disks. For instance, minimizing the average service time for several read requests, *i.e.,* the average time at which each request is read, on a linear tape remains an open question.

Therefore, in this paper, we aim at improving the quality of service experienced by users of tape storage systems, and not only the peak performance of such systems. To this end, we consider a simplified model of magnetic tape composed of a single linear track. This is a strong assumption as the serpentine nature of tapes leads to important optimization decisions. However, it still reflects local batch requests which would target files belonging to the same wrap. We also believe it is a fundamental model which should be deeply understood. In this model, a tape can therefore be seen as a linear sequence of files which all have to be read from the left to the right. The input of the problem we consider is a list of files that are requested, associated with a number of requests for each file. The objective is to design a schedule (*i.e.,* a trajectory of the reading head on the linear tape) to read all the requested files when the reading head is initially positioned on the right of the tape. We consider the average service time as a metric, to ensure a fair service among all requests. In order to model the temporality of a given schedule, we assume that the speed of the tape movement is constant, although it is a mechanical device with inertia. We moderate this inaccuracy by taking into account the deceleration induced by a U-turn of the tape as a nominal penalty. Note that we do not consider write requests, which are usually done separately, nor update requests, which are avoided as they damage nearby data. Following (Cardonha and Real 2016), we refer to this problem as the Linear Tape Scheduling Problem (LTSP), noting that our model differs from theirs by accounting for U-turn penalties.

LTSP has been previously studied by Cardonha and Real (2016, 2018) and conjectured to be impossible to be solved efficiently. Indeed, even simpler variations restricting either file requests to be unique or file sizes to be equal have been conjectured NP-hard (Cardonha and Real 2018). We answer this open question in this paper by providing a polynomial algorithm optimally solving the unrestricted LTSP problem, also considering U-turn penalties. More precisely, we show that a carefully designed Dynamic Programming implementation (technique which has been considered in (Cardonha and Real 2018) but was deemed not conclusive) allows us to compute an optimal schedule in a reasonable polynomial time. We then provide a faster suboptimal algorithm and compare the performance of these two original algorithms to that of existing algorithms on a dataset built from the recent history of the IN2P3 Computing Center tape library[1].

The remainder of this paper is organized as follows. In Section 2, we review the literature on tape scheduling and related optimization problems. In Section 3, we define and discuss precisely the model and the objective function. In Section 4 we expose our algorithmic solutions to this problem. Finally, in Section 5, we present the results of our simulations on a real-world dataset.

2 Related work

The closest works to the present paper (Cardonha and Real 2016, 2018) study LTSP under the same tape model, but without U-turn penalties. The authors note that the algorithm minimizing the maximal service time, *i.e.,* the time at which all files are read, can present an average service time arbitrarily far from the optimal. They show that the opposite algorithm reading the rightmost files first is however a 3-approximation, and design a few greedy optimizations. Finally, they provide several heuristics for the online variant and compare their solutions through simulations.

LTSP is related to several well-studied problems in theoretical computer science. The most famous is probably the Traveling Salesperson Problem, where the goal is to visit n points as fast as possible following given travel times between each pair of points. This problem is notoriously NP-hard in general metrics (Lawler et al. 1985) so approximation algorithms and special cases have been studied extensively. One of the most recent development has been the design of an algorithm surpassing the long-standing approximation ratio of 1.5 (Karlin, Klein, and Gharan 2021). LTSP is closer to its restriction on the real line, for which it can be solved in $O(n^2)$ (Bjelde et al. 2020). A key difference between LTSP and the Traveling Salesperson Problem resides in the objective function, as LTSP aims at minimizing the average service time. This objective is captured by the Minimum Latency Problem (also called Traveling Repairperson Problem) for which the best known approximation ratio is 3.59 (Chaudhuri et al. 2003). This problem is already strongly NP-hard on trees (Sitters 2002), although it admits a PTAS (Sitters 2021), but can be solved polynomially on the line if there are no deadlines (Afrati et al. 1986). Keeping the average service time objective function but adding delays at every visited vertex leads to a more general definition of the Traveling Repairperson Problem. This problem is strongly NP-hard on the line when deadlines or release times are involved (Bock 2015) but its complexity when requests can be served at any time is still unknown.

A different kind of related problems has been studied under the name of Dial-A-Ride. Here, requests are composed of a source and a destination and the goal is to move vehicles to transport all requests from their source to their destination. Several variants of the problem exist, even restricted to the offline setting, depending on the presence of release times or the number and capacities of the vehicles, see (de Paepe et al. 2004). The Dial-A-Ride problem can be seen as a generalization of LTSP but is often studied with the objective of minimizing the total service time. A simpler variant, close to our problem, considers a single vehicle able to transport one request at a time without being able to drop it before the destination, and is shown to be polynomially solvable (Atallah and Kosaraju 1988) when minimizing the total service time. A formulation aiming at minimizing the average service time has been shown to be NP-hard, relying on request irregularities (overlapping trips in different directions) (de Paepe et al. 2004, Theorem 7) which cannot happen in LTSP where requests are unidirectional and files are disjoint.

We did not cover all the work done on the online version of these problems, when future requests are unknown, but we refer the reader to (Bjelde et al. 2020) for an overview of such results.

[1]We discuss the connections with a concurrent work (Cardonha, Ciré, and Real 2021) in the Appendix

The literature on tape scheduling is rather scarce although the role of tape libraries is far from negligible in modern computing centers. Contrarily to this paper, most studies consider a more complex tape geometry, usually a serpentine. Hillyer and Silberschatz (1996) focus on low-level hardware information (*key points*) to evaluate several heuristics. Sandsta and Midtstraum (1999) propose a low-cost function to approximate the seeking time between two points of the tape. More and Choudhary (2000) design algorithms to schedule the mounts of different tapes in a library. Melia (2018) evaluates the seek times between any two points of a recent tape, data which is used as input in a few heuristics to compare their performance. Software designed to optimize tape usage appear to often sort read requests based on their tape position (Schaeffer and Casanova 2011; Zhang et al. 2006). A common point to these studies is that the focus has mostly been on cost modeling due to the two-dimensional nature of the tape and low-level hardware aspects, but publicly released scheduling algorithms are often greedy ones. A proprietary solution used by some tape libraries, named Recommended Access Order (RAO), exploits such two-dimensional tape information but its underlying algorithm is not available (IBM 2019, Section 4.27).

3 Model and Problem Descriptions

We consider a linear tape of length m, divided successively in n_f disjoint files $(f_1, \dots, f_{n_f})$ of integer size $s(f_i)$. Let $\ell(f_i)$ be the *length* between the left of the tape and the left of the file f_i and $r(f_i) = \ell(f_i) + s(f_i)$. We say that $f_i < f_j$ if file f_i is located on the left of f_j, *i.e.,* $\ell(f_i) < \ell(f_j)$. We are given a set of n requests on n_{req} files among the n_f files of the tape, with possible duplicates, where each request is a file. Let $x(f_i)$ be the number of requests allocated to file f_i.

At the beginning, the reading head is positioned on the right of the tape. A request is fulfilled when its file has been traversed from the left to the right by the reading head. We assume the reading head moves at constant speed (the tape is actually moving and the head is fixed, but switching roles helps the exposure), a time unit being necessary to traverse a file chunk of size 1 in either direction. We also consider a time penalty U for each U-turn performed by the head.

The main limitation of this model concerns the track geometry. Modern tapes are not constituted of a single linear track, and being aware of their serpentine geometry is essential to optimize the reading sequence and seeking costs. However, this simpler model is able to emulate accurately local considerations when files written in the same period are located in a single track. It is also fundamental to deeply understand the complexity of such a model knowing that the serpentine model is much closely related to NP-hard problems such as the Traveling Salesperson Problem.

The assumption of the tape moving at a constant speed in front of the reading head is obviously inaccurate due to acceleration and deceleration inherent to mechanical devices. However, the cruise speed is typically reached fast enough so this approximation is satisfactory apart from U-turns. The nominal U-turn penalty used to take into account these slowdowns therefore improves the model accuracy.

Other limitations of the model such as the undifferentiated reading speed or the forced starting position of the head are discussed as extensions in the supplementary material.

The objective is to provide a *schedule*, *i.e.,* a trajectory of the reading head on the tape, that serves all requests and minimizes the sum of service times of requests, *i.e.,* the sum of the times needed before each request is satisfied. Note that we formally define the objective as minimizing the sum, but it is more intuitive in terms of a quality of service to speak about the average service time, an objective which is completely equivalent.

A simple lower bound *VirtualLB* on the optimal solution is achieved by using n virtual heads serving each request optimally, *i.e.,* each reading head moves directly to the left of its assigned file then reads it.

$$\textit{VirtualLB} = \sum_f x(f) \cdot (m - \ell(f) + s(f) + U).$$

Minimizing the average service time is one of the most classical scheduling objective functions with minimizing the maximal service time. The latter has been the main focus of studies on the serpentine model as it minimizes the time spent using the tape which decreases wear and delay of other tapes reads. However, in the linear tape model, minimizing the maximal service time is trivial while minimizing the average service time leads to more fairness among users. This is especially true in a case of low tape usage in which tapes are rarely waiting to be mounted.

Note that the input consists of a list of n requests rather than the set of n_{req} requested files associated with their multiplicity. The motivation comes from practice, where a set of read requests has to be satisfied. Hence, polynomial time algorithms are allowed to be polynomial in n and not only $\log n$. It is natural to study first this variant of the problem, as so-called high-multiplicity problems are notoriously much harder to solve (Gabay 2014).

4 Algorithm

This section presents the main contribution of this paper, the **DP** algorithm solving LTSP in time $O(n_{req}^3 \cdot n)$. Before describing **DP**, we start with giving useful definitions, preliminary remarks, and brief descriptions of existing solutions. The complete correctness proof of **DP** is deferred to the supplementary material due to the lack of space. We then also present the **LOGDP** variant algorithm, which limits the search space of **DP** to provide a suboptimal solution with a smaller time complexity of $O(n_{req} \cdot n \cdot \log^2 n_{req})$.

4.1 Preliminaries

In this section, we study the structure of optimal solutions to provide a simple description of such schedules.

In any optimal solution, the reading head will move to the leftmost request, then to the rightmost still unread request. Before reaching the leftmost request, the head may move back and forth in possibly intricate patterns to read relevant files first. We say that the solution includes the *detour* (a, b), with a and b being two requested files such that

$a \leq b$, if the head goes directly to $r(b)$ then back to $\ell(a)$ after first attaining $\ell(a)$. As shown previously (Cardonha and Real 2016) and later stated formally in our setting (see Lemma 1), there always exists an optimal solution which can be described only via a set of detours. Furthermore, a detour can be totally surrounded by a later one (*i.e.,* (a_1, b_1) and (a_2, b_2) with $a_1 < a_2 < b_2 < b_1$) but otherwise two detours cannot intersect each other (*i.e.,* (a_1, b_1) and (a_2, b_2) with $a_1 \leq a_2 \leq b_1 \leq b_2$).

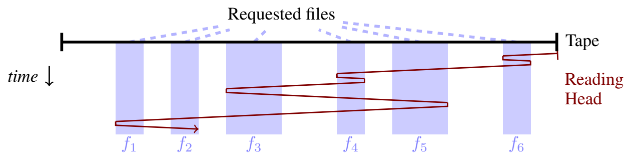

Figure 1: Example of schedule for reading six files described by the $[(f_6, f_6), (f_4, f_4), (f_3, f_5)]$ detour list. Note the delays caused by U-turn penalties.

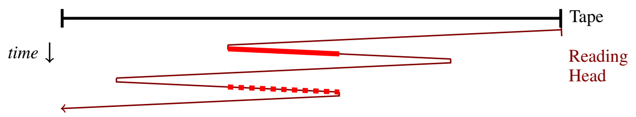

Figure 2: Example of non-optimal schedule. In the second detour, the movement in thick dotted lines is useless as these files have already been read earlier (thick solid line).

Figure 1 illustrates a possible solution while Figure 2 shows detours overlapping in a suboptimal manner.

We denote this property on the set of detours in any optimal solution as being *strictly laminar*, following a definition of *laminar* used in the scheduling literature, see for instance (Chen, Megow, and Schewior 2018). We consider that all solutions contain the detour (f_{n_1}, f_{n_f}), which reads all skipped files, even if the last movements may not count towards the objective as the rightmost requests may have already been served.

An unread file at the right of the current reading head position is called *skipped*. It will be read later when the head moves back to the right, possibly after the head read the leftmost file. For instance, on Figure 1, when f_4 is first reached by the head, f_5 is skipped, but when the head first reaches f_2, no file is skipped.

4.2 Existing algorithms

One of the simplest algorithm would be to make no detour. The head simply moves to the leftmost requested file and then reads all files left-to-right. Despite minimizing the makespan, it can be arbitrarily far from the optimal solution in our model (Cardonha and Real 2016). We refer to this algorithm as **NoDetour**.

The opposite strategy would be to perform a detour on each requested file. This algorithm, named **GS** for **G**reedy **S**cheduling, has been proved to be a 3-approximation without U-turn penalties (Cardonha and Real 2016). But of course harsh penalties can arbitrarily degrade its guarantees.

To improve the basic solution offered by **GS**, the **FGS** algorithm (Cardonha and Real 2018) detects detrimental detours in multiple evaluation passes and **F**ilters them out.

As **FGS** does not benefit from multi-file detours, the same authors designed the **NFGS** algorithm, allowing **N**on-atomic detours. In essence, for each pair of files $a < b$ starting from the left, it tests whether it would be beneficial to add the detour (a, b), after removing the detour starting from a if it existed. Despite its relatively large time complexity, **NFGS** remains greedy in nature, definitely sealing any detour that seems beneficial. A variant exploring only detours spanning over $O(\log n_{req})$ requested files, **LogNFGS**, has been proposed to trade search space for running time.

Note that the **FGS**, **NFGS**, and **LogNFGS** algorithms can be adapted to take into account the U-turn penalty in their decisions, although losing their approximation factor of 3 which was inherited from **GS**. We provide a description in the supplementary material for completeness.

The structure of existing solutions, relying on greedy evaluation passes, illustrates the difficulty of the problem. The decision of making a detour or not depends on what happens before (a detour increases the delay on skipped files) and after (subsequent detours will increase the delay on files that have been skipped). Detours can also be intricate, as shown by Figure 1. It thus seems hardly possible to take correct decisions on detours when each decision may influence the others. Consequently, Cardonha and Real (2018) only considered a very restricted model (identical file sizes and a single request per file) in which the exact solution is simple but did not otherwise get any algorithm with an approximation ratio below 3.

4.3 Algorithm

Here, we describe the **DP** dynamic programming algorithm. It uses carefully selected memoization to store the cost of specific solutions used to build an optimal schedule.

The dynamic program cells have a number and three parameters: two requested files a and b and a number $n_{skip} < n$. The objective for each cell is to compute the best possible strategy for the reading head between $r(b)$ and $\ell(a)$ knowing that:

1. there is a detour (a, f) for some file $f \geq b$,
2. there is no detour (f_1, f_2) for any files f_1, f_2 satisfying $a < f_1 < b < f_2$,
3. when the reading head reaches $r(b)$, exactly n_{skip} files have been skipped.

The content of the cell describes the impact on the total cost of the movement made by the reading head between the first time it reaches $r(b)$ and the first time it reaches $r(b)$ after having read a. In other words, it equals the sum of the lengths for all requests on any file f of the "unnecessary" paths traversed by the head in this time interval and before serving the file f. Unnecessary means that we do not count the cost that would also be incurred to *VirtualLB* on a file f between a and b, as it is inevitable and this simplifies the formulas. The U-turn penalty on a is therefore not counted as *VirtualLB* would also have one U-turn penalty, but other U-turn penalties in this interval are counted.

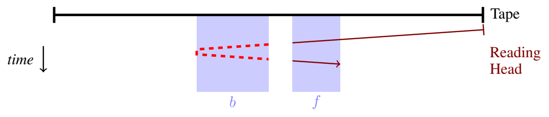

Figure 3: Cost incurred by a detour over file b to a skipped file f. The solid line represents the shortest path to serve f. The red dotted line represents the delay incurred by this detour to the service time of f. Other detours are not illustrated here. Subsequent figures follow the same logic.

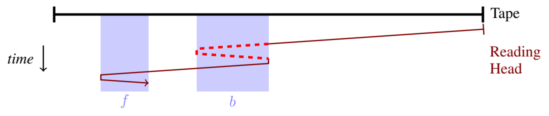

Figure 4: Cost incurred by the detour over b to a left file f.

We define $n_\ell(b)$ as the number of requests on files located on the left of b, excluding b, and let $\mathit{left}(b)$ be the closest requested file located to the left of b.

The value of cell $T[a, b, n_{skip}]$ is then defined as follows:

- If $b = a$, then there is a detour from $\ell(b)$ to at least $r(b)$ so we delay all pending requests by $2s(b)$, and incur no additional cost to b, see Figures 3 and 4. Therefore,
$$T[b, b, n_{skip}] = 2 \cdot s(b) \cdot (n_{skip} + n_\ell(b)).$$
- Otherwise, let $F_{a,b}$ be the set of requested files located between a and b excluding a. There are several possibilities to consider to determine the value of the cell: either b is skipped (it will be read with the detour starting from a), or read sooner than by the detour starting from a. In the latter case, it is read on a detour ending on b as there is no detour going to the right of b starting righter than a. This detour can start from any file in $F_{a,b}$. Then, we have:

$$\begin{aligned} \mathit{skip}(a, b, n_{skip}) &:= T[a, \mathit{left}(b), n_{skip} + x(b)] \\ &+ 2 \cdot (r(b) - r(\mathit{left}(b))) \cdot (n_{skip} + n_\ell(a)) \\ &+ 2 \cdot (\ell(b) - r(\mathit{left}(b))) \cdot x(b) \end{aligned}$$

$$\begin{aligned} \mathit{detour}_c(a, b, n_{skip}) &:= T[a, \mathit{left}(c), n_{skip}] + T[c, b, n_{skip}] \\ &+ 2 \cdot (r(b) - r(\mathit{left}(c))) \cdot (n_{skip} + n_\ell(a)) \\ &+ 2 \cdot U \cdot (n_{skip} + n_\ell(c)) \end{aligned}$$

$$T[a, b, n_{skip}] = \min\Big(\mathit{skip}(a, b, n_{skip})\ ; \min_{c \in F_{a,b}} \mathit{detour}_c(a, b, n_{skip}) \Big)$$

In the first case, we recurse on a smaller window skipping file b, hence increasing n_{skip}. We also account for the cost of the detour starting from a over the files between $\mathit{left}(b)$ and b for the requests that will be fulfilled later. The differences with earlier are that (1) we also have to account for the cost to traverse the unrequested files at the left of b and (2) requests between a and $\mathit{left}(b)$ are served before the head comes back to the right, hence there are $n_\ell(a)$ delayed files and not $n_\ell(b)$. See Figures 5 and 6. Finally, we account for the additional cost to serve b not covered by the recursive call: the path over the unrequested files directly at the left of b, see Figure 7.

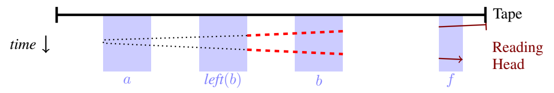

Figure 5: Impact of $\mathit{skip}(a, b, n_{skip})$ on a skipped file f. The thin dotted line represents the recursively computed impact (which may include subsequent detours), and the dashed line the impact directly accounted for.

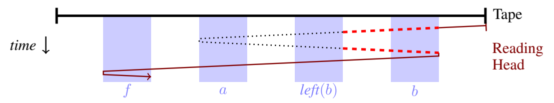

Figure 6: Impact of $\mathit{skip}(a, b, n_{skip})$ on a left file f.

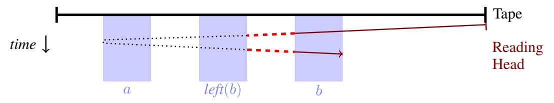

Figure 7: Impact of $\mathit{skip}(a, b, n_{skip})$ on b.

In the second case, we have a detour (c, b) for some c in $F_{a,b}$. Hence, all these files will be read when the head reaches $\mathit{left}(c)$ so we do not change n_{skip} in the recursive calls. We still need to account for the cost of the detour starting from a over the interval $(r(\mathit{left}(c)), b)$. See Figures 8 and 9. We also charge here the U-turn penalties for all requests who will be served after the head reaches a, *i.e.,* for all pending requests for which the U-turn at c is not the last one before they get served (the second U-turn penalty charged is for the U-turn occurring at b after the detour (c, b)).

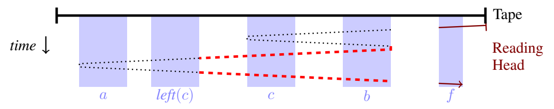

Figure 8: Impact of $\mathit{detour}_c(a, b, n_{skip})$ on a skipped file f.

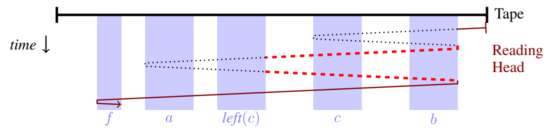

Figure 9: Impact of $\mathit{detour}_c(a, b, n_{skip})$ on a left file f.

Then, the overall solution can be computed through the call to $T[f_1, f_{n_f}, 0]$. The structure of the recursive calls minimizing this value leads to the detours taken by the underlying optimal solution.

4.4 Proof sketch of the algorithm

First, we need a structural result to guarantee that the restriction to strictly laminar detours preserves the optimal solution. A similar result has been established in (Cardonha and Real 2016). We state it here and prove it in the supplemental material for self-consistency and a more precise result.

Lemma 1. *There exists an optimal solution composed only of strictly laminar detours.*

We also refer the reader to the supplementary material for the full proof of **DP**. It relies on an induction involving several case distinctions ensuring every cost is counted once, requiring technical care to define what to count at each step.

Theorem 1. DP *solves optimally* LTSP *in time* $O(n_{req}^3 \cdot n)$.

Proof sketch. The complexity follows from the dynamic programming definition: there are $O(n_{req}^2 \cdot n)$ cells which are each computed in time $O(n_{req})$.

We show for all a, b, n_{skip} by induction on $b - a$ that the computation of cell $T[a, b, n_{skip}]$ is correct. Specifically, our induction hypothesis considers any best solution $S_{a,b,n_{skip}}$ of the problem given three additional constraints: (1) there is a detour starting from a and going to b or a righter file; (2) there is no detour starting between $r(a)$ and $\ell(b)$ and going to a file righter than b; and (3) when the reading head first reaches $r(b)$, exactly n_{skip} files have been skipped. Let t_1 be the time when the reading head first reaches $r(b)$ and t_2 be the first time the reading head reaches $r(b)$ (before performing a potential U-turn) after having read a in $S_{a,b,n_{skip}}$. For each file f, let $t(f)$ the time when it is served in $S_{a,b,n_{skip}}$. For each file $f \leq b$, let $VirtOPT_b(f) = r(b) - \ell(f) + s(f) + U$ be the minimum cost to serve f by a virtual head starting at $r(b)$ and $VirtOPT_b(f) = 0$ for $b > f$. See Figure 10.

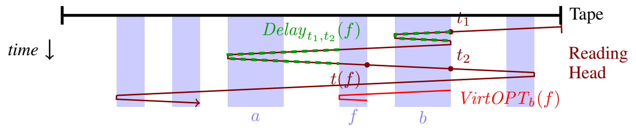

Figure 10: Illustration of t_1, t_2, $t(f)$, $VirtOPT_b(f)$ and $Delay_{t_1,t_2}(f)$ for $f \leq b$.

The hypothesis is that cell $T[a, b, n_{skip}]$ is equal to the sum for all files f of the impact $Delay_{t_1,t_2}(f)$ of what happens between t_1 and t_2 in $S_{a,b,n_{skip}}$ on the service time of f, with a basis corresponding to *VirtualLB*, *i.e.,*

$$T[a, b, n_{skip}] = \sum_f x(f) \cdot Delay_{t_1,t_2}(f) \tag{1}$$

with :

$$\begin{aligned} Delay_{t_1,t_2}(f) &:= 0 && \text{if } t(f) \leq t_1 \\ Delay_{t_1,t_2}(f) &:= t_2 - t_1 - U && \text{if } t(f) > t_2 \\ Delay_{t_1,t_2}(f) &:= t(f) - t_1 - VirtOPT_b(f) && \text{otherwise.} \end{aligned}$$

Intuitively, for files served after t_2, the reading head comes back at the place it had in t_1 at time t_2, with the opposite orientation. The delay is however not equal to $t_2 - t_1$ because we should not count the U-turn penalty here if a skipped file on the right of b is read within the same detour starting on a. Therefore, the delay equals $t_2 - t_1 - U$. Counting the cost based on *VirtualLB* allowed to simplify the computations, but in this definition it leads to a less intuitive value of the delay. For files served between t_1 and t_2, the file is served at $t(f)$ and we subtract $VirtOPT_b(f)$ to obtain the additional cost on top of the virtual lower bound.

We now show by induction on $b - a$ that Equation (1) is correct. First, consider $T[b, b, n_{skip}]$ for any b, n_{skip}. There are four types of files to consider.

- $f = b$: we have $t(f) = t_1 + 2s(b) + U$ and $VirtOPT_b(f) = 2s(b) + U$ so $Delay_{t_1,t_2}(f) = 0$,
- $f > b$ and is not skipped: we have $t(f) \leq t_1$ so $Delay_{t_1,t_2}(f) = 0$,
- $f > b$ and is skipped: we have $t(f) > t_2$ so $Delay_{t_1,t_2}(f) = 2s(b) + U - U$,
- $f < b$: we have $t(f) > t_2$ so $Delay_{t_1,t_2}(f) = 2s(b)$.

Overall, there are $n_{skip} + n_\ell(b)$ files who have a delay equal to $2s(b)$ so:

$$\begin{aligned} \sum_f x(f) \cdot Delay_{t_1,t_2}(f) &= 2 \cdot s(b) \cdot (n_{skip} + n_\ell(b)) \\ &= T[b, b, n_{skip}]. \end{aligned}$$

This completes the base case of the induction ($b - a = 0$).

Now, consider $T[a, b, n_{skip}]$ for any values of a, b and n_{skip} such that $a < b$ and assume the induction hypothesis. We want to show that:

$$T[a, b, n_{skip}] = \sum_f x(f) \cdot Delay_{t_1,t_2}(f).$$

See the supplementary material for this part of the proof. Finally, we get by induction, for all a, b, n_{skip} and $S_{a,b,n_{skip}}$:

$$T[a, b, n_{skip}] := \sum_f x(f) \cdot Delay_{t_1,t_2}(f).$$

Note that $S_{f_1,f_{n_f},0}$ is equal to the optimal solution of the problem. So, denoting by t_0 the starting time of the solution and $t_{\max}$ the time at which the reading head would reach back the right of the tape in $S_{f_1,f_{n_f},0}$ (it may stop earlier if the rightmost file is not skipped), we get that the content of the cell $T[f_1, f_{n_f}, 0]$ is equal to:

$$\begin{aligned} T[f_1, f_{n_f}, 0] &= \sum_f x(f) \cdot Delay_{t_0,t_{\max}}(f) \\ &= \sum_f x(f) \cdot (t(f) - t_0 - VirtOPT_{f_{n_f}}(f)) \\ &= cost(S_{f_1,f_{n_f},0}) - VirtualLB. \end{aligned}$$

Then, we obtain that the optimal cost is equal to $OPT = T[f_1, f_{n_f}, 0] + VirtualLB$, which completes the proof. □

4.5 Efficient heuristic

The complexity of **DP** may be prohibitive for an input containing hundreds of requested files. We address this issue by providing a lighter algorithm named **LOGDP**. It is equal to **DP** except that when computing $detour_c(a, b, n_{skip})$, c is restricted to be at most $\lambda \cdot \log n_{req}$ requested files apart from b, for a constant parameter λ. This reduces both the table dimensions and complexity to query a single cell so leads to a time complexity of $O(n_{req} \cdot n \cdot \log^2 n_{req})$. Only detours of span at most $\lambda \cdot \log n_{req}$ are then considered, and the solution returned is optimal among this class of schedules. The parameter λ can be adjusted to trade accuracy for computing time. We note that as this solution is by definition at least as good as **GS**, it is also a 3-approximation if $U = 0$.

5 Performance evaluation

In this section, we evaluate the performance, as the sum of service times of its generated sequence of detours, of our exact algorithm, **DP**, and its suboptimal version **LOGDP** with a reduced complexity on a real-world dataset. We also compare the performance of **DP** and **LOGDP** to existing algorithms (Cardonha and Real 2018) (see Section 4.2). Aiming for reproducibility, the source code[2] and dataset[3] used in this section are made publicly available online.

5.1 Evaluated algorithms

We consider two variants of **LOGDP** with the λ parameter equal to either 1 or 5, denoted by **LOGDP(1)** and **LOGDP(5)**. We adapted the **FGS**, **NFGS**, and **LOGNFGS** algorithms from (Cardonha and Real 2018) to take U-turn penalties into account. We further modified **NFGS** on three points which we believe were intended by the original authors as otherwise **FGS** can give better solutions, contradicting a claim in the paper. Details concerning our implementation can be found in the supplementary material and in the source code. All these algorithms were implemented in a single-thread Python program.

For each tape, each algorithm needs the following inputs: an ordered list of indices of the files requested on the tape; the number of requests for each requested file; the size of all files on the tape; and the cost of the U-turn penalty.

The output of an algorithm is a list of detours where a detour is a couple (a, b) which means that the head goes to the left of file a then to the right of file $b \geq a$. A value of $a = 0$ corresponds to the leftmost requested file on the tape. Then, we compute the sum of service times for each file request following the sequence of detours given by each algorithm.

5.2 Inputs from production logs

The IN2P3 Computing Center, from which our dataset comes, uses tape storage for long-term projects in High Energy Physics and Astroparticles physics. Its tape library is currently composed of 48 TS1160 drives and can store up to 6,700 20TB IBM Jaguar E tapes.

The raw dataset covers two weeks of activity. It contains millions of lines of reading, writing, and update requests with their associated timestamp. We applied several filtering steps to obtain the inputs needed by the algorithms. We restricted to reading requests, and selected a set of 169 tapes of interest storing $3, 387, 669$ files. Each tape is divided into segments whose size and number depend on the tape. In a segment, files and *aggregates* of files are described by several features such as a position and a size. An aggregate is a batch of related files that can be written sequentially. A segment contains an aggregate if there is more than one file referenced in this segment. Within an aggregate, the position of a file is described a couple (position, offset) where the position corresponds to the beginning of the aggregate, thus the beginning of a segment, and the offset is the relative position of the file within the aggregate. Note that an aggregate can span across several segments. We discarded such aggregates and their associated requests to focus on aggregates lying on a single segment. Reading files inside an aggregate is not straightforward and generates a non-negligible overhead as the head is required to go to the start of the aggregate before reading a file.

Finally, we decided to consider that requesting a file within an aggregate will be treated as a request to read the whole aggregate. While this simplifies log filtering process, this assumption also corresponds to a common optimization strategy. Read aggregates are stored on disks when a file it contains is read for the first time. Then, all the subsequent accesses to files in this aggregate will avoid the large delays induced by tapes and benefit of the smaller latency of disks. Consequently, we replace all the file requests in a given aggregate by a single request for a file of the size of this aggregate. Then we associate to this file a number of requests equal to the number of requested files in that aggregate.

To summarize, the processed dataset corresponds to a total of $119, 877$ files stored on the 169 tapes. We provide more details and statistics on this dataset in the supplementary material. To the best of our knowledge, this is the first time that a realistic dataset for magnetic tape storage is made publicly avaible. In the context of the evaluation of the considered algorithms, this dataset corresponds to 169 distinct instances of LTSP to solve.

5.3 Simulation results

The evaluations presented in this section have been performed on a single server with two Intel Xeon Gold 6130 CPUs with 16 cores each. To compare the performance of the different algorithms, we use the generic *performance profile* tool (Dolan and Moré 2002). We compute the cost of each algorithm on each instance of the dataset, normalize it by the optimal (**DP**), and report an empirical cumulative distribution function. For a given algorithm and an overhead τ expressed in percentage, we compute the fraction of instances for which the algorithm has a cost at most $(1 + \tau) \cdot cost(\mathbf{DP})$, and plot these results. Therefore, the higher the curve, the better the method. For instance, for an overhead of $\tau = 10\%$, the performance profile shows how often the performance of a given algorithm lies within 10% of the optimal solution.

[2]*https://figshare.com/s/80cee4b7497d004dbc70*

[3]*https://figshare.com/s/a77d6b2687ab69416557*

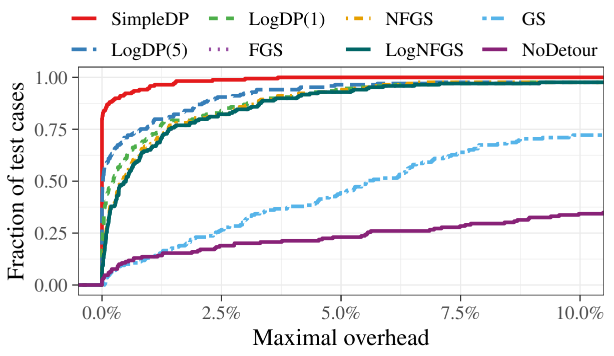

Figure 11: Performance of the algorithms, when $U = 0$.

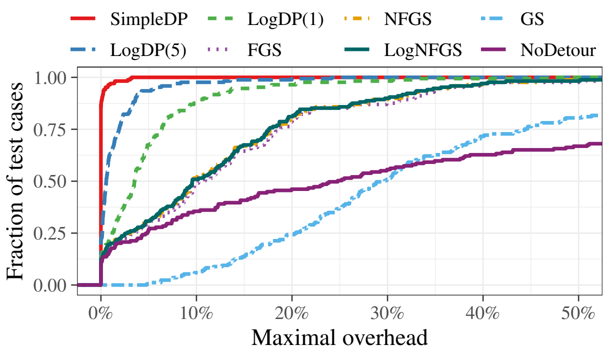

Figure 12: Performance of the different algorithms, when U is equal to the average segment size.

We evaluate the algorithms on each of the 169 instances for three different values of the U-turn penalty U: (i) no penalty (ii) a penalty equals to half of the average size of a segment in the 169 considered tapes, and (iii) a penalty equivalent to the average size of a segment. While we have not yet modeled seeking and reading speeds of the head, such penalties whose values are extracted from features of the input instances are useful to evaluate the impact of increasing U on the performance of the algorithms.

Algorithms Performance Figure 11 shows the performance profiles of the algorithms with $U = 0$. As expected, **GS** and **NoDetour** show poor performance, with an overhead of more than 10% for **NoDetour** over 60% of the instances. The **FGS**, **NFGS**, and **LogNFGS** heuristics exhibit very similar performance, with an overhead of less than 2.5% over 80% of the test cases. Both variants of **LogDP** heuristic slightly outperform the other heuristics. As expected, the higher λ, the closer to optimal the solution is. **NFGS** is better than **LogDP**(1) on 11% on the instances, and worse in 85%. It performs better when a single long detour is largely beneficial, and out of reach of **LogDP**.

Figure 12 illustrates the algorithms performance with a U-turn penalty equal to the average size of a segment. We see that U increases the discrepancy between the **FGS**-like heuristics and **LogDP**. Here, these heuristics cause at least 5% more overhead on half of the instances than **LogDP(1)**, and up to 10% more overhead than **LogDP(5)**. The suboptimal solutions of **LogDP** variants are more robust to the increase of U, with an overhead of a few percent for **LogDP(5)** when compared to **DP** for 90% of the inputs.

Time to solution The median running times for the algorithms **DP**, **LogDP**(5), **LogDP**(1), **NFGS** and **LogNFGS** are around 281, 47, 5, 0.4 and 0.1 seconds respectively. The other algorithms have insignificant running times ($<$1ms). However, our single-thread Python implementation was not designed with performance in mind. Estimations based solely on the documented maximum speed of the reading head leads to an average duration of 500s to schedule the requests on one tape of the dataset with an average service time of 80s. The observed gains thus have to be nuanced by the required computing times of the algorithms. It should also be noticed that the schedule computation can be done in parallel to robot operations mounting the tape, so the start of the schedule is not directly delayed by the computation time. The characteristics of the data set (a median $n > 2,600$ much larger than $n_{req} < 150$) also explain the longer running times of **LogDP** as the **FGS**-like algorithms complexity does not depend on n, see more details in the supplementary material. The λ parameter can be used to obtain a faster version of **LogDP** at the cost of lower performance. On large inputs (*i.e.,* list of requested files greater than 100), the cost of **DP** becomes prohibitive in a production context, making **LogDP** variants good replacement candidates.

6 Conclusion

In this article we studied the Linear Tape Scheduling Problem, aiming at minimizing the average service time for read requests on a linear magnetic tape. We proposed an exact polynomial-time dynamic programming algorithm, solving this problem whose complexity was open until now. Then, we derived a low-cost suboptimal algorithm, whose performance outperforms existing heuristics on a realistic dataset extracted from the tape library logs of a major computing center, a dataset we make publicly available.

This dataset could also be used for related problems such as k-server on the line for which few relevant datasets are available (Lindermayr, Megow, and Simon 2021). The remaining question on the theoretical side of LTSP resides in the dependency in n of an optimal algorithm. The obvious generalization of the problem would be to consider the two-dimensional tape geometry, but we expect that such a model would quickly become intractable. We also discuss in the supplemental material how **DP** can be adapted to handle two minor extensions: arbitrary starting position of the head and a different reading speed.

Acknowledgments

We thank Pierre-Emmanuel Brinette for fruitful discussions. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

Afrati, F.; Cosmadakis, S.; Papadimitriou, C. H.; Papageorgiou, G.; and Papakostantinou, N. 1986. The Complexity of the Travelling Repairman Problem. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, 20(1): 79–87.

Atallah, M. J.; and Kosaraju, S. R. 1988. Efficient Solutions to Some Transportation Problems with Applications to Minimizing Robot Arm Travel. *SIAM Journal on Computing*, 17(5): 849–869.

Bjelde, A.; Hackfeld, J.; Disser, Y.; Hansknecht, C.; Lipmann, M.; Meißner, J.; Schlöter, M.; Schewior, K.; and Stougie, L. 2020. Tight Bounds for Online TSP on the Line. *ACM Transactions on Algorithms*, 17(1): 1–58.

Bock, S. 2015. Solving the Traveling Repairman Problem on a Line with General Processing Times and Deadlines. *European Journal of Operational Research*, 244(3): 690–703.

Cano, E.; Bahyl, V.; Caffy, C.; Cancio, G.; Davis, M.; Keeble, O.; Kotlyar, V.; Leduc, J.; and Murray, S. 2021. CERN Tape Archive: a distributed, reliable and scalable scheduling system. In *EPJ Web of Conferences*, volume 251, 02037. EDP Sciences.

Cardonha, C.; and Real, L. C. V. 2016. Online Algorithms for the Linear Tape Scheduling Problem. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*. London, UK.

Cardonha, C.; and Real, L. C. V. 2018. Theoretical and Practical Aspects of the Linear Tape Scheduling Problem. *CoRR*, abs/1810.09005v1.

Cardonha, C. H.; Ciré, A. A.; and Real, L. C. V. 2021. On Exact and Approximate Policies for Linear Tape Scheduling in Data Centers. *CoRR*, abs/2112.07018.

Chaudhuri, K.; Godfrey, B.; Rao, S.; and Talwar, K. 2003. Paths, trees, and minimum latency tours. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, 36–45.

Chen, L.; Megow, N.; and Schewior, K. 2018. An O(m)-Competitive Algorithm for Online Machine Minimization. *SIAM Journal on Computing*, 47(6): 2057–2077.

Davis, M. C.; Bahyl, V.; Cancio, G.; Cano, E.; Leduc, J.; and Murray, S. 2019. CERN Tape Archive – from Development to Production Deployment. In *Proceedings of the 23rd International Conference on Computing in High Energy and Nuclear Physics*, volume 214 of *EPJ Web of Conferences*, 04015. EDP Sciences.

de Paepe, W. E.; Lenstra, J. K.; Sgall, J.; Sitters, R. A.; and Stougie, L. 2004. Computer-Aided complexity Classification of Dial-a-Ride Problems. *INFORMS Journal on Computing*, 16(2): 120–132.

Dolan, D. E.; and Moré, J. J. 2002. Benchmarking Optimization Software with Performance Profiles. *Mathematical Programming*, 91(2): 201–213.

Gabay, M. 2014. *High-multiplicity Scheduling and Packing Problems : Theory and Applications*. Theses, Université de Grenoble.

Hillyer, B. K.; and Silberschatz, A. 1996. On the Modeling and Performance Characteristics of a Serpentine Tape Drive. *ACM SIGMETRICS Performance Evaluation Review*, 24(1): 170–179.

IBM. 2019. *IBM System Storage Tape Drive 3592 SCSI Reference*. IBM.

Karlin, A. R.; Klein, N.; and Gharan, S. O. 2021. A (slightly) improved approximation algorithm for metric TSP. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 32–45.

Lawler, E. L.; Lenstra, J. K.; Kan, A. H. G. R.; and Shmoys, D. B. 1985. The traveling salesman problem: a guided tour of combinatorial optimization. *Wiley-Interscience Series in Discrete Mathematics*.

Lindermayr, A.; Megow, N.; and Simon, B. 2021. Double Coverage with Machine-Learned Advice. *arXiv preprint arXiv:2103.01640*.

Melia, G. C. 2018. LTO experiences at CERN. https://indico.cern.ch/event/730908/contributions/3153156/. Accessed: 2022-03-26.

More, S.; and Choudhary, A. 2000. Scheduling queries for tape-resident data. In *European Conference on Parallel Processing*, 1292–1301. Springer.

Mäsker, M.; Nagel, L.; Süß, T.; Brinkmann, A.; and Sorth, L. 2016. Simulation and Performance Analysis of the ECMWF Tape Library System. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 252–263. Salt Lake city, UT.

Reine, D.; and Kahn, M. 2015. Continuing the Search for the Right Mix of Long-term Storage Infrastructure – a TCO Analysis of Disk and Tape Solutions. Technical Report TCG2015006, The Clipper Group, Inc. [Online, Dec. 2021]www.clipper.com/research/TCG2015006.pdf.

Sandsta, O.; and Midtstraum, R. 1999. Improving the Access Time Performance of Serpentine Tape Drive. In *Proceedings 15th International Conference on Data Engineering*, 542–551. Sydney, Australia: IEEE.

Schaeffer, J.; and Casanova, A. G. 2011. TReqS: The Tape REQuest Scheduler. In *Journal of Physics: Conference Series*, volume 331, 042040. IOP Publishing.

Sitters, R. 2002. The Minimum Latency Problem is NP-hard for Weighted Trees. In *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization*, 230–239. Cambridge, MA: Springer.

Sitters, R. 2021. Polynomial Time Approximation Schemes for the Traveling Repairman and Other Minimum Latency Problems. *SIAM Journal on Computing*, 50(5): 1580–1602.

Zhang, X.; Du, D.; Hughes, J.; Kavuri, R.; and StorageTek, S. 2006. Hptfs: A high performance tape file system. In *Proceedings of 14th NASA Goddard/23rd IEEE conference on Mass Storage System and Technologies*. Citeseer.