Scheduling with Predictions

Alberto Marchetti Spaccamela
Sapienza U. of Rome

Learning-Augmented Algorithms

Beyond worst case: a line of research in the algorithm community

- Access to (machine learning) predictions on problem parameters
- No assumption on the quality of the prediction

Desired properties

Consistency: better than worst case if prediction errors are small

Robustness: bounded worst-case for arbitrary predictions

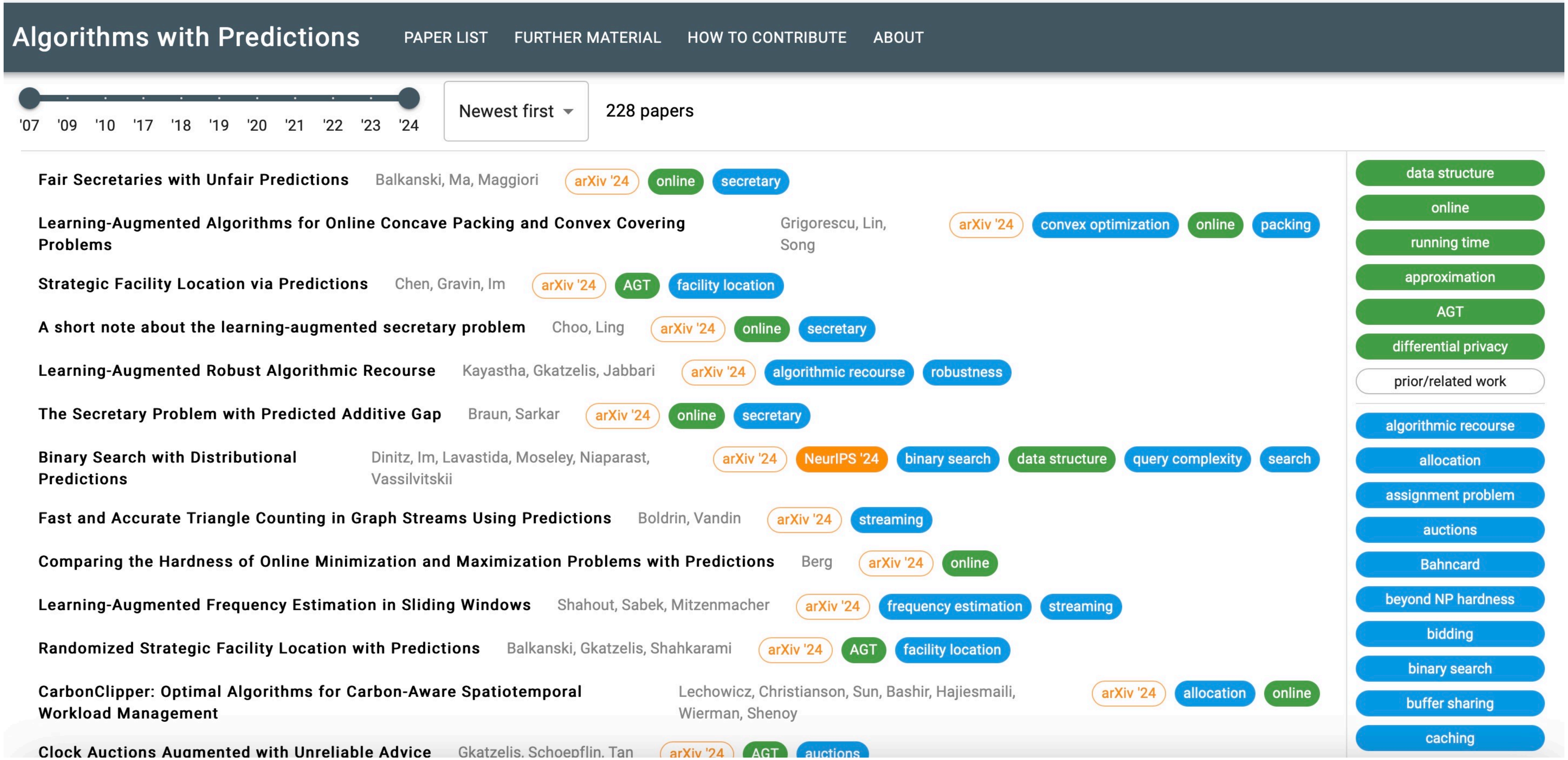
Smoothness: graceful degradation with the error

Learnability : good values of the predicted quantity can be learnt

- Line of research initiated by [Lykouris, Vassilvitskii, ICML 2018], [Kraska, Beutel, Chi, Dean, Polyzotis, SIGMOD 2018]

A vibrant area

https://algorithms-with-predictions.github.io/ [Lindermayr, Megow]



SUMMARY

1. Motivation and definitions

2. Minimizing sum of completion times

3. Energy minimization via speed scaling

Learning-Augmented Algorithms: search an array

BINARY SEARCH

Search items in an ordered array with n items:

Given an ordered array

- search 23

Binary search

- cost $O(\log_2 n)$

Question: is there an algorithm A such that given a prediction of the index of the searched item A has constant cost if the prediction is good and $O(\log_2 n)$ cost if the prediction is bad?

Finding a book in the library

Finding a book in the library

Books are usually ordered in alphabetical ordering of authors

Search
Author: **Al-Khwārizmī**
Book **Al-Jabr** *(820 CE)*
- the term *algebra* comes from this book

Will you use binary search?

Learning-Augmented Algorithms: search an array

Search an ordered array

- Small items are in first positions; large items in last positions
- Prediction: given the searched value guess its position

Question: is there an algorithm A such that

- A has constant cost if the prediction is good and same cost of binary search O(log n) cost if the prediction is bad?
- Can we smoothely bound the cost of A between constant and logarithm as a function of the quality of the prediction?

Learning-Augmented Algorithms: search an array

Search an ordered array

- Small items are in first positions; large items in last positions
- Prediction: given the searched value guess its position

Question: is there an algorithm A such that

- A has constant cost if the prediction is good and same cost of binary search O(log n) cost if the prediction is bad? YES
- Can we smoothely bound the cost of A between constant and logarithm as a function of the quality of the prediction? YES

Learning-Augmented Algorithm: search an array

Learning augmented algorithm [M. Mitzenmacher, S. Vassilvitskii 2022]

- Given item q and a predictor h, let h(q) be the predicted position

A simple approach

1. first probe the location A[h(q)] using the predictor
2. if q is not found there, we know whether it is smaller or larger. Suppose q is smaller than the element in A[h(q)] and the array is sorted in increasing order.
3. Probe elements at h(q) - 2, (h(q) – 2)- 4, (h(q) - 2 - 4) - 8, and so on, until an element larger than q in position p(q) is found (or the end of the array is reached).
4. Apply binary search on the interval [h(q), p(q)] that's guaranteed to contain q (if it exists).

Search an array: search 2, predicted position 9

Algorithm

- check items at increased distance from the predicted position
- the distance doubles at each step unless one end of the array is reached
- until
  - The queried item is found
  - Or an item smaller than the queried item is tested; in this case binary search is done in a subarray

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91
2	5	8	12	16	23	38	56	72	91
2	5	8	12	16	23	38	56	72	91
2	5	8	12	16	23	38	56	72	91
2	5	8	12	16	23	38	56	72	91

2 log N +1 is the worst case cost in an array of N items

Learning-Augmented Algorithm: search an array

Learning augmented algorithm [M. Mitzenmacher, S. Vassilvitskii 2022]

- Given item q and a predictor h, let h(q) be the predicted position
- If not found proceed via doubling binary search starting from h(q)

Analysis: Let q be the item we search

- h(q) = predicted position p(q) = effective position
- error $\mu = |h(q) - p(q)|$ distance between effective and predicted pos.
- Learning augmented algorithm : $O(\log \mu)$ ⟹ robust and smooth

Results:

- **Consistent**: perfect predictions recover constant lookup times
- **Robust**: if predictions are bad, not (much) worse than usual binary search
- **Smoothness**: the complexity increase with the log of the error

SUMMARY

1. Motivation and definitions

2. Minimizing sum of completion times

3. Energy minimization via speed scaling

Minimize Sum of Completion Times

Input: set of jobs with processing requirements p_i

Objective: Minimize sum of completion times $\Sigma_i C_i$

All jobs are released at time 0 and processing times are known.

Optimal Schedule

Shortest Processing Time first (SPT) [Smith 1956]

easy extensions

- jobs released over time: Shortest Remaining Processing Time first
- weighted case: order jobs according to the ratio weight/processing time

Minimize Sum of Completion Times

Input: set of jobs with processing requirements p_i

Objective: Minimize sum of completion times $\Sigma_i C_i$

Processing times are unknown.

We cannot expect to find the optimal solution.

An online algorithm is ρ-competitive if it achieves, for any input instance, a solution of cost within a factor ρ of the optimal cost:

- $\mathrm{Alg}(I) \leq \rho \cdot \mathrm{Opt}(I)$, for any input I.

- **Round-Robin (RR)** is 2-competitive for minimizing $\Sigma_i C_i$ on a single machine, and this is best-possible. [Motwani, Phillips, Torng 1994]

Minimize Sum of Completion Times: prediction

Toy example [Mitzenmacher]

- Two types of jobs, n short jobs (lenght S) and n long jobs (length L)
- Jobs are released at time 0
- Goal: Minimize Sum of Completion Times on a uniprocessor
- SRPT: If we know the sizes, put short jobs first

  sum of completion times is

  $$\underbrace{n(n+1)\,S/2}_{\text{short jobs}} \;+\; \underbrace{n(n+1)/2\;L\;2 + n^2\,S}_{\text{long jobs}} \quad \sim \quad n^2\,S/2 + n^2\,L/2 + n^2\,S$$

Minimize Sum of Completion Times

n short jobs (lenght S) and n long jobs (length L) are released at time 0

Goal: Minimize Sum of Completion Times on a uniprocessor

If we know the sizes, put short jobs first sum of completion times (SRPT)

$$\sim \quad n^2 S/2 + n^2 L/2 + n^2 S$$

If we don't know the sizes, randomize job order:

- about half of the jobs are in the wrong place (long jobs in the first half and short jobs in the second half)

  expected sum of completion times is

  $$\sim n^2 S/2 + n^2 L/2 + n^2 (S + L)/2$$

- Difference between optimal and randomized solution: $\sim n^2 (L - S)/2$

Toy example: use of prediction

n short jobs (lenght S) and n long jobs (length L) are released at time 0

Goal: Minimize Total Waiting Time on a uniprocessor

- If we know the sizes sum of completion times is $\sim\ n^2 S/2 + n^2 S + n^2 L/2$
- Suppose we have a predictor that can predict whether jobs are short or long

  $\Rightarrow$ short predicted jobs are first (in random order)
- If jobs are misclassified with probability p (short jobs) and q (long jobs)

  expected sum of completion times

  $\sim\ n^2 S/2 + n^2 L/2 + n^2 (S + (p+q)(L-S)/2)$
- Difference between clairvoyant and algorithm using prediction

  $\sim n^2 (p+q) (L-S)/2$

Toy Example: consistency, robustness

Difference between clairvoyant and algorithm using prediction

$\sim n^2$ (p+q) (L-S)/2

- **Consistency**: Clarvoyant case: p=q=0

  $\sim n^2$ (p+q) (L-S)/2 = 0 ⇒ algorithm is optimal

- **Robustness**: Worst case: random prediction: p=q=0.5

  $\sim n^2$ (p+q) (L-S)/2 = n^2 (L-S)/2 ⇒ algorithm behaves as classical randomized algorithm

- **Smoothness**: The ratio between algorithm and optimum is bouded by

  $1 + (p+q)\,(\sqrt{L/S} - 1)/2$

Minimize flow time: Jobs are released over time

r_i release time, known processing time p_i

Objective function:

minimize weighted flow time $\Sigma_i (C_i - r_i)$ C_i is the completion time of job i

Clarvoyant (Processing times are known at release times): SRPT

- 1 machine: SRPT is optimal
- m machines : SRPT optimal competitive ratio O(min (log (n/m), log P)) [P = max processing time]

Non clairvoyant: Randomized MultiLevel Feedback (RMLF)

- 1 machine: RMLF is optimal competitive O(log n)
- m machines : RMLF is O((log n) min(log (n/m),log P))

Minimize flow time: Jobs are released over time

General processing time: we have a predictor p_i on the actual processing time a_i of job i

Suppose you use SRPT and execute predicted short jobs first:

- If prediction is smaller than real execution $\Longrightarrow$ you may execute a job whose predicted remaining processing time is 0
- If keep on executing it then you may delay too many jobs (*if the prediction for this single job is very bad but exact for all other jobs*)
- On the other side you would like to limit the number of jobs that started execution and are not completed (*if for many jobs the prediction is slightly wrong and the error is small*)

Minimize flow time with prediction

jobs are defined by r_i release time, p_i, processing time, weight w_i

Objective: minimize $\Sigma_i w_i (C_i - r_i)$, C_i is completion time of job i

Prediction: Assume to have a prediction on p_i let μ to be the error

Many papers

- 1 machine, m identical machines, m unrelated machines
- Weighted, unweighted
- Different definition of error parameter; example: if p is predicted execution and p^* is the real execution then p^* is at least p/μ and at most μp (i.e. $\mu = \max_i (p/p^*, p^*/p)$)

Many papers: [Im et al. 2018] [Purohit et al. 2018] [Wei 2020] [Azar et al. 2021][Lindermayr, Megow 2022] [Zhao et al. 2022] ….

SUMMARY

1. Motivation and definitions

2. Minimizing sum of completion times

3. Energy minimization via speed scaling

Energy minimization

Schedule with minimum energy requirement

Toy example:

- One job, deadline D, predicted execution time P, wcet W, release time 0
- Our goal is to minimize energy consumption
- Let s(t) be the speed the processors is running at time t then the required energy is

$$\int s(t)^{\alpha}\, dt \text{ for } \alpha > 0 \quad (\alpha \text{ is assumed to be} \geq 2)$$

Energy minimization: minimal example

One job: P predicted execution time P, A actual exeution. time, W wcet, D deadline

Let s(t) be the speed the processors is running at time t then the required energy is

$$\int s(t)^{\alpha}\, dt \text{ for } \alpha > 0 \quad \text{(parameter } \alpha \text{ is assumed to be} \geq 2\text{)}$$

Assume you do not trust the prediction

- To complete the job in the worst case requires speed $s = \frac{W}{D}$
- If A is the actual execution time then execution requires time $\frac{A}{s} = \frac{AD}{W}$
- the energy requirement E(A) is

$$E(A) = \left(\frac{W}{D}\right)^{\alpha} \left(\frac{AD}{W}\right) = \left(\frac{W}{D}\right)^{\alpha} A \left(\frac{D}{W}\right) = A \left(\frac{W}{D}\right)^{\alpha - 1}$$

Energy minimization: minimal example

P predicted execution time P, A actual exec. time, W wcet, D deadline

Algorithm

ALG: Trust the prediction and up to time t_v run at speed $s' = \frac{P}{t_v} < s = \frac{W}{D}$

1. If $A \leq P$ then the job completes by time t_v and energy requirement $E_{ALG}(A)$ is

   $$E_{ALG}(A) = A\left(\frac{P}{t_v}\right)^{\alpha}$$

2. If $A > P$ then we need to run at higher speed $\frac{W-P}{D-t_v}$ in $[t_v, D]$ (ALG is late wrt off line algorithm);

   If you trust the prediction the energy consumption $E_P(A)$ is

   $$E_P(A) = P\left(\frac{P}{t_v}\right)^{\alpha} + (W-P)\left(\frac{W-P}{D-t}\right)^{\alpha}$$

Energy minimization: minimal example

We need to fix the virtual deadline t

Algorithm's energy requirementi wrt off-line (no trust to prediction)

- ALG requires less energy if A ≤ P
- ALG requires more energy if A > P

If the algorithm should be γ robust then it should require at most γ times the energy of the clarvoyant optimum in all possible cases

Since energy increases non lineraly with speed it is not hard to see that the bad case occurs when A= W (actual time = worst case time)

Energy minimization: minimal example

Fix the virtual deadline t that minimizes energy

- Since energy increases non lineraly with speed then the bad case occurs when A= WC (actual time = worst case time)
- It follows that the virtual deadline is equal to the largest value of t s.t.

$$E_{ALG}(W) \leq \gamma\, E(W)$$

- Equivalently

$$P\left(\frac{P}{t_v}\right)^{\alpha} + (W-P)\left(\frac{W-P}{D-t}\right)^{\alpha-1} \leq \gamma\, A\left(\frac{W}{D}\right)^{\alpha-1}$$

- If $\alpha = 2$ then we obtain a quadratic expression that is easy to solve

Jobs released over time

Schedule with minimum energy requirement 1 server

- a set J of n jobs; Input a set J of jobs: job $i \in J$ is defined by (i, r_i, p_i)
- A prediction $\hat{J}$ of the set of jobs: $(\hat{\imath}, \hat{r}_i, \hat{p}_i)$ predicted values
- Each job must be finished D_i time units after arrival

What metric to use to compare predicted workloaad and real one?

- Simplest metrics $||\,.\,||_1$, $||\,.\,||_2$ do not give enough information
- The exponent should take into account α
- err = $|| W_{pred} - W_{real} ||_{\alpha}^{\alpha}$

Jobs released over time

Simple case

assume $D_i = D$ and $r_i = i$ for all i [Bamas, et al.2020]

A first learning Algorithm

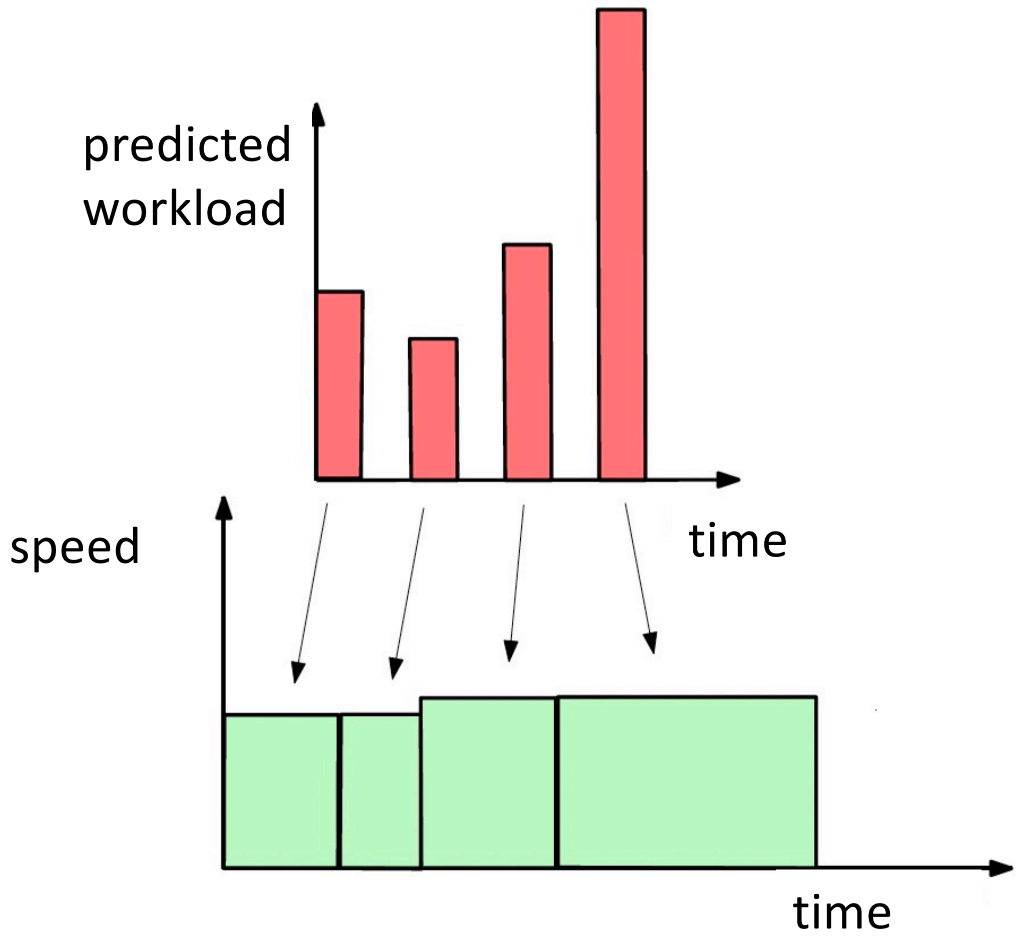
1. Compute offline the optimal solution for the prediction $\hat{J}$
2. At time instant t, t=1,2,….

- if job (i, r_i, p_i) is released and $(i, r_i, p_i) \in J \cap \hat{J}$ then do nothing
- if job (i, r_i, p_i) is mispredicted then increase/decrease speed: increase (decrease) speed in case of underprediction (overprediction)

Example

- Periodically the server receives a job to execute
- Each job comes with some workload w_i that must be finished within D milliseconds after arrival
- The server can choose its processor's speed $s(t)$ at will.
- The goal is to minimize the energy

$$\int s(t)^{\alpha} \, dt \text{ for } \alpha > 0$$

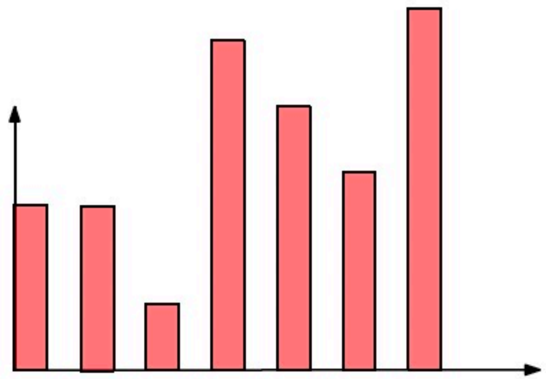


Example

- Every millisecond, the server receives a job to execute.
- Each job comes with some workload w_i that must be finished within D_i milliseconds after arrival.
- The server can choose its processor's speed $s(t)$ at will.
- The goal is to minimize the energy

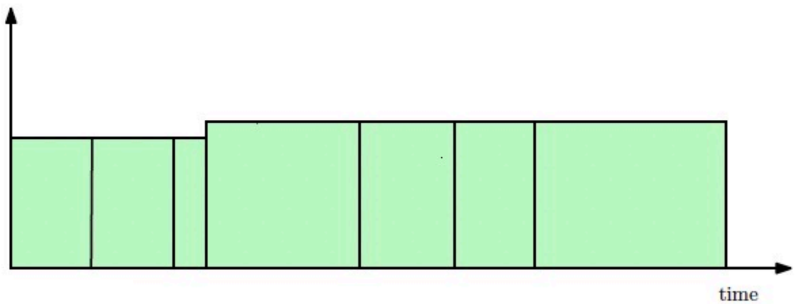
$$\int s(t)^{\alpha}\, dt \text{ for } \alpha > 0$$

Predicted workload



time

speed



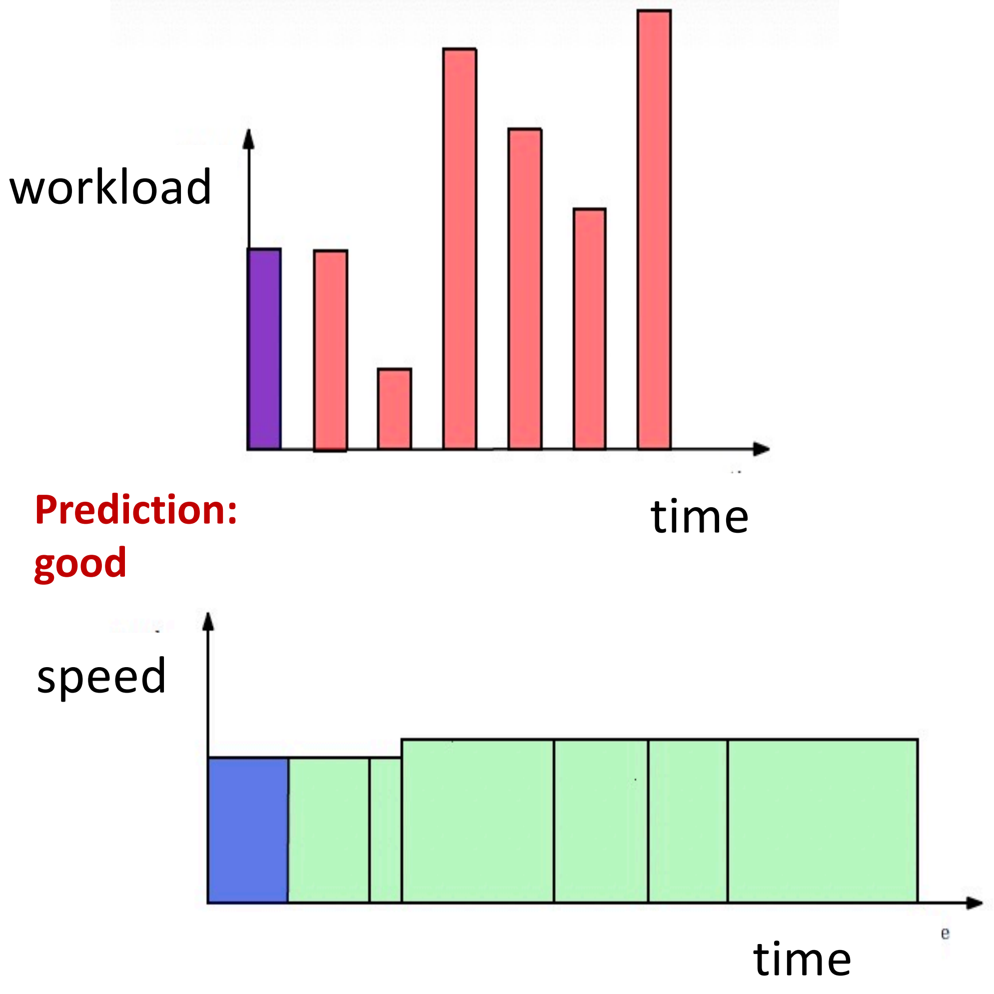
Predicted optimum

time

Example

- Periodically the server receives a job to execute
- Each job comes with some workload w_i that must be finished within D_i milliseconds after arrival
- The server can choose its processor's speed $s(t)$ at will.
- The goal is to minimize the energy

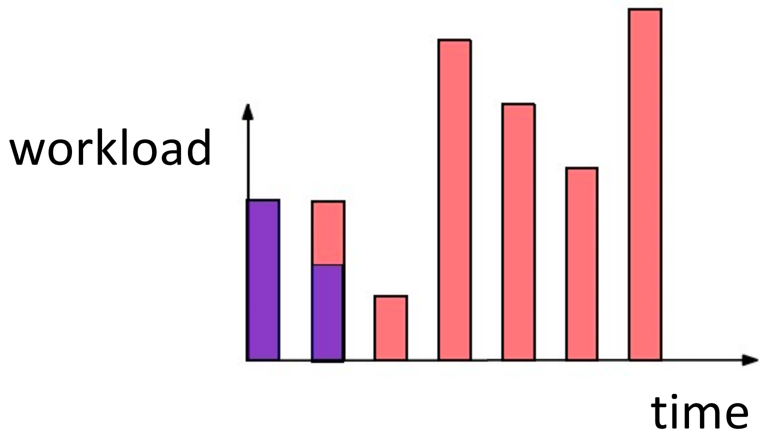
$$\int s(t)^{\alpha}\, dt \text{ for } \alpha > 0$$



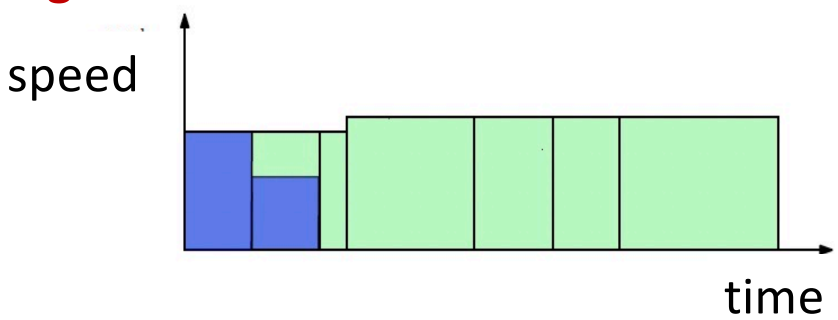
Example

- Periodically the server receives a job to execute
- Each job comes with some workload w_i that must be finished within D_i milliseconds after arrival
- The server can choose its processor's speed $s(t)$ at will.
- The goal is to minimize the energy

$$\int s(t)^{\alpha}\, dt \text{ for } \alpha > 0$$



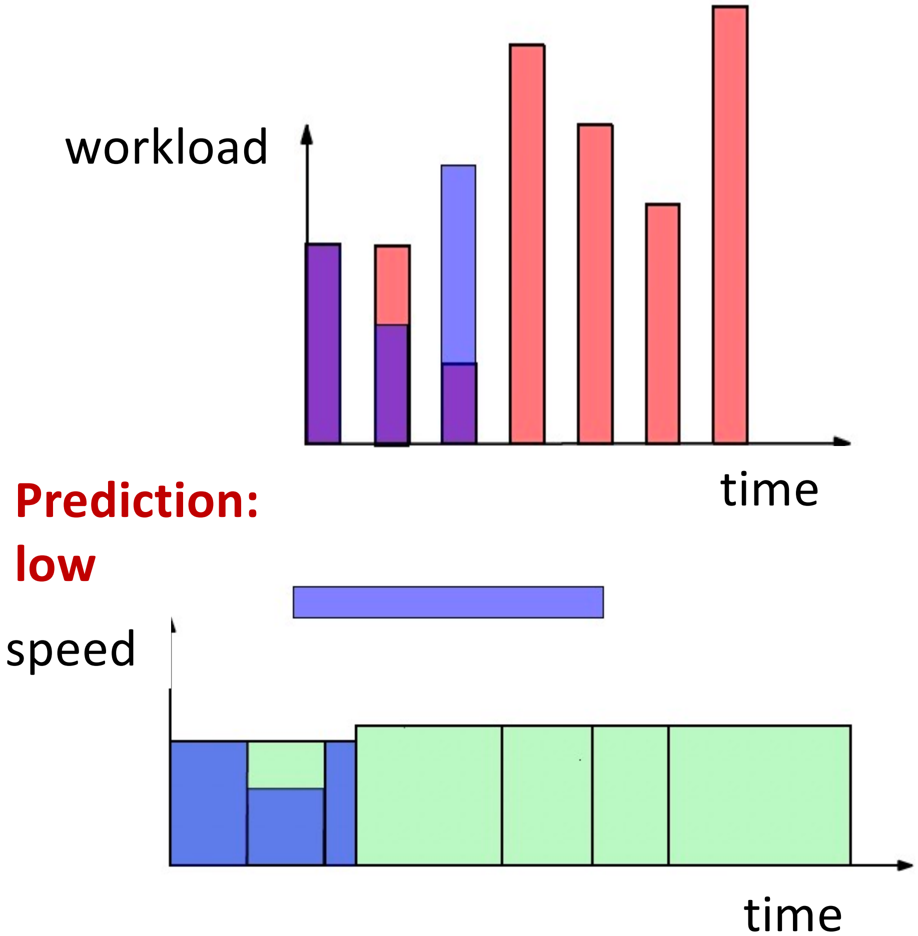
Prediction: high



Example

- Periodically the server receives a job to execute
- Each job comes with some workload w_i that must be finished within D_i milliseconds after arrival
- The server can choose its processor's speed $s(t)$ at will.
- The goal is to minimize the energy

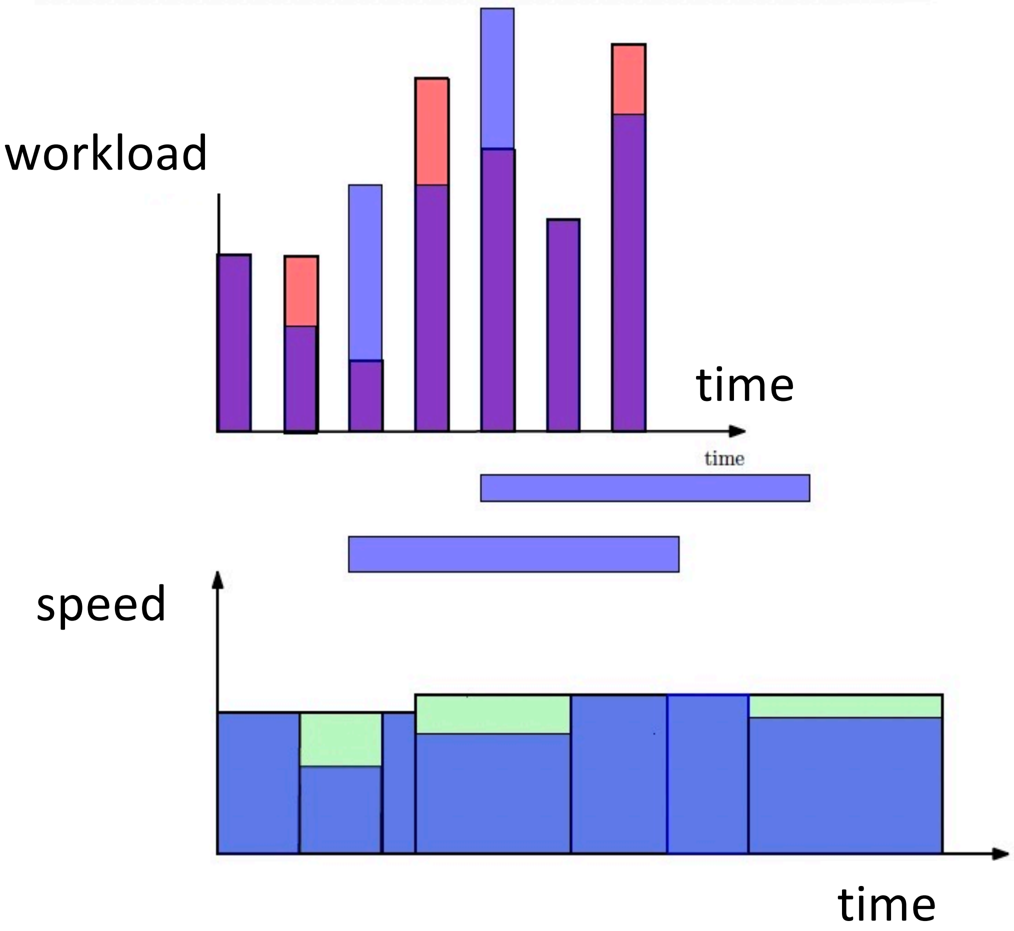
$$\int s(t)^{\alpha}\, dt \text{ for } \alpha > 0$$



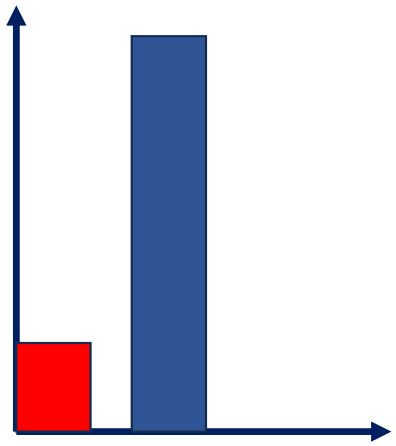
Example

- Periodically the server receives a job to execute
- Each job comes with some workload w_i that must be finished within D_i milliseconds after arrival
- The server can choose its processor's speed $s(t)$ at will.
- The goal is to minimize the energy

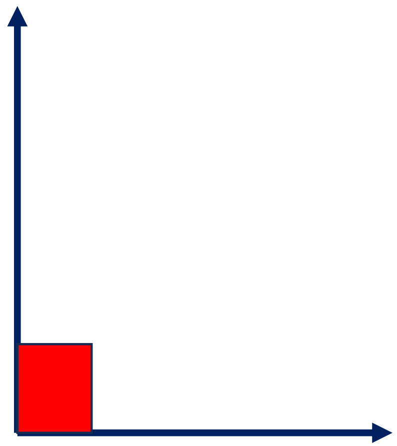
$$\int s(t)^{\alpha}\, dt \text{ for } \alpha > 0$$



Predicted workload



Workload

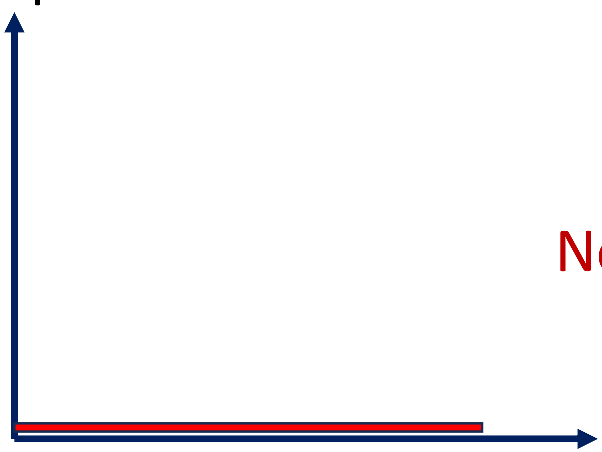


Perfect consistency

OK

Predicted solution

Optimal solution



But

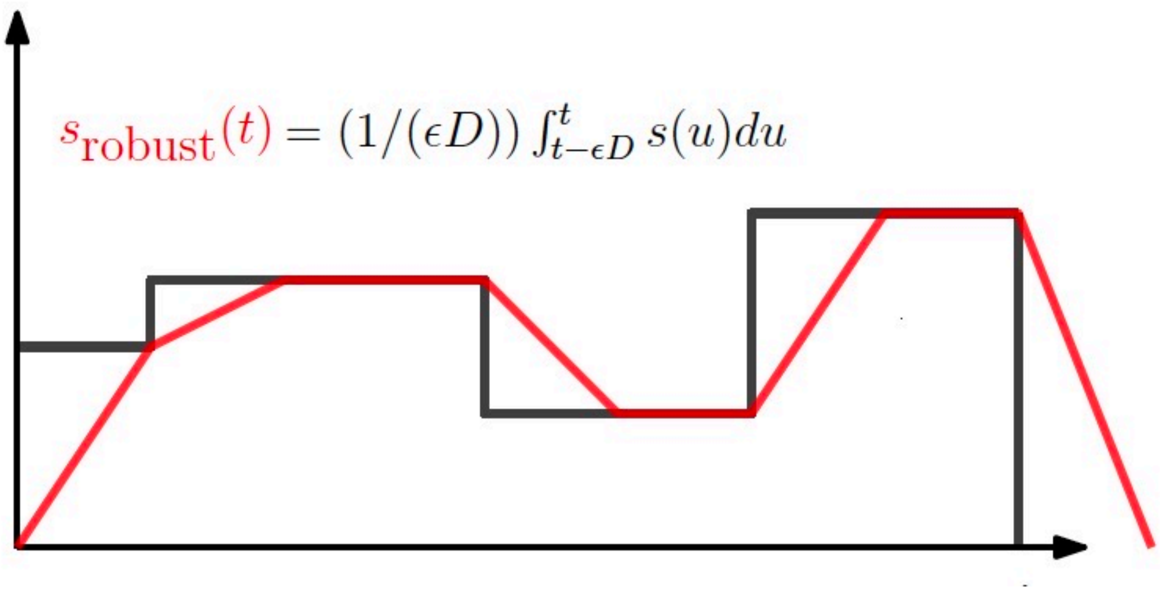
Not consistent

Energy minimization

The algorithm is consistent BUT is not robust

- How to make it robust?

Average out the speed to avoid huge peaks



- Generalize the simple case to general r_i and D_i

[Bamas et al. 2020]

General Energy-efficient Scheduling (GES)

Input

- a set J of jobs and a prediction $\hat{J}$ to be scheduled on one server
- an arbitrary quality of service function F(J,S), J set of jobs, S schedule

Given a schedule S , E(S) is its energy requirement minimize F(J,S) + E(S)

- Note jobs can have a deadline: if a job in J does not complete by its deadline in schedule S then F(J,S)= ∞

General Energy-efficient Scheduling [Balkansky et al. 2024]

- consistency and robustness bounds that are function of α (energy exponent) and of a parameter λ , $0 < \lambda \leq 1$ and their error parameter

General Energy-efficient Scheduling (GES)

Input a set J of jobs: job i $\in$ J is defined by (i, r_i, p_i)

- a set J of n jobs; job i has release time r_i and processing time p_i
- an arbitrary quality of service function F(J,S), J set of jobs, S schedule

Goal minimize F(J,S) + E(S) E(S) energy requirement of schedule S

- Prediction error μ: let $J^+ = J \cap \hat{J}$ be the set of correctly predicted jobs

$$\mu = \frac{\max\{OPT(\hat{J} \setminus J^+), OPT(\hat{J} \setminus J^+)\}}{OPT(\hat{J})}$$

$\max\{OPT(J \setminus J^+), OPT(J\hat{} \setminus J^+)\}$ is the maximum between

- the optimal cost of scheduling the jobs $J \setminus J^+$ that arrived but were not predicted
- the cost of the jobs $\hat{J} \setminus J^+$ that were predicted to arrive but did not arrive

General Energy-efficient Scheduling (GES) [Balkansky et al.]

Algorithm TPE: uses an OfflineAlg and OnlineAlg

Proceeds in two phases

1. ignore the predictions: until time t_λ runs the OnlineAlg over the true jobs $J_{\leq t}$ that have been released
2. use the predictions: after time t_λ runs two algorithms (summing speed)

   2.1 the OfflineAlg for remaining jobs that were correctly predicted (i.e., $J_{\geq t\lambda} \cap \hat{J}_{\geq t\lambda}$)

   2.2 the OnlineAlg for uncompleted jobs in Phase 1 and not predicted jobs released in the second phase

Theorem. For any $\lambda \in (0, 1)$, TPE with a c-competitive algorithm OnlineAlg and an optimal offline algorithm OfflineAlg is $1 + c\, 2^{\alpha} \lambda^{(1/\alpha)}$

General Energy-efficient Scheduling (GES) [Balkansky et al.]

Theorem. For any $\lambda \in (0, 1)$, TPE with a c-competitive algorithm OnlineAlg and an optimal offline algorithm OfflineAlg is $1 + c\, 2^{\alpha} \lambda^{(1/\alpha)}$ robust

Note: robustness > 1 implies that when the prediction is perfect then algorithm does not necessarily finds the optimal solution

Theorem For the objective of minimizing total energy plus (non-weighted) flow time, there is no algorithm that is 1-consistent and $o(\sqrt{n})$-robust, even if all jobs have unit-size work and if $J \subseteq \hat{J}$

Implicit Sporadic task system: predicting the period

An implicit deadline sporadic task τ_i is defined by three parameters:

- worst-case execution time C_i
- period T_i the minimum time between successive triggering of task τ_i

In many cases estimating C_i and T_i is challenging: safety critical often assign

- a *large safe upper bound* value to the worst execution time parameter
- a *small safe lower bound* value to the period parameter

This conservative approach often leads to underutilization of resources when jobs are released much further apart

Previous results on energy minimization assume that speed can assume any vlaure (and is unbounded.

Implicit Sporadic task system: predicting the period

Use prediction of the period deadline task systems
[Baruah, Ekberg, Lindermayr, MS, Megow, Stougie 2024]

Given a system $\Gamma = \text{U}_i \{\tau_i = (C_i, T_i, P_i)\}$ to be processed on one machine

- C_i the WCET, T_i deadline, P_i the period

We assume that each periodic task's period parameter is given two values:

- a conservative one that is guaranteed to be safe
- A more optimistic one that is very likely to be safe but it is not guaranteed to be safe

Assume that maximum speed is bounded

Goal: find an algorithm that

1. Runs at lower speed if predictions are correct
2. is safe if predictions are wrong

Run-Time Algorithm

Assume the maximum processor speed is 1

The Run-time scheduling algorithm

- Starts running the processor with speed s_0 , $s_0 < 1$ (*we implicitly assume predictions are good*)
- Monitors job-release time to check whether successive jobs of any task have been released sooner than P_i
- If so increases the processor speed up to its maximum; it remains at speed 1 until the processor is idle; at that instant returns to speed s_0

***Question:** What is the minimum value of s_0 that ensures the run-time algorithm always meets all job deadlines?*

Run-Time Algorithm: computing s_0

Computing s_0

What is the minimum value of s that ensures the run-time algorithm always meets all job deadlines under all circumstances?

1. we derive a necessary condition for a deadline miss for a given initial speed s_0
2. by negating this condition we obtain a formula to assign to s_0 a value that guarantees no deadline miss

Run-Time Algorithm: computing s_0

A necessary condition for a deadline miss for a given initial speed s_0

Assume we start with speed s_0. Let

- t_d be the earliest time at which a deadline miss can possibly occur
- $t_f < t_d$ be the earliest time at which a prediction failure occurred
- $\delta_i(t_f, t_d)$ be a tight upper bound on the cumulative execution by jobs of task i in the interval $[t_f, t_d]$

1. We prove that if we run at speed s_0 in $[0, t_f]$ and speed 1 in $[t_f, t_d]$

   ⇒ *a failure at t_f implies $\Sigma_i\ \delta_i(t_f, t_d) > s_0\ t_f + 1\ (t_d - t_f)$*

2. Hence if $s_0 > \{[\Sigma_i\ \delta_i(t_f, t_d)] - (t_d - t_f)\} / t_f$ for all values t_f, t_d

   ⇒ *no deadline is missed*

Computing s_0 - running time

Computing s_0 the minimum value of s that ensures correctness

We have shown that if

$$s_0 > \{[\sum_i \delta_i(t_f, t_d)] - (t_d - t_f)\} / t_f \text{ for all values } t_f, t_d$$

then there is no deadline miss

How to compute $[\sum_i \delta_i(t_f, t_d) - (t_d - t_f)] / t_f$ for all values t_f, t_d ?

- Given **task τ_i, and time instants t_f, t_d** then $\delta_i(t_f, t_d)$ can be computed in constant time
- Hence $[\sum_i \delta_i(t_f, t_d) - (t_d - t_f)] / t_f$ can be computed in *pseudopolynomial time* for all values t_f, t_d (we assume release instants and parameter to be integer)

Computing s_0 – approximation algorithm

Computing s_0 the minimum value of s that ensures correctness

- **How to compute $[\sum_i \delta_i(t_f, t_d) - (t_d - t_f)] / t_f$ for all values t_f, t_d**

We propose an **approximation algorithm** based on

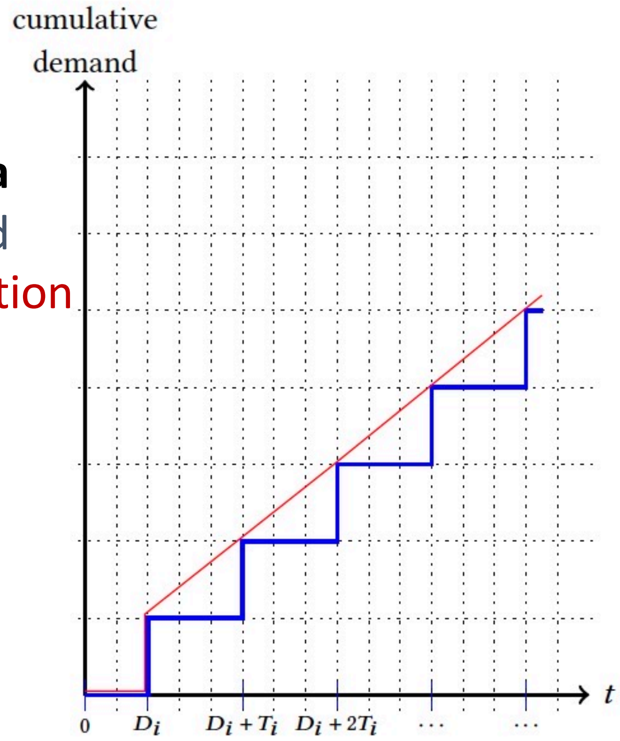
1. Determining for each pair of values t_f, t_d the worst case scenario
2. Using Albers-Smolka approximation of the DBF function
3. Discretizing the considered values to reduce the number of interesting values (the running time increases with the quality of the approximation)

Run-Time Algorithm

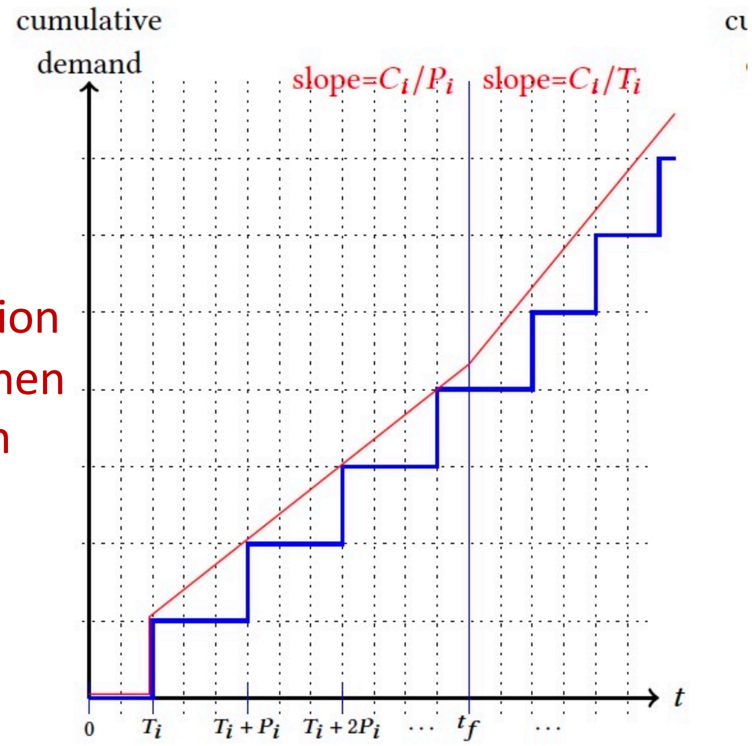
Computing s_0 the minimum value of s that ensures correctness

2. ***Using Albers-Smolka approximation of the DBF function***

Albers-Smolka
DBF (blue) and
AS-approximation
of DBF (red)



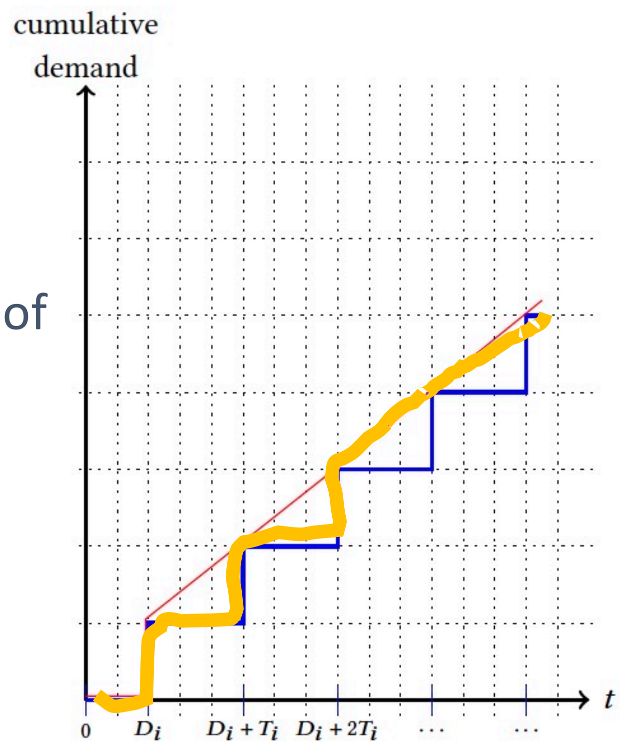
Albers-Smolka
DBF (blue) and
AS-approximation
of DBF (red) when
a misprediction
occurs at t_f



Run-Time Algorithm

2. *Using Albers-Smolka approximation of the DBF function*
3. ***Discretizing the considered values to reduce the number of interesting values;***

Albers-Smolka
Black
Approximation of DBF k= 3



For a given integer k traces the dbf for the first *k* steps and its approximation for subsequent steps

Large *k* ➡ higher computation cost and better precision

Let s_k be the speed computed by the algorithm with parameter k

➡ $(1 – 2/k)\ s_k$ is a lower bound on minimal speed s_0

Predicting the period

- Results can be extended to constrained deadline task systems
- We do not consider
  - smoothness
  - multiprocessors

Conclusions and questions

- Machine learning is the present
- Learning-augmented seems like a potentially general and powerful paradigm to go beyond worst case analysis

- Learning augmented algorithms and real time scheduling
  - What problems are amenable to advice from learning algorithms?
  - Which data can be learned?
  - What should be predicted – is it realistic?
  - Are there minimal prediction that are practically useful (i.e. allow to obtain better bounds and have good robustness)?
  - What is a reasonable (for the community) error measure (that allows to prove good bounds)?

Thanks for your attention!

Questions?