An Exact Algorithm for the Linear Tape Scheduling Problem

Valentin Honoré, Bertrand Simon, Frédéric Suter

IN2P3 Computing Center – CNRS, Villeurbanne – Lyon, France

JCAD – October 2022
Tape usage today

≈ 20TB on 1000s × 1km read at 10m/s – 100s MB/s
(or 11 football fields)

Primordial for HTC (High Throughput Computing)

e.g., CERN, CCIN2P3, ECMWF (100s PB)
also: media companies, cloud archive...

Impressive technology improvements
density: + 30% / year (vs HDD: + 8%)

High latency (mount, load, position → few mn)
Adapted for Write Once Read Many
Overview of a tape

≈ 1km  
≈ 0.5 inch or 1.3cm

> 200 wraps (linear serpentine)

wrap = dozens of tracks read / written simultaneously by parallel heads
Overview of our tape model

≈ 0.5 inch or 1.3 cm

≈ 1km
Linear Tape Scheduling Problem

Assumptions:

- Files are read left-to-right
- Start on the right
- Constant speed

Input:

- Tape of \( n_f \) consecutive files
- \( n \) file requests (44 here)
- \( n_{req} \) distinct files requested (6)

Objective: average service time

Motivation: lack of fundamental theoretical results, models local files
**Assumptions:**

- files are read left-to-right
- start on the right
- constant speed
- [new] U-turn penalty $U$

**Input:**

- tape of $n_f$ consecutive files
- $n$ file requests (44 here)
- $n_{req}$ distinct files requested (6)

**Objective:** average service time

Motivation: lack of fundamental theoretical results, models local files
Related to the Linear Tape Scheduling Problem

**Travelling Salesperson Problem (TSP)**
- super-famous NP-hard problem
- recent \((1.5 - 10^{-36})\) - approximation \([\text{KarlinKleinGaran'21}]\)
- 😞 minimizes makespan, trivial on the real line

**Minimum Latency Problem / TRP (Repair) - variant**
- 😊 minimize average service time \(\in \mathbb{P}\) on the real line
- delays to repair a node: complexity open

**Dial-a-ride variant on the real line**
- \(\approx\) LTSP but with overlapping files in both directions
  \(\rightarrow\) NP-hard

Tapes except LTSP: 2 specific experimental papers in the 90’s
Structural results

Any optimal solution

▶ after reaching $\ell(f_1)$, go straight to the rightmost unread request
▶ can be described by a set of **detours** done before

Definition (Detours)

A solution includes the **detour** $(a,b)$ with $a \leq b$ if:

▶ the 1st time the head reaches $\ell(a)$, go straight to $r(b)$, back to $\ell(a)$

Requested files

| $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ |
|-------|-------|-------|-------|-------|-------|

detours: $[(f_6, f_6), (f_4, f_4), (f_3, f_5)]$
Any optimal solution

- after reaching $\ell(f_1)$, go straight to the rightmost unread request
- can be described by a set of detours done before

**Definition (Detours)**

A solution includes the detour $(a,b)$ with $a \leq b$ if:

- the 1st time the head reaches $\ell(a)$, go straight to $r(b)$, back to $\ell(a)$

**Lemma**: detours never partially overlap (strictly laminar)
Any optimal solution

- after reaching $\ell(f_1)$, go straight to the rightmost unread request
- can be described by a set of detours done before

**Definition (Detours)**

A solution includes the detour $(a,b)$ with $a \leq b$ if:

- the 1st time the head reaches $\ell(a)$, go straight to $r(b)$, back to $\ell(a)$

Lemma: detours never partially overlap (*strictly laminar*)
**Naive algorithms**

**NoDetour**: go to the leftmost request, then to the rightmost

can be arbitrarily bad (place urgent requests on the right)

![Diagram of NoDetour algorithm]

**GS (Greedy Schedule)**: do all atomic detours, i.e., \( \{(f_i, f_i)\}_{\forall i} \)

Lemma [CardonhaReal’16]: **GS** is a 3-approximation if \( U = 0 \)

Proof: does \( \leq 3 \) times the optimal distance before reading each request

![Diagram of Greedy Schedule algorithm]
Heuristic improvements

**FGS** (Filtered): remove detrimental atomic detours in $O(n_{req}^2)$

**NFGS** (Non-atomic): greedily add long detours if currently beneficial. Make one pass from left to right. Complexity in $O(n_{req}^3)$. 

FGS (Filtered): remove detrimental atomic detours in $O(n_{req}^2)$

NFGS (Non-atomic): greedily add long detours if currently beneficial. Make one pass from left to right. Complexity in $O(n_{req}^3)$
Dynamic Program DP: overview

Each cell: three parameters $T[a, b, n_{\text{skip}}]$

- compute the best strategy from $r(b)$ to $\ell(a)$ assuming:
  1. there is a detour $(a, f)$ for some $f \geq b$,
  2. there is no detour $(f_1, f_2)$ such that $a < f_1 < b < f_2$,
  3. when reaching $r(b)$, exactly $n_{\text{skip}}$ requests have been skipped.

$\Rightarrow$ value $\approx$ cost contribution from ‘first $r(b)$’ to ‘$r(b)$ after reading $a$’

Subtleties: $\forall$ request on $f$, do not count the cost $m \rightarrow \ell(f) \rightarrow r(f)$ if $f$ is read after $b$, remove one $U$ (counted before)
Dynamic Program DP: overview

Each cell: three parameters $T[a, b, n]$.

$\Rightarrow$ value $\approx$ cost contribution from 'first $r(b)$' to 'after reading $a$'.

Subtleties: $\forall$ request on $f$, do not count the cost $m \rightarrow \ell(f) \rightarrow r(f)$ if $f$ is read after $b$, remove one $U$ (counted before).
More dynamic programs

**Theorem**

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

**LogDP**($\lambda$): **DP** restricted to detours spanning $\lambda \log n_{req}$ requested files

Reduced complexity in $O(\lambda^2 \cdot n_{req} \cdot n \cdot \log^2(n_{req}))$, tested with $\lambda \in \{1, 5\}$

**SimpleDP**: **DP** forbidding intertwined (i.e., overlapping) detours

Reduced complexity in $O(n \cdot n_{req}^2)$, better theoretical guarantees
Simulations: overview

Dataset: 2 weeks at CC-IN2P3

- 169 tapes, > 3M files
- focus on reading operations
- filtering steps, data processing (e.g., merge reads on aggregates)
- median data: 150 files requested, 3k requests, 50% file size variation

Code + dataset (with statistical descriptions) available online

Experimental methodology

- vary $U$
- median timings (seconds, on a compiled Python program):

|       | FGS | NFGS | LogDP(1) | SimpleDP | LogDP(5) | DP |
|-------|-----|------|----------|----------|----------|----|
| Median | < 0.1 | 1 | 2 | 3 | 7 | 30 |
Simulation results, $U = 0$

Performance profile: best is top-left (most instances with low overhead vs OPT)
Simulation results, $U = \text{file}$

Performance profile: best is top-left (most instances with low overhead vs $\text{OPT}$)
Conclusion

General: tapes are past & future

▶ tapes stay primordial in some fields but neglected by CS research
▶ fundamental problems are still open

On LTSP

▶ high-multiplicity variant remains open
▶ huge gap between theoretically studied models and practical heuristics

Perspectives on other tape-related topics

▶ multi-tape requests: optimize waiting queues
▶ optimize tape / disk storage ratio