

Scheduling Theory and Applications. Part II

Alexander Lazarev

Lomonosov Moscow State University

National Research University Higher School of Economics

Moscow Institute of Physics and Technology (State University)

V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences (ICS RAS)

jobmath@mail.ru

www.orsot.ru



V.A. TRAPEZNIKOV
INSTITUTE
OF CONTROL
SCIENCES
OF RUSSIAN ACADEMY
OF SCIENCES



Outline I

- 1 NP-hard problems
 - Recognition problem
 - Computational complexity
- 2 Problem classification and denotations in scheduling theory
- 3 Problem $1|r_j|L_{\max}$
 - Minimizing maximum lateness
 - Solvable cases
 - Algorithms
 - Pareto optimal schedules
 - Metric
 - Metric + Application
 - Absolute error
 - Linear programming problem
 - Any not decreasing penalty functions
 - Dual problem

Elements of computational complexity theory. Classes P and NP .

Any finite set of symbols is called an **alphabet** Σ .

Any finite string of symbols of an alphabet is called a **word**.

The set of all possible **words** over the alphabet Σ is denoted as Σ^* .

Any subset $L \subset \Sigma^*$ is called a **language** over the alphabet Σ .

Recognition problem

Any finite set of symbols is called an **alphabet** Σ .

Any finite string of symbols of an alphabet is called a **word**.

The set of all possible **words** over the alphabet Σ is denoted as Σ^* .

Any subset $L \subset \Sigma^*$ is called a **language** over the alphabet Σ .

The **recognition problem** for language L over alphabet Σ is to define whether or not a given word belongs to language L .

Example.

Consider a classic knapsack problem: given a knapsack of maximum capacity V , and n items with volumes v_i and costs c_i , $i = 1, \dots, n$, we need to take some of them, so that total volume of taken items does not exceed the capacity of the knapsack, and their total cost is maximal.

Recognition problem

Knapsack problem:

Input data:

$V \in \mathbb{R}^+$ — knapsack capacity

Knapsack problem:

Input data:

$V \in \mathbb{R}^+$ — knapsack capacity

$v_i, \in \mathbb{R}^+$ — volumes of items

$c_i \in \mathbb{R}^+$ — costs of items

Knapsack problem:

Input data:

$V \in \mathbb{R}^+$ — knapsack capacity

$v_i, \in \mathbb{R}^+$ — volumes of items

$c_i \in \mathbb{R}^+$ — costs of items

Restrictions:

$x_i \in \{0, 1\}$ — $x_i = 1$ if item i is taken and $x_i = 0$ if it's not

Knapsack problem:

Input data:

$V \in \mathbb{R}^+$ — knapsack capacity

$v_i, \in \mathbb{R}^+$ — volumes of items

$c_i \in \mathbb{R}^+$ — costs of items

Restrictions:

$x_i \in \{0, 1\}$ — $x_i = 1$ if item i is taken and $x_i = 0$ if it's not

$$\sum_{i=1}^n v_i * x_i \leq V$$

Knapsack problem:

Input data:

$V \in \mathbb{R}^+$ — knapsack capacity

$v_i, \in \mathbb{R}^+$ — volumes of items

$c_i \in \mathbb{R}^+$ — costs of items

Restrictions:

$x_i \in \{0, 1\}$ — $x_i = 1$ if item i is taken and $x_i = 0$ if it's not

$$\sum_{i=1}^n v_i * x_i \leq V$$

Objective function:

$$\sum_{i=1}^n c_i * x_i \rightarrow \max$$

Knapsack problem:

Input data:

$V \in \mathbb{R}^+$ — knapsack capacity

$v_i, \in \mathbb{R}^+$ — volumes of items

$c_i \in \mathbb{R}^+$ — costs of items

Restrictions:

$x_i \in \{0, 1\}$ — $x_i = 1$ if item i is taken and $x_i = 0$ if it's not

$$\sum_{i=1}^n v_i * x_i \leq V$$

Objective function:

$$\sum_{i=1}^n c_i * x_i \rightarrow \max$$

Knapsack problem can be solved by a pseudopolynomial algorithm (dynamic programming) with computational complexity $O(nV)$

Recognition problem

In this example, 0 and 1 are the symbols that form alphabet $\Sigma = \{0, 1\}$.

Recognition problem

In this example, 0 and 1 are the symbols that form alphabet $\Sigma = \{0, 1\}$.

Any finite sequence of these symbols (e. g., '011010101110') is a word over the alphabet Σ . The set Σ^* of all possible words over this alphabet would be, of course, the set of all finite sequences of ones and zeros.

In this example, 0 and 1 are the symbols that form alphabet $\Sigma = \{0, 1\}$.

Any finite sequence of these symbols (e. g., '011010101110') is a word over the alphabet Σ . The set Σ^* of all possible words over this alphabet would be, of course, the set of all finite sequences of ones and zeros.

Let us form a language L that would contain all the words from Σ^* that 1) are exactly of length n , 2) follow the capacity restriction and 3) deliver maximum possible total cost (supposed that we know it).

Recognition problem

The **recognition problem**, in this case, would be to check if a given word σ — a sequence of n ones and zeros — belongs to the language L , i. e. this sequence occurs to be a solution of this knapsack problem.

Recognition problem

The **recognition problem**, in this case, would be to check if a given word σ — a sequence of n ones and zeros — belongs to the language L , i. e. this sequence occurs to be a solution of this knapsack problem. Running ahead, this instance of recognition problem is solvable in polynomial time if we know the maximum possible total cost (because we don't have to know exactly all the words from language L , we just have to know how it's formed).

The **recognition problem**, in this case, would be to check if a given word σ — a sequence of n ones and zeros — belongs to the language L , i. e. this sequence occurs to be a solution of this knapsack problem.

Running ahead, this instance of recognition problem is solvable in polynomial time if we know the maximum possible total cost (because we don't have to know exactly all the words from language L , we just have to know how it's formed).

However, the formulated knapsack problem itself is NP-complete, i.e. the problem of finding the maximum possible total cost is not solvable in polynomial time. Let's discuss what all of this means.

Suppose we are examining some instance of a recognition problem.

Suppose we are examining some instance of a recognition problem. Let us denote as n a numerical characteristic of input data that affects computational complexity of this problem in the most significant way (usually it is either the amount of input data itself, or dimensionality of the problem — the number of variables, equations and inequalities that define an instance of the problem).

Suppose we are examining some instance of a recognition problem.

Let us denote as n a numerical characteristic of input data that affects computational complexity of this problem in the most significant way (usually it is either the amount of input data itself, or dimensionality of the problem — the number of variables, equations and inequalities that define an instance of the problem).

Suppose we also know some sort of algorithm that can be used to solve this problem within a finite time period.

- If computational complexity of the algorithm that solves the problem is $O(n^k)$ operations, where k is some constant number independent from n , then this problem is called **solvable in polynomial time**. Algorithms to the 4 problems mentioned before (Jackson's, Smith's, Johnson's problems and the problem of two production lines) are polynomial.

- If computational complexity of the algorithm that solves the problem is $O(n^k)$ operations, where k is some constant number independent from n , then this problem is called **solvable in polynomial time**. Algorithms to the 4 problems mentioned before (Jackson's, Smith's, Johnson's problems and the problem of two production lines) are polynomial.
- All problems that are solvable within polynomial time formulate a class of problems denoted as P . Algorithms with corresponding computational complexity are called **polynomial**.

- If computational complexity of the algorithm that solves the problem is $O(n^k)$ operations, where k is some constant number independent from n , then this problem is called **solvable in polynomial time**. Algorithms to the 4 problems mentioned before (Jackson's, Smith's, Johnson's problems and the problem of two production lines) are polynomial.
- All problems that are solvable within polynomial time formulate a class of problems denoted as P . Algorithms with corresponding computational complexity are called **polynomial**.
- If complexity of the algorithm depends on the values of numerical parameters of an example, for example, $O(nA)$, then this algorithm is called **pseudo-polynomial**.

- If computational complexity of the algorithm that solves the problem is $O(n^k)$ operations, where k is some constant number independent from n , then this problem is called **solvable in polynomial time**. Algorithms to the 4 problems mentioned before (Jackson's, Smith's, Johnson's problems and the problem of two production lines) are polynomial.
- All problems that are solvable within polynomial time formulate a class of problems denoted as P . Algorithms with corresponding computational complexity are called **polynomial**.
- If complexity of the algorithm depends on the values of numerical parameters of an example, for example, $O(nA)$, then this algorithm is called **pseudo-polynomial**.
- If complexity of the algorithm has the form of $O(n^x y^n)$, where x and y are some constants, then this algorithm is called **exponential**.

Suppose that we have a computer that includes a special "guessing" component (**oracle**).

The oracle, given correct input data (i.e. a solution to the given instance exists), provides some (possibly correct) output data.

The output data provided by oracle needs to be verified, i. e. we should construct an algorithm that checks if the output data contains a correct solution that is in accordance with provided input data. The problem of verifying data provided by oracle could also be formulated as an instance of recognition problem.

Class **NP** includes all the problems to which the solution (if such exists) can be guessed by an oracle, and:

Class **NP** includes all the problems to which the solution (if such exists) can be guessed by an oracle, and:

- The amount of data in solution provided by oracle is polynomially bounded;

Class **NP** includes all the problems to which the solution (if such exists) can be guessed by an oracle, and:

- The amount of data in solution provided by oracle is polynomially bounded;
- The solution provided by oracle can be verified in polynomial time.

Reduction of one problem to another

It is said that problem A **can be reduced** to problem B in polynomial time ($A \propto B$), if a modification algorithm exists, such that:

Reduction of one problem to another

It is said that problem A **can be reduced** to problem B in polynomial time ($A \propto B$), if a modification algorithm exists, such that:

- The algorithm transforms any given instance I_A of problem A into a corresponding instance I_B of problem B in polynomial time

Reduction of one problem to another

It is said that problem A **can be reduced** to problem B in polynomial time ($A \propto B$), if a modification algorithm exists, such that:

- The algorithm transforms any given instance I_A of problem A into a corresponding instance I_B of problem B in polynomial time
- The answer to received instance I_B of problem B is **"YES"** if and only if the answer to the corresponding instance I_A of problem A is **"YES"**, too. (Or, less strictly, the solutions of corresponding instances I_A, I_B of problems A, B always match)

NP-complete and NP-hard problems

Problem B is called **NP-hard**, if any other problem $A \in NP$ can be reduced to problem B in polynomial time.

NP-complete and NP-hard problems

Problem B is called **NP-hard**, if any other problem $A \in NP$ can be reduced to problem B in polynomial time.

Problem B is called **NP-complete**, if:

- B is NP-hard;
- B belongs to class NP .

If any NP-complete problem is solvable in polynomial time, then all of the NP-complete are solvable in polynomial time ($P = NP$).

NP-complete and NP-hard problems

Problem B is called **NP-hard**, if any other problem $A \in NP$ can be reduced to problem B in polynomial time.

Problem B is called **NP-complete**, if:

- B is NP-hard;
- B belongs to class NP .

If any NP-complete problem is solvable in polynomial time, then all of the NP-complete are solvable in polynomial time ($P = NP$).

NP-hard problem B is called **NP-hard in the strong sense** if there is no pseudo-polynomial algorithm of solving this problem (supposed that $P \neq NP$).

In 1971 Stephen Arthur Cook proved that *NP*-complete problems exist on example of Boolean Satisfiability Problem (SAT problem), which is the problem of determining if there is any interpretation that satisfies a given Boolean formula (makes it equal to TRUE). Here, an interpretation is a combination of values of all the Boolean variables in the formula.

S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of 3rd Annual ACM Symposium on Theory of Computing*, pg. 151-158. ACM-Press 1971.

Problem classification and denotations in scheduling theory

In scheduling theory, problems are classified according to:

- Type of solution
- Type of objective function
- How input data is provided
- Subfields of Scheduling Theory

Problem classification according to the type of solution:

- Arrangement problems
- Matching problems
- Distribution problems

Problem classification according to the type of objective function:

- Problems with summary optimization criteria
- Problems with min-max optimization criteria
- Multicriterial optimization problems
- Problems of constructing a feasible schedule

Problem classification according to the how input data is provided:

- Deterministic problems (**offline**)
- Dynamic problems (**online**)

Problem classification according to subfields of Scheduling Theory:

- Project scheduling (PS)
- Machine scheduling (MS)
- Timetabling
- Shop-floor scheduling
- Transport scheduling and vehicle routing
- Sports scheduling
- Medical scheduling
- ... and so on.

In Scheduling Theory, tasks are referred to as *requests* or *jobs*. General parameters of requests:

- r_j — release time
- p_j — processing time
- d_j — due date (may be violated, but a penalty is issued)
- D_j — deadline (should never be violated)
- w_j — job weight

Additional denotations:

- *pmtn* — preemptive scheduling is allowed
- *prec* — precedence relations between the jobs are defined (also: *tree*, *out – tree*, *in – tree*, *chain*)
- *batch* — the batching problem is considered (jobs are grouped into batches)

Objective functions:

- C_j — completion time
- $L_j = C_j - d_j$ — lateness
- $T_j = \max\{0, C_j - d_j\}$ — tardiness
- $E_j = \max\{0, d_j - C_j\}$ — earliness
- U_j — unit penalty: equals 1 if job j is late ($C_j > d_j$) and 0 in the opposite case

If request weights w_j are provided, all of the previous objective functions are called *weighed*, and are multiplied by the value of request weight (ex., weighed tardiness $w_j T_j$ is calculated as $w_j \max\{0, C_j - d_j\}$)

Optimization criteria:

1. Min-max criteria

- $C_{max} \rightarrow \min$ — minimizing maximum completion time (makespan), $C_{max} = \max_{j \in N} C_j$. These problems are also called *performance problems*.
- $L_{max} \rightarrow \min$ — minimizing maximum lateness $L_{max} = \max_{j \in N} L_j$

2. Summary criteria

- $\sum_{j \in N} C_j \rightarrow \min$ — minimizing total completion time
- $\sum_{j \in N} T_j \rightarrow \min$ — minimizing total tardiness
- $\sum_{j \in N} U_j \rightarrow \min$ — minimizing total number of late jobs

Also, problems of maximizing these objective functions are considered (ex.,

$\sum_{j \in N} T_j \rightarrow \max$).

Project scheduling

Resource-Constrained Project Scheduling Problem (RCPSP)

- Set of n requests $N = \{1, \dots, n\}$
- k renewable resources $K = 1, \dots, Q_k$
- p_i — processing time of request i , $\forall i \in N$.
- During processing of request i amount $q_{ik} \leq Q_k$ of resource k is used, $k = 1, \dots, n$.
- Some requests are bound by *precedence relations*: $i \rightarrow j$ means request j cannot start processing before request i has finished processing, $i, j \in N$.

Resource-Constrained Project Scheduling Problem (RCPSP)

The goal is to find processing start times S_i for all requests $i \in N$ so that minimum makespan C_{max} is achieved:

$$C_{max} = \max_{i \in N} \{C_i\}, \quad C_i = S_i + p_i, \quad C_{max} \rightarrow \min$$

Obtained schedule should comply to the following conditions:

- Resource constraints are not violated:

$$\forall t \in [0, C_{max}), \forall k = 1, \dots, K \quad \sum_{i=1}^n q_{ik} \varphi_i(t) \leq Q_k$$

- Precedence relations are not violated:

$$\forall i, j \in N : \text{if } i \rightarrow j, \text{ then } S_i + p_i \leq S_j$$

It is necessary to notice that RCPSP is not the only problem in project scheduling, though it is the main one. For example, some resources can be non-renewable, such as money, fuel, oils and so on.

Machine scheduling

In Project Scheduling, processing of each request requires participation of several *processors* (renewable resources could be viewed as equipment).

In Machine scheduling, usually each request is processed by *only one processor at a time*.

Processors can also be referred to as *machines* or *devices*. If not specified otherwise, all machines are considered equivalent.

- *Single-machine problems*: only one request can be processed at a time.
- *Parallel machines' problems*: each request can be processed by any of the machines. Machines can be non-equivalent (processing time can vary). Precedence relations can be specified.
- *Shop scheduling*: m , machines M_1, \dots, M_m . Each request $j \in N$ includes a number of *stages* ("operations") O_1, \dots, O_{n_j} . Precedence relations between operations can be specified. Each operation O_{ij} is assigned to a machine μ_{ij} that it should be processed on. For each request, only one operation can be processed at a time. Each machine can only process one operation at a time.
- *Job-shop*: Precedence relations between operations are $O_{1j} \rightarrow O_{2j} \rightarrow \dots \rightarrow O_{n_j}$. No precedence relations between requests. Number of operations may vary between requests.

- *Flow-shop* ("Conveyor problem"): Each request contains the same number of operations: $\forall j \in N \ n_j = m$. Same operations are assigned to the same machine: $\mu_{ij} = M_i, \ i = 1, \dots, m, \ j = 1, \dots, n$.
- *Open-shop*: same as Flow-shop, but no precedence relations between operations.
- *Other problems*: batching problems, multiprocessor problems, ...

Classification of problems in Machine scheduling

Classification of problems in Machine scheduling

Each problem is denoted as $\alpha|\beta|\gamma$, where

- α describes characteristics of the problem that are related to machines
- β describes constraints and conditions of processing of requests.
- γ describes objective function.

Classification of problems in Machine scheduling

α describes characteristics of the problem related to machines. Possible values of α :

- 1 — single machine
- Pm — parallel machines
- Qm — parallel machines (non-equivalent)
- Fm — Flow-shop problem
- Om — Open-shop problem
- Jm — Job-shop problem
- ...

Classification of problems in Machine scheduling

β describes constraints and conditions of processing of requests. Possible contents of field β :

- r_j — release dates are specified
- d_j — due dates are specified
- D_j — deadlines are specified
- $prec$ — precedence relations are specified
- $pmnt$ — preemption is allowed
- $batch$ — batching problem: groups of requests (*batches*) can be processed simultaneously.
- Other conditions: $p_j = p, \dots$

γ describes objective function (e.g., C_{max}).

Classification of problems in Machine scheduling

Thus, record $F2|r_j|C_{max}$ denotes problem of minimizing makespan in Flow-shop system with two machines in case of non-simultaneous admission of requests. Other examples: $1|p_j = p, r_j|\sum w_j T_j$, $Pm|r_j, pmtn|\sum C_j, \dots$

Classification of problems in Machine scheduling

Thus, record $F2|r_j|C_{max}$ denotes problem of minimizing makespan in Flow-shop system with two machines in case of non-simultaneous admission of requests. Other examples: $1|p_j = p, r_j|\sum w_j T_j$, $Pm|r_j, pmtn|\sum C_j, \dots$

Let's review some of previously considered problems in terms of machine scheduling:

- $1|r_j|L_{max}$ (Jackson's problem with non-zero release times) is **NP-hard in the strong sense**
- $1|r_j|\sum C_j$ (Smith's problem with non-zero release times) is **NP-hard**
- $F3||C_{max}$ (Johnson's problem with more than 2 machines) is **NP-hard in the strong sense**

An Approximation Scheme for the $1|r_j|\sum T_j$ Scheduling Problem with Guaranteed Absolute Error

- 1 Formulation problem
- 2 Metric for the problem
- 3 Approximation scheme
- 4 Computational experiments
- 5 Further research

Set $N = \{1, 2, \dots, n\}$ of n independent jobs that must be processed on a **single machine**.

Set $N = \{1, 2, \dots, n\}$ of n independent jobs that must be processed on a **single machine**.

- The machine can handle only one job at a time.

Set $N = \{1, 2, \dots, n\}$ of n independent jobs that must be processed on a **single machine**.

- The machine can handle only one job at a time.
- Preemptions are not allowed.

Set $N = \{1, 2, \dots, n\}$ of n independent jobs that must be processed on a **single machine**.

- The machine can handle only one job at a time.
- Preemptions are not allowed.
- The machine is ready to start processing at time 0.

Set $N = \{1, 2, \dots, n\}$ of n independent jobs that must be processed on a **single machine**.

- The machine can handle only one job at a time.
- Preemptions are not allowed.
- The machine is ready to start processing at time 0.

For each job j , $j \in N$, a processing time $p_j \geq 0$, release date $r_j \geq 0$ and due date d_j are given.

A schedule describes order of processing the jobs: a permutation (sequence) $\pi = (j_1, j_2, \dots, j_n)$.

A schedule describes order of processing the jobs: a permutation (sequence) $\pi = (j_1, j_2, \dots, j_n)$.

- $S_j(\pi)$ denotes the **starting time** of the processing of job j in schedule π .
- $C_j(\pi) = S_j(\pi) + p_j$ denotes the **completion time** of the processing of job j in schedule π .
- In **early schedule**: π $S_{j_1} = r_{j_1}$ and $S_{j_k} = \max\{r_{j_k}, C_{j_{k-1}}\}$ for $k = 2, \dots, n$,

- $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$ denotes **tardiness** of job j in schedule π .
- $\sum_{j \in N} T_j(\pi)$ is the **total tardiness** in schedule π .

- $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$ denotes **tardiness** of job j in schedule π .
- $\sum_{j \in N} T_j(\pi)$ is the **total tardiness** in schedule π .

The total tardiness minimization problem is denoted by $1|r_j|\sum T_j$.

Du J., Leung J.Y.-T, *Minimizing total tardiness on one machine is NP-hard*
Mathematics of Operations Research, Vol. 15. 1990 г., №3, P. 483 – 495.
Problem $1||\sum T_j$ is **NP-hard in the ordinary sense**.

Lawler E.L., *A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness*, Ann. Discrete Math., Vol. 1, 1977, 331 – 342.

A pseudo-polynomial algorithm of time complexity $O(n^4 \sum p_j)$ for $1||\sum T_j$ problem is presented

A.A. Lazarev, F. Werner., *Algorithms for Special Single Machine Total Tardiness Problem and an Application to the Even-Odd Partition Problem*. Math. and Comp. Model. 49:2078–2089 (2009).

A pseudo-polynomial algorithm of time complexity $O(n^2 \sum p_j)$ for the case

$$d_1 \leq d_2 \leq \dots \leq d_n,$$

$$p_1 \geq p_2 \geq \dots \geq p_n,$$

of $1||\sum T_j$ problem is presented

E.L. Lawler, *A fully polynomial approximation scheme for the total tardiness problem*. Oper. Res. Lett. 1 207-208 (1982).

An $O(\frac{n^7}{\epsilon})$ approximation scheme for $1||\sum T_j$ problem is presented

Baptiste, P., *Scheduling equal-length jobs on identical parallel machines*, Discrete Appl. Math., Discrete Applied Mathematics, 103, 2000, 1, 21-32.

The problem $1|r_j; p_j = p|\sum T_j$ is polynomially solvable in $O(n^7)$ operations.

- Set of parameters $\Omega = \{r_1, \dots, r_n, p_1, \dots, p_n, d_1, \dots, d_n\}$ characterizes an instance.
- An instance can be considered as a vector in $3n$ -dimensional space of parameters.

For a particular value of parameter $\omega \in \Omega$ in an instance A we will use upper index: ω^A . The value of an objective function F in an instance A under the schedule π will be denoted by $F^A(\pi)$.

Theorem

Function

$$\rho(A, B) = n \cdot \max_{j \in N} |r_j^A - r_j^B| + n \cdot \sum_{j \in N} |p_j^A - p_j^B| + \sum_{j \in N} |d_j^A - d_j^B|$$

satisfies metric axioms and can be considered as a metric of space of parameteres

Theorem

Function

$$\rho(A, B) = n \cdot \max_{j \in N} |r_j^A - r_j^B| + n \cdot \sum_{j \in N} |p_j^A - p_j^B| + \sum_{j \in N} |d_j^A - d_j^B|$$

satisfies metric axioms and can be considered as a metric of space of parameteres

Lemma

For any instances A, B and schedule π

$$\left| \sum_{j \in N} T_j^A(\pi) - \sum_{j \in N} T_j^B(\pi) \right| \leq \rho(A, B).$$

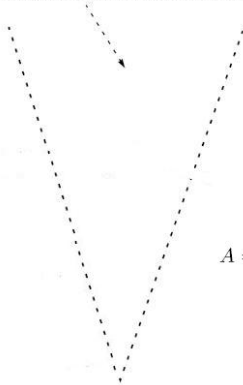
Theorem

For any instances A, B

$$\sum_{j \in N} T_j^A(\pi^B) - \sum_{j \in N} T_j^A(\pi^A) \leq 2\rho(A, B),$$

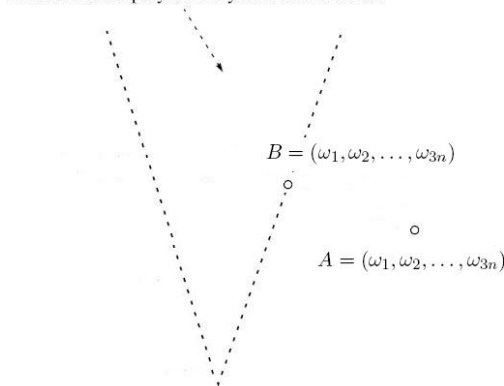
where π^A and π^B are optimal schedules for instances A and B , respectively.

The domain of polynomially solvable instances



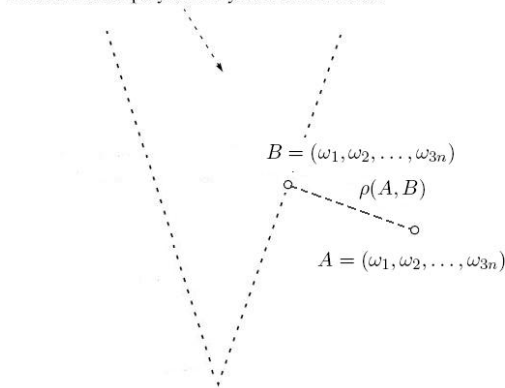
○
 $A = (\omega_1, \omega_2, \dots, \omega_{3n})$

The domain of polynomially solvable instances



- **Step 1.** We find the least distanced in the metric from a given instance A a polynomially (pseudopolynomially) solvable instance B
- **Step 2.** We use known optimal schedule π^B as an approximate solution for instance A

The domain of polynomially solvable instances



- An absolute error of the scheme is not more than $2\rho(A, B)$.
- The problem $1|r_j| \sum T_j$ is reduced to the problem of the function $\rho(A, B)$ minimization.

Polynomially solvable class

Consider a polynomially solvable class defined by the system of linear inequalities

$$\mathcal{A} \cdot R^B + \mathcal{B} \cdot P^B + \mathcal{C} \cdot D^B \leq H,$$

where

$$R^B = (r_1^B, \dots, r_n^B)^T,$$

$$P^B = (p_1^B, \dots, p_n^B)^T,$$

$$D^B = (d_1^B, \dots, d_n^B)^T,$$

$$p_j^B \geq 0, r_j^B \geq 0, j \in N,$$

T is transposition symbol, $\mathcal{A}, \mathcal{B}, \mathcal{C}$ – $m \times n$ matrices, and H – a column of m elements.

Linear programming problem

$$\text{minimize } f = n \cdot (y^r - x^r) + n \cdot \sum_{j=1}^n (y_j^p - x_j^p) + \sum_{j=1}^n (y_j^d - x_j^d),$$

subject to

$$x^r \leq r_j^A - r_j^B \leq y^r,$$

$$x_j^p \leq p_j^A - p_j^B \leq y_j^p,$$

$$x_j^d \leq d_j^A - d_j^B \leq y_j^d,$$

$$r_j^B \geq 0, p_j^B \geq 0, j \in N,$$

$$A \cdot R^B + B \cdot P^B + C \cdot D^B \leq H.$$

It is the problem of the linear programming, with $7n + 2$ variables:

$$r_j^B, p_j^B, d_j^B, x_j^p, y_j^p, x_j^d, y_j^d, x^r, y^r, j = 1, \dots, n.$$

\mathcal{PR} -case

In the first group a polynomially solvable instance B have been searched in class $\{\mathcal{PR} : p_j = p, r_j = r, j \in N\}$.

In this case to minimize distance between A and B one has to minimize the function

$$f(p, r) = +n \cdot \sum_{j=1}^n |p_j^A - p| + n \cdot \max_{j \in N} |r_j^A - r|.$$

Lemma

Function $f(p, r)$ has minimum at point $(p \in \{p_1^A, \dots, p_n^A\}, r = \frac{r_{\max}^A + r_{\min}^A}{2})$, where $r_{\max}^A = \max_{j \in N} r_j^A$, $r_{\min}^A = \min_{j \in N} r_j^A$, therefore minimum can be found in $O(n)$ operations.

\mathcal{PD} -case

In the second group a polynomially solvable instance B have been searched in class $\{\mathcal{PD} : p_j = p, d_j = d, j \in N\}$.

In this case to minimize distance between A and B one has to minimize the function

$$g(p, d) = n \cdot \sum_{j \in N} |p_j^A - p| + \sum_{j \in N} |d_j^A - d|.$$

Lemma

Function $g(p, d)$ has minimum at point

$(p \in \{p_1^A, \dots, p_n^A\}, d \in \{d_1^A, \dots, d_n^A\})$, therefore minimum can be found in $O(n)$ operations.

\mathcal{RD} -case

In the third group a polynomially solvable instance B have been searched in class $\{\mathcal{RD} : r_j = r, d_j = d, j \in N\}$.

In this case to minimize distance between A and B one has to minimize the function

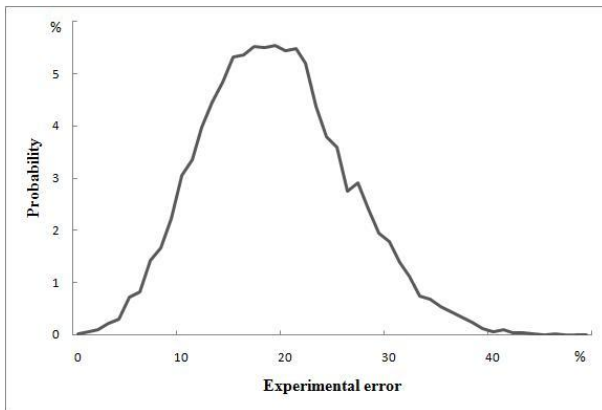
$$h(r, d) = n \cdot \max_{j \in N} |r_j^A - r| + \sum_{j \in N} |d_j^A - d|.$$

Lemma

Function $h(r, d)$ has minimum at point $(r = \frac{r_{\max}^A + r_{\min}^A}{2}, d \in \{d_1^A, \dots, d_n^A\})$, where $r_{\max}^A = \max_{j \in N} r_j^A$, $r_{\min}^A = \min_{j \in N} r_j^A$, therefore minimum can be found in $O(n)$ operations.

- $n = 4, 5, \dots, 10$.
- 10000 instances were generated for each value of n .
- $p_j \in [1, 100]$
- $d_j \in [-100, 100]$
- $r_j \in [0, 100]$.

- $n = 4, 5, \dots, 10$.
 - 10000 instances were generated for each value of n .
 - $p_j \in [1, 100]$
 - $d_j \in [-100, 100]$
 - $r_j \in [0, 100]$.
-
- F_a denotes an approximate objective value of an instance
 - F^* denotes an optimal objective value of an instance.
 - $\delta = F_a - F^*$ is experimental error.



The typical distribution of experimental error.

Table: Average experimental error in percentage of the theoretical error

n	\mathcal{PR} -case	\mathcal{PD} -case	\mathcal{RD} -case
4	19%	4,5%	15%
5	19,5%	6,2%	17,2%
6	19,2%	7,3%	18,4%
7	19,6%	8,5%	19,4%
8	19,3%	9,2%	20,7%
9	19,4%	10%	21,7%
10	19%	10,5%	22,5%

$$1|r_j|L_{max}$$

Lazarev A.A., *Estimation of Absolute Error in Scheduling Problems of Minimizing the Maximum Lateness*, Dokl. Math., Vol. 76, 2007, 572 – 574.
Metric for $1|r_j|L_{max}$ problem is proposed.

$$1|r_j|L_{max}$$

Lazarev A.A., *Estimation of Absolute Error in Scheduling Problems of Minimizing the Maximum Lateness*, Dokl. Math., Vol. 76, 2007, 572 – 574.
Metric for $1|r_j|L_{max}$ problem is proposed.

General case

$$F(\pi) = \sum_{j \in N} \phi_j(\pi, r_1, \dots, r_n, p_1, \dots, p_n, d_j),$$

$$\rho(A, B) = \sum_{j \in N} \sum_{i \in N} (R_{ji}|r_j^A - r_j^B| + P_{ji}|p_j^A - p_j^B|) + \sum_{j \in N} D_j |d_j^A - d_j^B|,$$

where $R_{ji} \geq \left| \frac{\partial \phi_j}{\partial r_i} \right|$, $P_{ji} \geq \left| \frac{\partial \phi_j}{\partial p_i} \right|$, $D_j \geq \left| \frac{\partial \phi_j}{\partial d_j} \right|$.

Polynomially solvable algorithm for a dual of the NP -hard problem

$$1|r_j|\varphi_{max}$$

Minimizing maximum lateness

$1|r_j|L_{\max}$

Single machine, n jobs

r_j – release time;

$p_j > 0$ – processing time;

d_j – due date.

$j \in N = \{1, 2, \dots, n\}$

Minimizing maximum lateness

$1|r_j|L_{\max}$

Single machine, n jobs

r_j – release time;

$p_j > 0$ – processing time;

d_j – due date.

$j \in N = \{1, 2, \dots, n\}$

Preemptions of a job are not allowed. The machine can process at most one job at any time.

Minimizing maximum lateness

$1|r_j|L_{\max}$

Single machine, n jobs

r_j – release time;

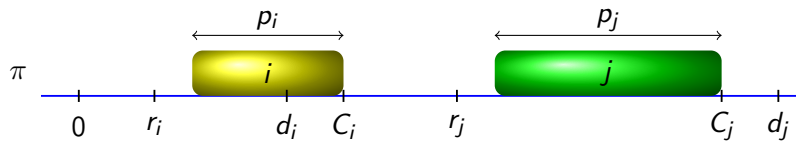
$p_j > 0$ – processing time;

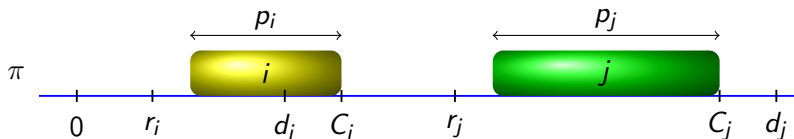
d_j – due date.

$j \in N = \{1, 2, \dots, n\}$

Preemptions of a job are not allowed. The machine can process at most one job at any time.

Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. **1979**



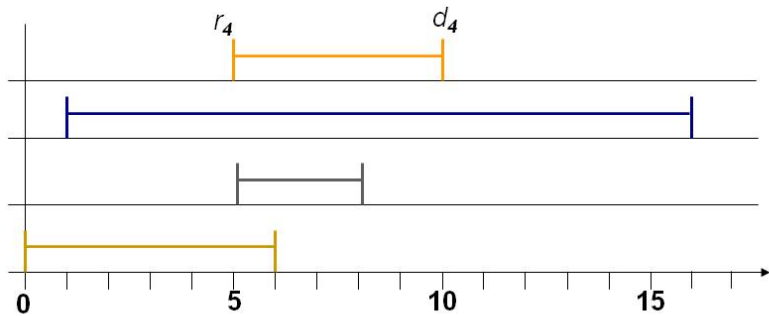


$$F(\pi) = \max_{j \in N} \{C_j - d_j\} \rightarrow \min_{\pi}$$

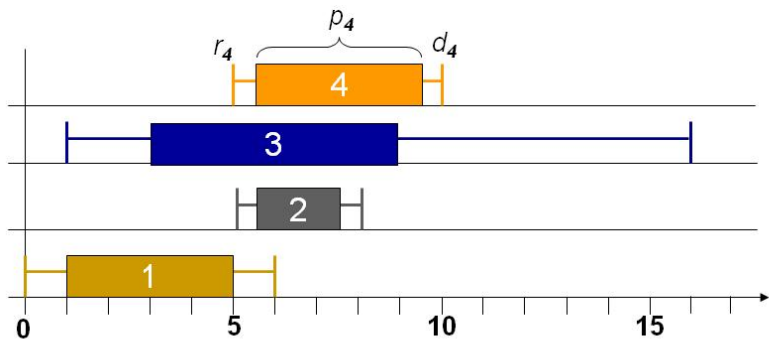
NP-hard in strong sense

Lenstra J.K., Rinnooy Kan A.H.G., Brucker, P. **1977**

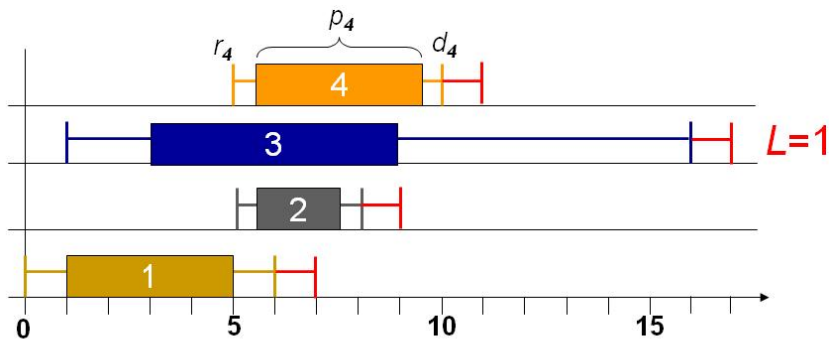
$$1|r_j|L_{\max}$$



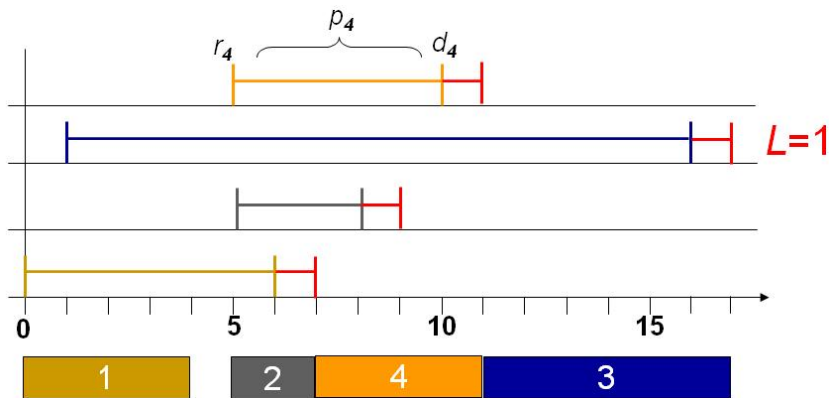
$$1/r_j |L_{\max}$$



$$1/r_j | L_{\max}$$



$$1/r_j | L_{\max}$$



Solvable cases:

1) $r_j = 0, \forall j \in N.$

Jackson J.R. **1955**

$O(n \log n)$

1') $d_j = \text{const}, \forall j \in N.$

$O(n \log n)$

1'') $p_j = \text{const}, \forall j \in N.$

Simons B. **1983.**

$O(n^2 \log n)$

2)

$O(n^3 \log n)$

$$\begin{cases} d_1 \leq d_2 \leq \dots \leq d_n; \\ d_1 - r_1 - p_1 \geq d_2 - r_2 - p_2 \geq \dots \geq d_n - r_n - p_n. \end{cases} \quad (1)$$

2') $d_j = r_j + p_j + \text{const}, \forall j \in N.$

$O(n^3 \log n)$

$\{1, P, Q, R\} | r_j | \{L_{\max}, C_{\max}\}$

$O(n^3 \log n)$

Lazarev A.A., Sadykov R.R., Sevastyanov S.V. 1988-2007

2)

$O(n^3 \log n)$

$$\begin{cases} d_1 \leq d_2 \leq \dots \leq d_n; \\ d_1 - r_1 - p_1 \geq d_2 - r_2 - p_2 \geq \dots \geq d_n - r_n - p_n. \end{cases} \quad (1)$$

2') $d_j = r_j + p_j + \text{const}, \forall j \in N.$

$O(n^3 \log n)$

$\{1, P, Q, R\} | r_j | \{L_{\max}, C_{\max}\}$

$O(n^3 \log n)$

Lazarev A.A., Sadykov R.R., Sevastyanov S.V. **1988-2007**

3) $\max_{k \in N} \{d_k - r_k - p_k\} \leq d_j - r_j, \forall j \in N.$

$O(n^2 \log n)$

Hoogeveen J. A. **1996**

2)

$O(n^3 \log n)$

$$\begin{cases} d_1 \leq d_2 \leq \dots \leq d_n; \\ d_1 - r_1 - p_1 \geq d_2 - r_2 - p_2 \geq \dots \geq d_n - r_n - p_n. \end{cases} \quad (1)$$

2') $d_j = r_j + p_j + \text{const}, \forall j \in N.$

$O(n^3 \log n)$

$\{1, P, Q, R\} | r_j | \{L_{\max}, C_{\max}\}$

$O(n^3 \log n)$

Lazarev A.A., Sadykov R.R., Sevastyanov S.V. 1988-2007

3) $\max_{k \in N} \{d_k - r_k - p_k\} \leq d_j - r_j, \forall j \in N.$

$O(n^2 \log n)$

Hoogeveen J. A. 1996

4) *NP*-hard in ordinary sense

$O(n^2P + np_{\max}P)$

$$\begin{cases} d_1 \leq d_2 \leq \dots \leq d_n; \\ r_1 \geq r_2 \geq \dots \geq r_n; \\ r_j, p_j, d_j \in \mathbb{Z}^+, \forall j \in N. \end{cases} \quad (2)$$

Lazarev A.A., Schulgina O.N. **1998**

$$P = r_{\max} + \sum_{j=1}^n p_j - r_{\min}, \quad r_{\max} = \max_{j \in N} r_j, \quad r_{\min} = \min_{j \in N} r_j, \quad p_{\max} = \max_{j \in N} p_j$$

5)

$$\begin{cases} d_1 \leq d_2 \leq \dots \leq d_n; \\ d_1 - \alpha r_1 - \beta p_1 \geq d_2 - \alpha r_2 - \beta p_2 \geq \dots \geq d_n - \alpha r_n - \beta p_n; \\ \alpha \in [1, \infty), \beta \in [0, 1]. \end{cases} \quad (3)$$

5)

$$\begin{cases} d_1 \leq d_2 \leq \dots \leq d_n; \\ d_1 - \alpha r_1 - \beta p_1 \geq d_2 - \alpha r_2 - \beta p_2 \geq \dots \geq d_n - \alpha r_n - \beta p_n; \\ \alpha \in [1, \infty), \beta \in [0, 1]. \end{cases} \quad (3)$$

5')

$$d_j = \alpha r_j + \beta p_j + \text{const}, \quad \forall j \in N, \alpha \in [1, \infty), \beta \in [0, 1].$$

2009

5)

$$\begin{cases} d_1 \leq d_2 \leq \dots \leq d_n; \\ d_1 - \alpha r_1 - \beta p_1 \geq d_2 - \alpha r_2 - \beta p_2 \geq \dots \geq d_n - \alpha r_n - \beta p_n; \\ \alpha \in [1, \infty), \beta \in [0, 1]. \end{cases} \quad (3)$$

5')

$$d_j = \alpha r_j + \beta p_j + \text{const}, \quad \forall j \in N, \alpha \in [1, \infty), \beta \in [0, 1].$$

2009

$O(n^3 \log n)$

Algorithm 1.

$O(n \log n)$

Step 0) $\omega = \emptyset$; $t = -\infty$;

Step 1) $f := f(N, t)$ and $s := s(N, t)$;

$$f(N, t) = \arg \min_{j \in N} \{d_j \mid r_j(t) = r(N, t)\},$$

$$s(N, t) = \arg \min_{j \in N \setminus \{f\}} \{d_j \mid r_j(t) = r(N \setminus \{f\}, t)\},$$

$$r_j(t) = \max\{r_j, t\}, r(N, t) = \min_{j \in N} \{r_j(t)\}.$$

Step 2) if $d_f \leq d_s$ then

begin

$\omega := (\omega, f)$; $N := N \setminus \{f\}$, $t := r_f(t) + p_f$ and goto Step 1)

end

else RETURN.

Algorithm 2. $\alpha \in [1, \infty), \beta \in [0, 1]$

$O(n^2 \log n)$

$1 \mid d_i \leq d_j, d_i - \alpha r_i - \beta p_i \geq d_j - \alpha r_j - \beta p_j; L_{\max} \leq y \mid C_{\max}$

Step 0) $\theta := \omega(N, t)$; if $L_{\max}(\theta, t) > y$ then $\theta := \emptyset$ and RETURN.

Step 1) $N := N \setminus \{\theta\}$; $t := C_{\max}(\theta)$;

$\omega^1 = (f, \omega(N \setminus \{f\}, r_f(t) + p_f)$; $\omega^2 = (s, \omega(N \setminus \{s\}, r_s(t) + p_s)$;

if $L_{\max}(\omega^1, t) \leq y$ then $\theta := (\theta, \omega^1)$ and goto Step 1);

Step 2) if $L_{\max}(\omega^1, t) > y$ and $L_{\max}(\omega^2, t) \leq y$ then $\theta := (\theta, \omega^2)$ and goto Step 1);

Step 3) if $L_{\max}(\omega^1, t) > y$ and $L_{\max}(\omega^2, t) > y$ then $\theta := \emptyset$ and RETURN.

Algorithm 3. $\alpha \in [1, \infty), \beta \in [0, 1]$

$O(n^3 \log n)$

$1 \mid d_i \leq d_j, d_i - \alpha r_i - \beta p_i \geq d_j - \alpha r_j - \beta p_j \mid L_{\max}$

Step 0) $y := +\infty; \pi^* := \omega(N, t); \Phi := \emptyset; m := 0; N' := N \setminus \{\pi^*\};$
 $t' := C_{\max}(\pi^*);$ if $N' = \emptyset$ then $\Phi := \Phi \cup (\pi^*); m := 1$ and RETURN.

Step 1) if $L_{\max}(\omega^1, t') \leq L_{\max}(\pi^*)$ then $\pi^* := (\pi^*, \omega^1); N' := N \setminus \{\pi^*\};$
 $t' := C_{\max}(\pi^*);$ goto Step 1);

Step 2) if $(L_{\max}(\omega^1, t') > L_{\max}(\pi^*)) \& (L_{\max}(\omega^1, t') < y)$ then
 $\theta := \theta(N', t', y'), y' := L_{\max}(\omega^1, t');$

if $\theta = \emptyset$ then $\pi^* := (\pi^*, \omega^1);$ goto Step 1) else $\pi' := (\pi^*, \theta);$

if $C_{\max}(\pi_m) < C_{\max}(\pi')$ then $m := m + 1; \pi_m := \pi'; \Phi := \Phi \cup (\pi_m);$

$y = L_{\max}(\pi_m)$ else $\pi_m = \pi';$ goto Step 1);

Step 3) if $(L_{\max}(\omega^1, t') \geq y) \& (L_{\max}(\omega^2, t') < y)$ then $\pi^* = (\pi^*, \omega^2);$ goto
Step 1) else $\pi^* = \pi'_m$ and RETURN.

Pareto optimal schedules for

$1 \mid d_i \leq d_j, d_i - \alpha r_i - \beta p_i \geq d_j - \alpha r_j - \beta p_j \mid L_{\max}, C_{\max}$

$$\begin{cases} d_1 \leq d_2 \leq \dots \leq d_n; \\ d_1 - \alpha r_1 - \beta p_1 \geq d_2 - \alpha r_2 - \beta p_2 \geq \dots \geq d_n - \alpha r_n - \beta p_n; \\ \alpha \in [1, \infty), \beta \in [0, 1]. \end{cases} \quad (4)$$

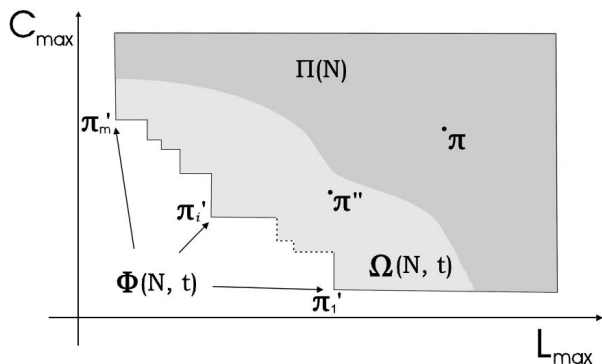
$1 \mid d_i \leq d_j, d_i - \alpha r_i - \beta p_i \geq d_j - \alpha r_j - \beta p_j \mid L_{\max}, C_{\max}$

$$1 \leq \|\Phi(N, t)\| \leq n$$

$O(n^3 \log n)$

Pareto optimal schedules for

$$1 \mid d_i \leq d_j, d_i - \alpha r_i - \beta p_i \geq d_j - \alpha r_j - \beta p_j \mid L_{\max}, C_{\max}$$

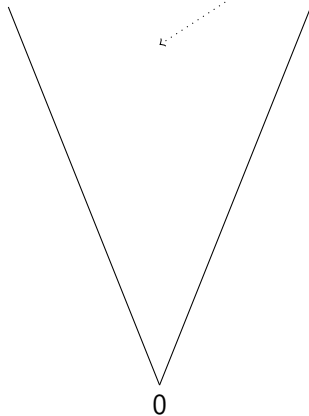


Any instance is point in $m = 3n$ -dimension space.

- A – "hard" instance

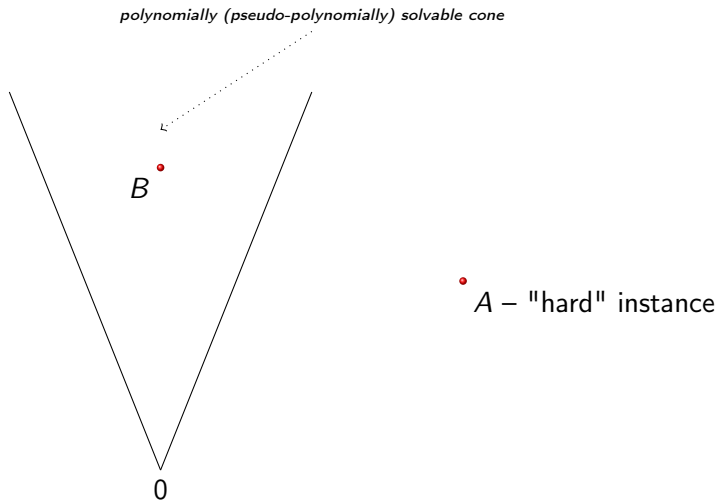
Any instance is point in $m = 3n$ -dimension space.

polynomially (pseudo-polynomially) solvable cone

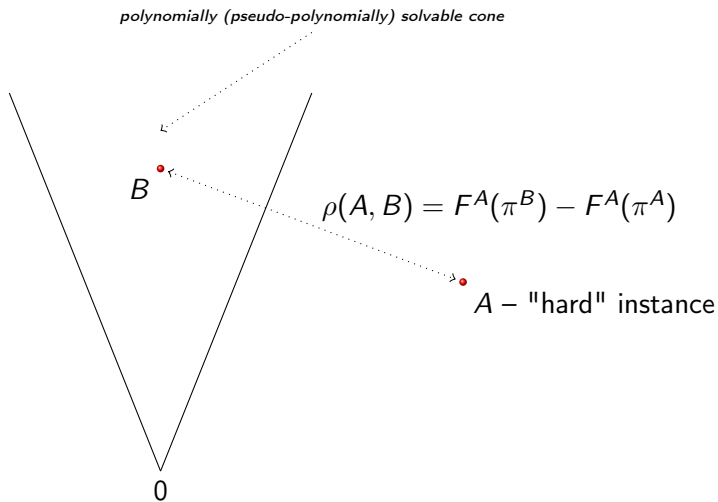


• A – "hard" instance

Any instance is point in $m = 3n$ -dimension space.



Any instance is point in $m = 3n$ -dimension space.



$1/r_j |L_{max}$

$$0 \leq \rho(A, B) = F^A(\pi^B) - F^A(\pi^A) \leq$$
$$(\max\{r_j^A - r_j^B\} - \min\{r_j^A - r_j^B\}) +$$
$$(\sum |p_j^A - p_j^B|) +$$
$$(\max\{d_j^A - d_j^B\} - \min\{d_j^A - d_j^B\})$$

Property of metric

$$\varphi(A) = \max_{j \in N} (r_j^A) - \min_{j \in N} (r_j^A) + \max_{j \in N} (d_j^A) - \min_{j \in N} (d_j^A) + \sum_{j \in N} |p_j^A| \geq 0.$$

$$\begin{cases} \varphi(A) = 0 \iff A \equiv 0; \\ \varphi(\alpha A) = \alpha \varphi(A); \\ \varphi(A + B) \leq \varphi(A) + \varphi(B). \end{cases} \quad (5)$$

$$\|A\| = \varphi(A) \quad \rho(A, B) = \|A - B\|.$$

$$\|A\| = \varphi(A)$$

$$\rho(A, B) = \|A - B\|$$

$$\|A\| = \varphi(A)$$

$$\rho(A, B) = \|A - B\|$$

Polynomially (pseudo-polynomially) solvable case

$$AR + BP + CD \leq \mathcal{H}$$

A, B, C – matrixes, R, P, D, \mathcal{H} – vectors.

Polynomially (pseudo-polynomially) solvable case

$$AR + BP + CD \leq \mathcal{H}$$

A, B, C – matrixes, R, P, D, \mathcal{H} – vectors.

Projection of an instance A to a polynomially (pseudo-polynomially) solvable case

The minimum absolute error among all instances from solvable area, – instance B .

Projection of an instance A to a polynomially (pseudo-polynomially) solvable case

The minimum absolute error among all instances from solvable area, – instance B .

$O(n \log n)$

$$\left\{ \begin{array}{l} \rho(A, B) = (x_r - y_r) + \sum (x_p - y_p) + (x_d - y_d) \rightarrow \min \\ y_r \leq r_j^A - r_j^B \leq x_r, \forall j; \\ -x_p^j \leq p_j^A - p_j^B \leq x_p^j, \forall j, x_p^j \geq 0; \\ y_d \leq d_j^A - d_j^B \leq x_d, \forall j; \\ AR^B + BP^B + CD^B \leq \mathcal{H}. \end{array} \right.$$

Any penalties

Initial problem

$$\mu^* = \min_{\pi \in \Pi(N)} \max_{k=1, n} \varphi_{j_k}(C_{j_k}(\pi)), \quad (6)$$

Not decreasing functions $\varphi_j(C_j(\pi))$

Initial problem

$$\mu^* = \min_{\pi \in \Pi(N)} \max_{k=1, n} \varphi_{j_k}(C_{j_k}(\pi)), \quad (6)$$

Not decreasing functions $\varphi_j(C_j(\pi))$

Dual problem

$$\nu^* = \max_{k=1, n} \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi)). \quad (7)$$

$r_j = 0, \forall j \in N$

Conway R.W., Maxwell W.L., Miller L.W. Theory of Scheduling // Addison-Wesley, Reading, MA. 1967.

$$\nu^* = \max_{k=1, n} \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi))$$

$$\nu^* = \max_{k=1, n} \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi))$$

$$\nu_k = \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi)), k = 1, 2, \dots, n. \quad (8)$$

Dual problem

$$\nu^* = \max_{k=1, n} \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi))$$

$$\nu_k = \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi)), k = 1, 2, \dots, n. \quad (8)$$

$$\nu^* = \max_{k=1, n} \nu_k. \quad (9)$$

Lemma

$\varphi_j(t), j = 1, 2, \dots, n$, any not decreasing functions $1 \mid r_j \mid \varphi_{\max}$,
 $\forall k = 1, 2, \dots, n, \quad \nu_n \geq \nu_k, \quad \nu^* = \nu_n.$

Lemma

$\varphi_j(t), j = 1, 2, \dots, n$, any not decreasing functions $1 \leq r_j \leq \varphi_{\max}$,
 $\forall k = 1, 2, \dots, n, \quad \nu_n \geq \nu_k, \quad \nu^* = \nu_n.$

Algorithm

$\pi^r = (i_1, i_2, \dots, i_n), \quad r_{i_1} \leq r_{i_2} \leq \dots \leq r_{i_n};$
 $\pi_k = (\pi^r \setminus i_k, i_k), k = 1, 2, \dots, n, \quad \varphi_{i_k}(C_{i_k}(\pi_k));$
 $\nu^* = \max_{k=1, n} \varphi_{i_k}(C_{i_k}(\pi_k)).$

Lemma

$\varphi_j(t), j = 1, 2, \dots, n$, any not decreasing functions $1 \leq r_j \leq \varphi_{\max}$,
 $\forall k = 1, 2, \dots, n, \quad \nu_n \geq \nu_k, \quad \nu^* = \nu_n.$

Algorithm

$\pi^r = (i_1, i_2, \dots, i_n), \quad r_{i_1} \leq r_{i_2} \leq \dots \leq r_{i_n};$
 $\pi_k = (\pi^r \setminus i_k, i_k), k = 1, 2, \dots, n, \quad \varphi_{i_k}(C_{i_k}(\pi_k));$
 $\nu^* = \max_{k=1, n} \varphi_{i_k}(C_{i_k}(\pi_k)).$

$O(n^2)$

Initial problem

$$\mu^* = \min_{\pi \in \Pi(N)} \max_{k=1, \dots, n} \varphi_{j_k}(C_{j_k}(\pi)), \quad (10)$$

Not decreasing function $\varphi_j(C_j(\pi))$

Initial problem

$$\mu^* = \min_{\pi \in \Pi(N)} \max_{k=1, n} \varphi_{j_k}(C_{j_k}(\pi)), \quad (10)$$

Not decreasing function $\varphi_j(C_j(\pi))$

Dual problem

$$\nu^* = \max_{k=1, n} \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi)). \quad (11)$$

Initial problem

$$\mu^* = \min_{\pi \in \Pi(N)} \max_{k=1, n} \varphi_{j_k}(C_{j_k}(\pi)), \quad (10)$$

Not decreasing function $\varphi_j(C_j(\pi))$

Dual problem

$$\nu^* = \max_{k=1, n} \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi)). \quad (11)$$

Theorem

$\varphi_j(t), j = 1, 2, \dots, n$, any not decreasing functions $1 \mid r_j \mid \varphi_{\max}$,
 $\forall k = 1, 2, \dots, n, \quad \mu^* \geq \nu^*.$

Initial problem

$$\mu^* = \min_{\pi \in \Pi(N)} \max_{k=1, n} \varphi_{j_k}(C_{j_k}(\pi)), \quad (10)$$

Not decreasing function $\varphi_j(C_j(\pi))$

Dual problem

$$\nu^* = \max_{k=1, n} \min_{\pi \in \Pi(N)} \varphi_{j_k}(C_{j_k}(\pi)). \quad (11)$$

Theorem

$\varphi_j(t), j = 1, 2, \dots, n$, any not decreasing functions $1 \mid r_j \mid \varphi_{\max}$,
 $\forall k = 1, 2, \dots, n, \quad \mu^* \geq \nu^*.$

Branch and bounds

Preceding, Dual problem

G	single machine	$O(n^2)$
G	many machines	<i>NP-hard</i>

Not decreasing penalty functions $\varphi_j(C_j(\pi))$

Scheduling Theory and Applications. Part II

Alexander Lazarev

Lomonosov Moscow State University

National Research University Higher School of Economics

Moscow Institute of Physics and Technology (State University)

V.A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences (ICS RAS)

jobmath@mail.ru

www.orsot.ru



V.A. TRAPEZNIKOV
INSTITUTE
OF CONTROL
SCIENCES
OF RUSSIAN ACADEMY
OF SCIENCES



Thank you for your attention!