Maximum constrained sequencing to minimize tardy task weight

task

length

task

length
deadline
task

length
deadline
task weight

length
deadline
task weight
isPriority

task

sequencing task

sequencing to minimize
task weight

sequencing to minimize
tardy task weight

Maximum constrained sequencing to minimize tardy task weight

MAXIMUM CONSTRAINED SEQUENCING TO MINIMIZE TARDY TASK WEIGHT

- INSTANCE: Set T of tasks, for each task $t \in T$ a length $l(t) \in Z^+$, a weight $w(t) \in Z^+$, and a deadline $d(t) \in Z^+$, a subset $S \subseteq T$, and a positive integer K.
- SOLUTION: A one-processor schedule σ for T such that the sum of $w(t)$, taken over all $t \in T$ for which $\sigma(t) + l(t) > d(t)$ does not exceed K.
- MEASURE: Cardinality of jobs in S completed by the deadline.

MAXIMUM CONSTRAINED SEQUENCING TO MINIMIZE TARDY TASK WEIGHT

- INSTANCE: Set T of tasks, for each task $t \in T$ a length $l(t) \in Z^+$, a weight $w(t) \in Z^+$, and a deadline $d(t) \in Z^+$, a subset $S \subseteq T$, and a positive integer K.
- SOLUTION: A one-processor schedule σ for T such that the sum of $w(t)$, taken over all $t \in T$ for which $\sigma(t) + l(t) > d(t)$ does not exceed K.
- MEASURE: Cardinality of jobs in S completed by the deadline.

K = maximum allowable tardy weight

MAXIMUM CONSTRAINED SEQUENCING TO MINIMIZE TARDY TASK WEIGHT

- INSTANCE: Set T of tasks, for each task $t \in T$ a length $l(t) \in Z^+$, a weight $w(t) \in Z^+$, and a deadline $d(t) \in Z^+$, a subset $S \subseteq T$, and a positive integer K.
- SOLUTION: A one-processor schedule σ for T such that the sum of $w(t)$, taken over all $t \in T$ for which $\sigma(t) + l(t) > d(t)$ does not exceed K.
- MEASURE: Cardinality of jobs in S completed by the deadline.

K = maximum allowable tardy weight

Maximize tasks done in subset S.

Brute force solution

```
def evaluate_schedule(tasks, permutation):
    # simulate schedule

    if total_tardy_weight <= K and s_tasks_on_time > optimal_schedule:
        # update optimal schedule

def generate_permutations(tasks):
    # ... code from permutation.c
      permutation = [0, 4, 2, 1, 3]
      evaluate_schedule(tasks, permutation)
    # ...

tasks = import_tasks()
generate_permutations(tasks)
```

```
def evaluate_schedule(tasks, permutation):
    # simulate schedule

    if total_tardy_weight <= K and s_tasks_on_time > optimal_schedule:
        # update optimal schedule

def generate_permutations(tasks):
    # ... code from permutation.c
      permutation = [0, 4, 2, 1, 3]
      evaluate_schedule(tasks, permutation)
    # ...

tasks = import_tasks()
generate_permutations(tasks)
```

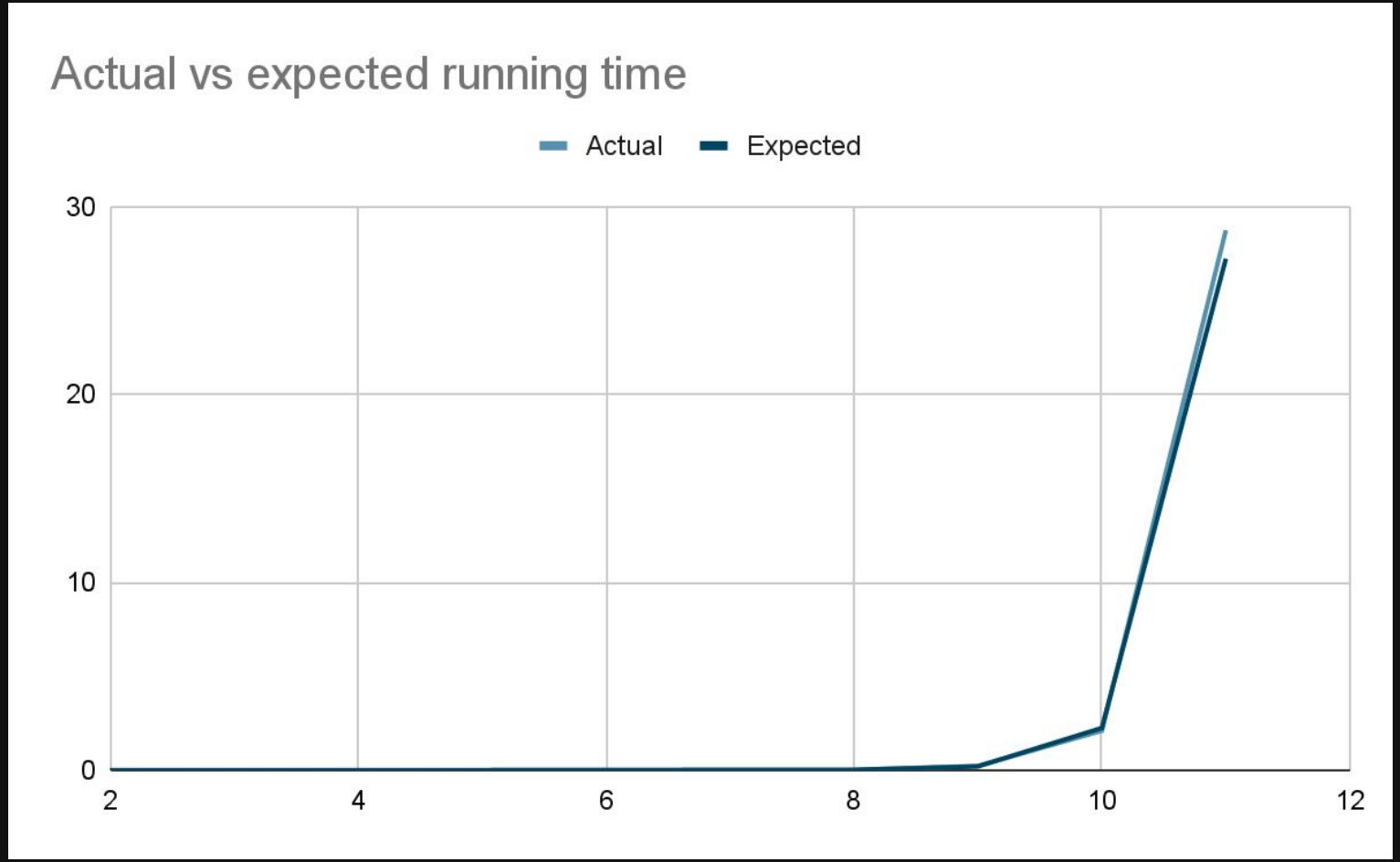
O(n!)

```
def evaluate_schedule(tasks, permutation):
    # simulate schedule

    if total_tardy_weight <= K and s_tasks_on_time > optimal_schedule:
        # update optimal schedule

def generate_permutations(tasks):
    # ... code from permutation.c
        permutation = [0, 4, 2, 1, 3]
        evaluate_schedule(tasks, permutation)
    # ...

tasks = import_tasks()
generate_permutations(tasks)
```

O(n)

O(n!)

$O(n! \times n)$


Actual vs expected running time
Actual
Expected
30
20
10
0
2
4
6
8
10
12

Non-exhaustive solution

O(n^2)

Real life applications

OS scheduling

Tasks: processes, threads
Length: CPU time to complete a task
Deadline: expiration time, desired response time
Weight: grade percentage
S: system processes, high priority processes

Task prioritization

Tasks: tasks, requirements, activities
Length: time to complete task
Deadline: due date
Weight: grade percentage
S: blocking tasks, high impact tasks

Ad placement

Tasks: articles, ad
Length: space taken
Deadline: must appear before page x
Weight: revenue lost
S: front page articles, high paying ads
K: maximum tolerable lost revenue