Optimization Methods Introduction

Jean-Louis Bouquard



Beijing Institute of Technology

Optim. Meth.

Content

- Introduction
- 2 A scheduling problem
- 3 Local search
- Experimentation

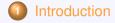
Thanks to Prof. Li Dongni

Jean-Louis Bouquard
Graduate School of Engineering
University of Tours (France)

Introduction Optimization Methods

Thanks to Prof. Li Dongni

Jean-Louis Bouquard Graduate School of Engineering University of Tours (France) jean-louis.bouquard@univ-tours.fr



- Optimization problems
- Find X such that f(X) is minimal subject to C(X)
- f(X) is the Objective function or the Criterion
- defines you are looking for
- \circ $\mathcal{C}(X)$ is the set of constraints on X
- o defines where you are looking for
- Ex: Tiraveling Salesman Problem.
- Ex. Shortest oath in a graph
- Evi Knansank nroble

- Optimization problems
- Find X such that f(X) is minimal subject to C(X)

- Optimization problems
- Find X such that f(X) is minimal subject to C(X)
- f(X) is the Objective function or the Criterion

Optimization problems

- Optimization problems
- Find X such that f(X) is minimal subject to C(X)
- f(X) is the Objective function or the Criterion
- defines what you are looking for
- C(X) is the set of constraints on X
- defines where you are looking for
- Ex Shortest call in a grap

<ロ> < 回 > < 回 > < 巨 > < 巨 > 三 り < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > つ < つ > の < の > の < の > の < の > の < の > の < の > の < の > の < の > の < の > の < の > の < の > の < の > の < の > の < の る < の > の < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る < の る へ の る < の る < の る < の る < の る < の る < の る < の る < の る <

- Optimization problems
- Find X such that f(X) is minimal subject to C(X)
- f(X) is the Objective function or the Criterion
- defines what you are looking for
- C(X) is the set of constraints on X

- Optimization problems
- Find X such that f(X) is minimal subject to C(X)
- f(X) is the Objective function or the Criterion
- defines what you are looking for
- C(X) is the set of constraints on X
- defines where you are looking for
- Ex: Traveling Salesman Problem
- Ex: Shortest path in a graph
- ____

- Optimization problems
- Find X such that f(X) is minimal subject to C(X)
- f(X) is the Objective function or the Criterion
- defines what you are looking for
- C(X) is the set of constraints on X
- defines where you are looking for
- Ex: Traveling Salesman Problem
- Ex: Shortest path in a graph
- Ex: Knapsack problem

- Optimization problems
- Find X such that f(X) is minimal subject to C(X)
- f(X) is the Objective function or the Criterion
- defines what you are looking for
- C(X) is the set of constraints on X
- defines where you are looking for
- Ex: Traveling Salesman Problem
- Ex: Shortest path in a graph

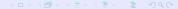
- Optimization problems
- Find X such that f(X) is minimal subject to C(X)
- f(X) is the Objective function or the Criterion
- defines what you are looking for
- C(X) is the set of constraints on X
- defines where you are looking for
- Ex: Traveling Salesman Problem
- Ex: Shortest path in a graph
- Ex: Knapsack problem

Algorithm Complexity

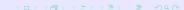
- Complexity of algorithms
- With the Θ and the O notations
- Ex: the complexity of "brute force" sort is $\Theta(n!)$
- the complexity of the merge sort (or heap sort) is $\Theta(n \log n)$
- the complexity of the insertion sort (or selection sort) is Θ(n²)
- Moore-Diisktra algorithm: O(V²)
- Wrating Multiplication with the "usual" algorithm: \(\text{O}(n^2)\)
- Matrix Multip
- TSP with B&B procedure

Algorithm Complexity

- Complexity of algorithms
- With the Θ and the O notations



- Complexity of algorithms
- With the Θ and the O notations
- Ex: the complexity of "brute force" sort is $\Theta(n!)$
- the complexity of the merge sort (or heap sort) is $\Theta(n \log n)$
- the complexity of the insertion sort (or selection sort) is Θ(n²)
- Moore-Diisktra algorithm: O(V²)
- Mains Multiplication with the "usual" algorithm: (6) n²);



- Complexity of algorithms
- With the Θ and the O notations
- Ex: the complexity of "brute force" sort is $\Theta(n!)$
- the complexity of the merge sort (or heap sort) is $\Theta(n \log n)$
- the complexity of the insertion sort (or selection sort) is $\Theta(n^2)$
- Moore-Dijsktra algorithm: O(V²)
- Wairix Wuliuliosation with the "usural" algorithms (O(n²))

△ TSP with R&

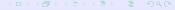
- Complexity of algorithms
- With the Θ and the O notations
- Ex: the complexity of "brute force" sort is $\Theta(n!)$
- the complexity of the merge sort (or heap sort) is $\Theta(n \log n)$
- the complexity of the insertion sort (or selection sort) is Θ(n²)
- Moore-Dijsktra algorithm: O(V²)
- Matrix Multiplication with the "usual" algorithm: $\Theta(n^3)$

- Complexity of algorithms
- With the Θ and the O notations
- Ex: the complexity of "brute force" sort is $\Theta(n!)$
- the complexity of the merge sort (or heap sort) is $\Theta(n \log n)$
- the complexity of the insertion sort (or selection sort) is $\Theta(n^2)$
- Moore-Dijsktra algorithm: O(V²)
- ullet Matrix Multiplication with the "usual" algorithm: $\Theta(\emph{n}^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$

- Complexity of algorithms
- With the Θ and the O notations
- Ex: the complexity of "brute force" sort is $\Theta(n!)$
- the complexity of the merge sort (or heap sort) is $\Theta(n \log n)$
- the complexity of the insertion sort (or selection sort) is $\Theta(n^2)$
- Moore-Dijsktra algorithm: O(V²)
- Matrix Multiplication with the "usual" algorithm: $\Theta(n^3)$



- Complexity of algorithms
- With the Θ and the O notations
- Ex: the complexity of "brute force" sort is $\Theta(n!)$
- the complexity of the merge sort (or heap sort) is $\Theta(n \log n)$
- the complexity of the insertion sort (or selection sort) is Θ(n²)
- Moore-Dijsktra algorithm: O(V²)
- Matrix Multiplication with the "usual" algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: O(n!)



- Complexity of algorithms
- With the Θ and the O notations
- Ex: the complexity of "brute force" sort is $\Theta(n!)$
- the complexity of the merge sort (or heap sort) is $\Theta(n \log n)$
- the complexity of the insertion sort (or selection sort) is Θ(n²)
- Moore-Dijsktra algorithm: O(V²)
- Matrix Multiplication with the "usual" algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: O(n!)



- Complexity of algorithms usually easy to evaluate

- Complexity of algorithms usually easy to evaluate
- for the same problem, several algorithms
- We look for good (i.e. low complexity) algorithms
- Is it possible to design a sort in $\Theta(n)$?

- Complexity of algorithms usually easy to evaluate
- for the same problem, several algorithms
- We look for good (i.e. low complexity) algorithms
- \circ Is it possible to design a sort in $\Theta(n)$?

- Complexity of algorithms usually easy to evaluate
- for the same problem, several algorithms
- We look for good (i.e. low complexity) algorithms
- Is it possible to design a sort in $\Theta(n)$?

- Complexity of problems
- Polynomial problems: there exists a $\Theta(n^k)$ algoritm
- Considered as easy or tractable
- \circ Superpolynomial problems: e.g. in $\Omega(a^n)$ (a>1)
- Considered as hard or untractable
- *NP*-complete problems: $P \neq NP$ question since 1971
- Called NP-hard
- ♦ Any comparison sort algorithm is 0(n loc n)

- Complexity of problems
- Polynomial problems: there exists a $\Theta(n^k)$ algoritm
- Considered as easy or tractable
- Superpolynomial problems: e.g. in $\Omega(a^n)$ (a > 1)
- Considered as hard or u
- *NP*-complete problems: $P \neq NP$ question since 1971
- Called NP-hard
- Any companson sort algorithm is Current

- Complexity of problems
- Polynomial problems: there exists a $\Theta(n^k)$ algoritm
- Considered as easy or tractable

- Complexity of problems
- Polynomial problems: there exists a $\Theta(n^k)$ algoritm
- Considered as easy or tractable
- Superpolynomial problems: e.g. in $\Omega(a^n)$ (a > 1)
- Considered as hard or untractable
- *NP*-complete problems: $P \neq NP$ question since 1971
- A Called NP-hard
- Any compensor sort algorithm is \(\Omega(n)\) algorithm.

- Complexity of problems
- Polynomial problems: there exists a $\Theta(n^k)$ algoritm
- Considered as easy or tractable
- Superpolynomial problems: e.g. in $\Omega(a^n)$ (a > 1)
- Considered as hard or untractable
- *NP*-complete problems: $P \neq NP$ question since 1971
- Called NP-hard
- Any comparison son apprillim is 0/ n/oc n'

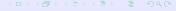
Introduction **Problem Complexity**

Complexity of problems

- Polynomial problems: there exists a $\Theta(n^k)$ algoritm
- Considered as easy or tractable
- Superpolynomial problems: e.g. in $\Omega(a^n)$ (a > 1)
- Considered as hard or untractable
- NP-complete problems: P ≠ NP question since 1971 !!!

Introduction Problem Complexity

- Complexity of problems
- Polynomial problems: there exists a $\Theta(n^k)$ algoritm
- Considered as easy or tractable
- Superpolynomial problems: e.g. in $\Omega(a^n)$ (a > 1)
- Considered as hard or untractable
- NP-complete problems: P ≠ NP question since 1971
 !!!
- Called NP-hard
- Any comparison sort algorithm is $\Omega(n \log n)$



Introduction Problem Complexity

- Complexity of problems
- Polynomial problems: there exists a $\Theta(n^k)$ algoritm
- Considered as easy or tractable
- Superpolynomial problems: e.g. in $\Omega(a^n)$ (a > 1)
- Considered as hard or untractable
- NP-complete problems: P ≠ NP question since 1971
 !!!
- Called NP-hard
- Any comparison sort algorithm is $\Omega(n \log n)$

Problem Complexity

As an effective conclusion:

- For NP-hard problems
- it is not possible to get an optimal solution in a "reasonable" time
- Exact methods used only for small data
- For other (real, realistic) instances, we use
- Approximation methods of
- Heuristics



Introduction

Problem Complexity

- For NP-hard problems
- it is not possible to get an optimal solution in a "reasonable" time
- Exact methods used only for small data
- For other (real, realistic) instances, we use
- Approx
- Heuristics



Introduction

Problem Complexity

- For NP-hard problems
- it is not possible to get an optimal solution in a "reasonable" time
- Exact methods used only for small data
- For other (real, realistic) instances, we use
- Approximation methods or
- Heuristics



Introduction Problem Complexity

- For NP-hard problems
- it is not possible to get an optimal solution in a "reasonable" time
- Exact methods used only for small data
- For other (real, realistic) instances, we use
- Approximation methods or
- Heuristics



Introduction Problem Complexity

- For NP-hard problems
- it is not possible to get an optimal solution in a "reasonable" time
- Exact methods used only for small data
- For other (real, realistic) instances, we use
- Approximation methods or
- Heuristics



Introduction **Problem Complexity**

- For NP-hard problems
- it is not possible to get an optimal solution in a "reasonable" time
- Exact methods used only for small data
- For other (real, realistic) instances, we use
- Approximation methods or
- Heuristics

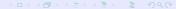




- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is $p_{1,j}$
- ullet then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion time of a job is before its duedate d
- then it is early: T = 0
- ullet else it is tardy: $I_1=C_{2,1}-d_{\parallel}$
- The sum of tardinesses T is to be minimized

11 / 30

- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is p_{1,j}
- ullet then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion time of a job is before its duedate of
 - then it is early: T₁ = 0
- else it is tardy
- The sum of tardinesses T is to be minimized



- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is p_{1,j}
- then on machine 2: processing time is p_{2,j}
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion time of a job is before its duedate a
 - o then it is early: $T_t = 0$
- ase it is tardy:
- The sum of tardinesses T is to be minimized



- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is p_{1,j}
- then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion time of a job is before its duedate d_j
- then it is early: I,
- else it is tardy:
- The sum of tardinesses / is to be minimized



- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is $p_{1,i}$
- then on machine 2: processing time is $p_{2,i}$
- Each machine can process only one job at a time



- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is p_{1,j}
- then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- \circ If the completion time of a job is before its duedate d_j ,
- then it is **early**: $T_j = 0$
- else it is tardy:
- The sum of tardinesses T is to be minim



Optim. Meth.

- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is p_{1,j}
- then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion time of a job is before its duedate d_j,
- then it is early: $T_i = 0$
- ullet else it is tardy: $T_j = C_{2,j} d_j$
- The sum of tardinesses T is to be minimized



- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is p_{1,j}
- then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion time of a job is before its duedate d_j,
- then it is early: $T_j = 0$
- ullet else it is tardy: $T_j = C_{2,j} d_j$
- \bullet The sum of tardinesses \overline{T} is to be minimized



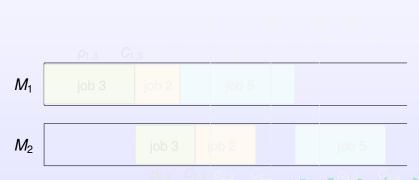
- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is p_{1,j}
- then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion time of a job is before its duedate d_j,
- then it is early: $T_j = 0$
- else it is tardy: $T_j = C_{2,j} d_j$
 - The sum of tardinesses I is to be



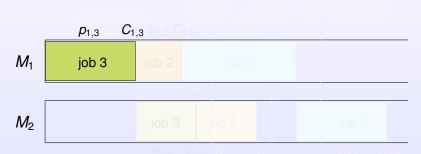
- There are n jobs
- Each job j (1 $\leq j \leq n$) has to be processed
- on machine 1: processing time is p_{1,j}
- then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion time of a job is before its duedate d_j,
- then it is early: $T_j = 0$
- else it is tardy: $T_j = C_{2,j} d_j$
- The sum of tardinesses \overline{T} is to be minimized



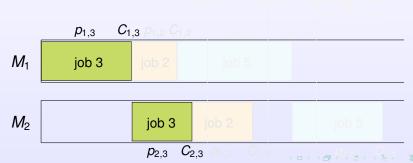
Job 3 early:
$$C_{2,3} \le d_3$$
, $T_3 = 0$
Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$
Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$



Job 3 early:
$$C_{2,3} \le d_3$$
, $T_3 = 0$
Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$
Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$

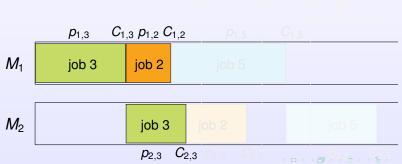


Job 3 early:
$$C_{2,3} \le d_3$$
, $T_3 = 0$
Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$
Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$

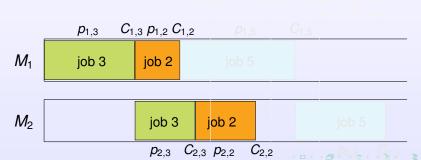


J-L Bouquard

Job 3 early:
$$C_{2,3} \le d_3$$
, $T_3 = 0$
Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$
Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$

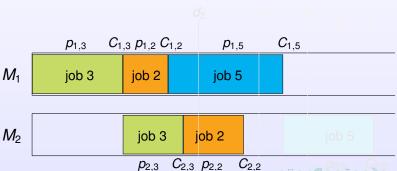


Job 3 early:
$$C_{2,3} \le d_3$$
, $T_3 = 0$
Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$
Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$

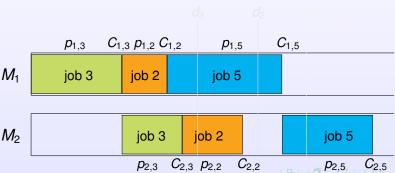


J-L Bouquard

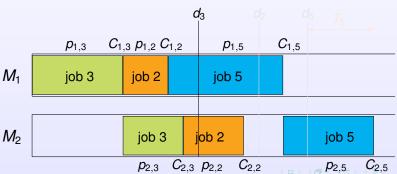
Job 3 early:
$$C_{2,3} \le d_3$$
, $T_3 = 0$
Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$
Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$



Job 3 early:
$$C_{2,3} \le d_3$$
, $T_3 = 0$
Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$
Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$

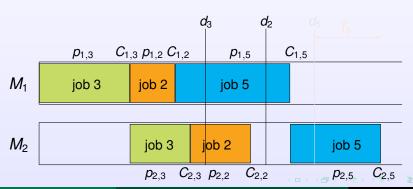


Job 3 early:
$$C_{2,3} \le d_3$$
, $T_3 = 0$
Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$
Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$



Job 3 early: $C_{2,3} \le d_3$, $T_3 = 0$ Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$

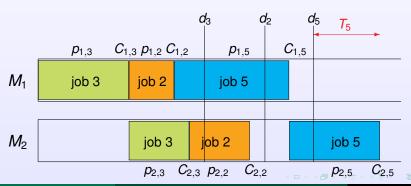
Job 5 tardy: $C_{2.5} > d_5$, $T_5 = C_{2.5} - d_5 > 0$



J-L Bouquard

Job 3 early: $C_{2,3} \le d_3$, $T_3 = 0$ Job 2 early: $C_{2,2} \le d_2$, $T_2 = 0$

Job 5 tardy: $C_{2.5} > d_5$, $T_5 = C_{2.5} - d_5 > 0$



J-L Bouquard

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job ?
- What is the k^m job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1,2,...
- \circ s(1) is the first lob. s(2) is the second lob.
- Do not confuse who still the init your lender to



- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \dots, n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job?
- What is the k^{th} job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1,2.
- \bullet s(1) is the first job. s(2) is the second job.
- Do not confuse pair at 1). The pair year a



- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job ?
- What is the second job?
- What is the k^m job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1, 2
- s(1) is the first job, s(2) is the second job,
- Do not confuse
- a

- The data are: *n* number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job?
- What is the k^{th} job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1, 2.
- s(1) is the first job, s(2) is the second job,

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job?
- What is the k^{th} job ? (for $1 \le k \le n$
- A solution is given by a permutation of {1, 2, ..., n}
- \bullet s(f) is the first job, s(2) is the second job,

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job?
- What is the k^{th} job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1,2,...,n}
- \circ s(1) is the first job, s(2) is the second job, ...

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job?
- What is the k^{th} job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1,2,...,n}
- s(1) is the first job, s(2) is the second job, . . .
- Do not confuse job s(1), the job you decide to schedule first



- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job?
- What is the k^{th} job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1, 2, ..., n}
- s(1) is the first job, s(2) is the second job, ...
- Do not confuse job s(1), the job you decide to schedule first
- and job 1, the job numbered 1 in the data.

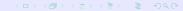


- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job?
- What is the k^{th} job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1,2,...,n}
- s(1) is the first job, s(2) is the second job, ...
- Do not confuse job s(1), the job you decide to schedule first

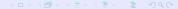
and job 1, the job numbered 1 in the data



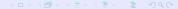
- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, ..., n\}$
- We have to decide how to schedule the jobs
- What is the first job?
- What is the second job?
- What is the k^{th} job ? (for $1 \le k \le n$)
- A solution is given by a permutation of {1,2,...,n}
- s(1) is the first job, s(2) is the second job, ...
- Do not confuse job s(1), the job you decide to schedule first
- and job 1, the job numbered 1 in the data.



- From s, we can compute:
- The completion times
- $C_{1,s(1)} = p_{1,s(1)}$
- $\circ C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $ullet \ C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^{\kappa} p_{1,s(j)}$
- $\circ \ C_{2,s(1)} = C_{1,s(1)} +
 ho_{2,s(1)} =
 ho_{1,s(1)} +
 ho_{2,s(1)}$
- $\circ \ \mathit{C}_{2,s(2)} = \mathsf{max}(\mathit{C}_{1,s(2)},\mathit{C}_{2,s(1)}) + \mathit{p}_{2,s(2)}$
- \bullet $G_{2,s(k)} = \max(G_{1,s(k)}, G_{2,s(k-1)}) + \rho_{2,s(k)}$
 - The tardinesses
- $I_{S(k)} = \max(0, C_{2,S(k)} C_{S(k)})$
- Then $T = \sum_{i=1}^n T_i$



- From s, we can compute:
- The completion times
- $C_{1,s(1)} = p_{1,s(1)}$
- $\quad \bullet \quad C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $ullet \ C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^{\kappa} p_{1,s(j)}$
- $ullet \ C_{2,s(1)} = C_{1,s(1)} +
 ho_{2,s(1)} =
 ho_{1,s(1)} +
 ho_{2,s(1)}$
- $ullet \ C_{2,s(2)} = \max(C_{1,s(2)},C_{2,s(1)}) + p_{2,s(2)}$
- \bullet $G_{2,s(k)} = \max(G_{1,s(k)}, G_{2,s(k-1)}) + p_{2,s(k)} + p_{2,s(k)}$
- The tardinesses
- $T_{\text{sum}} = \max(0, C_{\text{sum}} d_{\text{sum}})$
- Then $\overline{T} = \sum_{i=1}^n T_i$



- From s, we can compute:
- The completion times
- \circ $C_{1,s(1)} = p_{1,s(1)}$
- $C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $ullet \ C_{2,s(1)} = C_{1,s(1)} +
 ho_{2,s(1)} =
 ho_{1,s(1)} +
 ho_{2,s(1)}$
- $\circ C_{2,s(2)} = \max(C_{1,s(2)}, C_{2,s(1)}) + p_{2,s(2)}$
- \bullet $G_{2,s(k)} = \max(G_{1,s(k)}, G_{2,s(k-1)}) + p_{2,s(k)} + p_{2,s(k)}$
 - The tardinesses
- \bullet $T_{sim} = \max(0, C_{sim} c_s)$
- Then $T = \sum_{i=1}^n T_i$

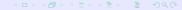
- From s, we can compute:
- The completion times
- $C_{1,s(1)} = p_{1,s(1)}$
- $\bullet \ \ C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $C_{2,s(1)} = C_{1,s(1)} + p_{2,s(1)} = p_{1,s(1)} + p_{2,s(1)}$
- \circ $C_{2,s(2)} = \max(C_{1,s(2)}, C_{2,s(1)}) + p_{2,s(2)}$
- ullet $C_{2,s(k)} = \max(C_{1,s(k)}, C_{2,s(k-1)}) + p_{2,s(k)}$
- The tardinesses
- \bullet $T_{\text{eff}} = \text{max}($
- Then $\overline{T} = \sum_{i=1}^{n} T_i$

- From s, we can compute:
- The completion times
- $C_{1,s(1)} = p_{1,s(1)}$
- $\bullet \ \ C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $C_{2,s(1)} = C_{1,s(1)} + p_{2,s(1)} = p_{1,s(1)} + p_{2,s(1)}$
- $ullet C_{2,s(2)} = \max(C_{1,s(2)},C_{2,s(1)}) + p_{2,s(2)}$
- $C_{2,s(k)} = \max(C_{1,s(k)}, C_{2,s(k-1)}) + p_{2,s(k)}$
- The tardinesses
- $\bullet \ I_{s(k)} = \max(\mathbf{C})$
- Then $T = \sum_{j=1}^n T_j$



- From s, we can compute:
- The completion times
- $C_{1,s(1)} = p_{1,s(1)}$
- $\bullet \ \ C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $\bullet \ \ C_{2,s(1)} = C_{1,s(1)} + p_{2,s(1)} = p_{1,s(1)} + p_{2,s(1)}$
- $\bullet \ C_{2,s(2)} = \max(C_{1,s(2)},C_{2,s(1)}) + p_{2,s(2)}$
- \circ $C_{2,s(k)} = \max(C_{1,s(k)}, C_{2,s(k-1)}) + p_{2,s(k)}$
- The tardinesses
- \bullet $T_{s(k)} = \max(0)$
- \bullet Then $T = \sum_{i=1}^n T_i$

- From s, we can compute:
- The completion times
- $C_{1,s(1)} = p_{1,s(1)}$
- $\bullet \ \ C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $\bullet \ C_{2,s(1)} = C_{1,s(1)} + p_{2,s(1)} = p_{1,s(1)} + p_{2,s(1)}$
- $m{O}_{2,s(2)} = \max(C_{1,s(2)}, C_{2,s(1)}) + p_{2,s(2)}$
- $ullet C_{2,s(k)} = \max(C_{1,s(k)},C_{2,s(k-1)}) + oldsymbol{
 ho}_{2,s(k)}$
- The tardinesses
- \bullet $T_{S(R)} = \Pi$
- Then $\overline{T} = \sum_{i=1}^n T_i$



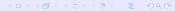
- From s, we can compute:
- The completion times
- \circ $C_{1,s(1)} = p_{1,s(1)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $\bullet \ \ C_{2,s(1)} = C_{1,s(1)} + p_{2,s(1)} = p_{1,s(1)} + p_{2,s(1)}$
- $\bullet \ \ C_{2,s(2)} = \max(C_{1,s(2)},C_{2,s(1)}) + p_{2,s(2)}$
- $\bullet \ C_{2,s(k)} = \max(C_{1,s(k)}, C_{2,s(k-1)}) + p_{2,s(k)}$
- $T_{s(k)} = \max(0, C_{2,s(k)} d_{s(k)})$
- Then $\overline{T} = \sum_{i=1}^n T_i$



- From s, we can compute:
- The completion times
- $C_{1,s(1)} = p_{1,s(1)}$
- $\bullet \ \ C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $\bullet \ \ C_{2,s(1)} = C_{1,s(1)} + p_{2,s(1)} = p_{1,s(1)} + p_{2,s(1)}$
- $m{O}_{2,s(2)} = \max(C_{1,s(2)}, C_{2,s(1)}) + p_{2,s(2)}$
- $\bullet \ \ C_{2,s(k)} = \max(C_{1,s(k)},C_{2,s(k-1)}) + p_{2,s(k)}$
- The tardinesses
- $T_{s(k)} = \max(0, C_{2,s(k)} d_{s(k)})$
- Then $\overline{T} = \sum_{i=1}^{n} T_i$



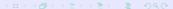
- From s, we can compute:
- The completion times
- $C_{1,s(1)} = p_{1,s(1)}$
- $\bullet \ \ C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $\bullet \ \ C_{2,s(1)} = C_{1,s(1)} + p_{2,s(1)} = p_{1,s(1)} + p_{2,s(1)}$
- $m{O}_{2,s(2)} = \max(C_{1,s(2)}, C_{2,s(1)}) + p_{2,s(2)}$
- $ullet C_{2,s(k)} = \max(C_{1,s(k)},C_{2,s(k-1)}) + p_{2,s(k)}$
- The tardinesses
- $T_{s(k)} = \max(0, C_{2,s(k)} d_{s(k)})$
- Then $T = \nabla'' \cdot T$



- From s, we can compute:
- The completion times
- \circ $C_{1,s(1)} = p_{1,s(1)}$
- $\bullet \ \ C_{1,s(2)} = C_{1,s(1)} + p_{1,s(2)} = p_{1,s(1)} + p_{1,s(2)}$
- $C_{1,s(k)} = C_{1,s(k-1)} + p_{1,s(k)} = \sum_{j=1}^k p_{1,s(j)}$
- $\bullet \ \ C_{2,s(1)} = C_{1,s(1)} + p_{2,s(1)} = p_{1,s(1)} + p_{2,s(1)}$
- $m{O}_{2,s(2)} = \max(C_{1,s(2)},C_{2,s(1)}) + p_{2,s(2)}$
- $\bullet \ C_{2,s(k)} = \max(C_{1,s(k)}, C_{2,s(k-1)}) + p_{2,s(k)}$
- The tardinesses
- $T_{s(k)} = \max(0, C_{2,s(k)} d_{s(k)})$
- Then $\overline{T} = \sum_{j=1}^{n} T_j$







- Two families of algorithms for optimization problems:
- Constructive methods: solutions are computed from the data
- Iterative improvement algorithms
- Starting from one (or several) solution(s),
- better solution(s) is (are) computed
- iteratively, until no improvement is done
- or until a time limit is reached

- Two families of algorithms for optimization problems:
- Constructive methods: solutions are computed from the data
- Iterative improvement algorithms:
- Starting from one (or several) solution(s).
- better solution(s) is (are) computed
- iteratively, until no improvement is done
- o or until a time limit is re-

- Two families of algorithms for optimization problems:
- Constructive methods: solutions are computed from the data
- Iterative improvement algorithms:
- Starting from one (or several) solution(s).
- better solution(s) is (are) computed
- ilteratively, until no improvement is done
- or until a time limit is read

- Two families of algorithms for optimization problems:
- Constructive methods: solutions are computed from the data
- Iterative improvement algorithms:
- Starting from one (or several) solution(s),
- better solution(s) is (are) computed
- iteratively, until no improvement is done
- or until a time limit is reached

- Two families of algorithms for optimization problems:
- Constructive methods: solutions are computed from the data
- Iterative improvement algorithms:
- Starting from one (or several) solution(s),
- better solution(s) is (are) computed
- iteratively, until no improvement is done
- or until a time limit is reached

- Two families of algorithms for optimization problems:
- Constructive methods: solutions are computed from the data
- Iterative improvement algorithms:
- Starting from one (or several) solution(s),
- better solution(s) is (are) computed
- iteratively, until no improvement is done
- or until a time limit is reached

- Two families of algorithms for optimization problems:
- Constructive methods: solutions are computed from the data
- Iterative improvement algorithms:
- Starting from one (or several) solution(s),
- better solution(s) is (are) computed
- iteratively, until no improvement is done
- or until a time limit is reached.

- Local Search is an Iterative improvement algorithm
- From a solution, "slight modifications" are tried with the hope it will improve it
- These modifications are called moves
- The solutions obtained are called neighbors
- They form a neighborhood of the current solution.

- Local Search is an Iterative improvement algorithm
- From a solution, "slight modifications" are tried with the hope it will improve it
- These modifications are called moves
- The solutions obtained are called neighbors
- They form a neighborhood of the current solution

- Local Search is an Iterative improvement algorithm
- From a solution, "slight modifications" are tried with the hope it will improve it
- These modifications are called moves
- The solutions obtained are called neighbors
- They form a neighborhood of the current solution

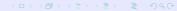
- Local Search is an Iterative improvement algorithm
- From a solution, "slight modifications" are tried with the hope it will improve it
- These modifications are called moves
- The solutions obtained are called neighbors
- They form a neighborhood of the current solution

17/30

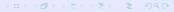
- Local Search is an Iterative improvement algorithm
- From a solution, "slight modifications" are tried with the hope it will improve it
- These modifications are called moves
- The solutions obtained are called neighbors
- They form a neighborhood of the current solution

17/30

- Here a solution is a permutation of {1, 2, ..., n}



- Here a solution is a permutation of $\{1, 2, ..., n\}$
- Move 1: swap the consecutive jobs s(k) and s(k + 1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k + 1$, t(j) = s(j)
- For k = 1, . . . , n −
- The neighborhood has (n-1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- $s(k_0) = s(k_0), t(k_0) = s(k_0)$
- For all j ≠ k, and j ≠ k, i(i) = s(j)
- 0



- Here a solution is a permutation of {1,2,...,n}
- Move 1: swap the consecutive jobs s(k) and s(k + 1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k + 1$, t(j) = s(j)
- For k = 1, ..., n-1
- \circ The neighborhood has (n-1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- $ot(k_1) = s(k_0), t(k_0) = s(k_1)$
- For all $i \neq k_0$ and $j \neq k_0$, t(i) = s(j)

- •



- Here a solution is a permutation of {1,2,...,n}
- Move 1: swap the consecutive jobs s(k) and s(k+1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k+1$, t(j) = s(j)

- Here a solution is a permutation of $\{1, 2, ..., n\}$
- Move 1: swap the consecutive jobs s(k) and s(k + 1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k + 1$, t(j) = s(j)
- For k = 1, ..., n-1
- The neighborhood has (n-1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- $o(t(k_0) = s(k_0) \cdot t(k_0) = s(k_0)$
- For all $j \neq k_1$ and $j \neq k_2$, $i(j) = \alpha$

- Here a solution is a permutation of {1, 2, ..., n}
- Move 1: swap the consecutive jobs s(k) and s(k + 1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k + 1$, t(j) = s(j)
- For k = 1, ..., n-1
- The neighborhood has (n-1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- \bullet $t(k_1) = s(k_2), t(k_2) = s(k_1)$
- For all $j \neq k$, and $j \neq k_2$, t(j) = 1

- Here a solution is a permutation of {1,2,...,n}
- Move 1: swap the consecutive jobs s(k) and s(k + 1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k + 1$, t(j) = s(j)
- For k = 1, ..., n-1
- The neighborhood has (n-1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- $t(k_1) = s(k_2), t(k_2) = s(k_1)$
- For all $j \neq k_1$ and $j \neq k_2$, t(j) = s(j)



- Here a solution is a permutation of {1,2,...,n}
- Move 1: swap the consecutive jobs s(k) and s(k + 1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k + 1$, t(j) = s(j)
- For k = 1, ..., n-1
- The neighborhood has (n − 1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- $t(k_1) = s(k_2), t(k_2) = s(k_1)$
- For all $j \neq k_1$ and $j \neq k_2$, t(j) = s(j)
- For $1 \le k_1 < k_2 \le n$





- Here a solution is a permutation of {1,2,...,n}
- Move 1: swap the consecutive jobs s(k) and s(k+1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k + 1$, t(j) = s(j)
- For k = 1, ..., n-1
- The neighborhood has (n-1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- $t(k_1) = s(k_2), t(k_2) = s(k_1)$
- For all $j \neq k_1$ and $j \neq k_2$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ element



- Here a solution is a permutation of {1, 2, ..., n}
- Move 1: swap the consecutive jobs s(k) and s(k+1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $i \neq k$ and $i \neq k+1$, t(i) = s(i)
- For k = 1, ..., n-1
- The neighborhood has (n-1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- \bullet $t(k_1) = s(k_2), t(k_2) = s(k_1)$
- For all $j \neq k_1$ and $j \neq k_2$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$

- Here a solution is a permutation of {1,2,...,n}
- Move 1: swap the consecutive jobs s(k) and s(k + 1)
- t(k) = s(k+1), t(k+1) = s(k)
- For all $j \neq k$ and $j \neq k + 1$, t(j) = s(j)
- For k = 1, ..., n-1
- The neighborhood has (n − 1) elements
- Move 2: swap the jobs $s(k_1)$ and $s(k_2)$
- $t(k_1) = s(k_2), t(k_2) = s(k_1)$
- For all $j \neq k_1$ and $j \neq k_2$, t(j) = s(j)
- For $1 \le k_1 < k_2 \le n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements



- Move 3: rotate the jobs $s(k_1)$, $s(k_2)$ and $s(k_3)$
- $t_1(k_1) = s(k_2), t_1(k_2) = s(k_3), t_1(k_3) = s(k_1)$
- For all $j \notin \{k_1, k_2, k_3\}, t_1(j) = s(j)$
- \bullet $t_2(k_1) = s(k_3), t_2(k_2) = s(k_1), t_2(k_3) = s(k_2)$
- For all $j \notin \{k_1, k_2, k_3\}, t_2(j) = s(j)$
- \circ 2 neighbors for the same (k_1, k_2, k_3)
- For 1 ≤ k₁ < k₅ < k₃ ≤ n
- The maghinishment has 2 | 1 | 200 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100

- Move 3: rotate the jobs $s(k_1)$, $s(k_2)$ and $s(k_3)$
- $t_1(k_1) = s(k_2), t_1(k_2) = s(k_3), t_1(k_3) = s(k_1)$
- For all $j \notin \{k_1, k_2, k_3\}, t_1(j) = s(j)$
- $t_2(k_1) = s(k_3), t_2(k_2) = s(k_1), t_2(k_3) = s(k_2)$
- For all $j \notin \{k_1, k_2, k_3\}, t_2(j) = s(j)$
- \circ 2 neighbors for the same (k_1, k_2, k_3)
- \circ For $1 \le k_1 < k_2 < k_3 \le$
- The manifestation has a first

- Move 3: rotate the jobs $s(k_1)$, $s(k_2)$ and $s(k_3)$
- $t_1(k_1) = s(k_2), t_1(k_2) = s(k_3), t_1(k_3) = s(k_1)$
- For all $j \notin \{k_1, k_2, k_3\}, t_1(j) = s(j)$
- $t_2(k_1) = s(k_3), t_2(k_2) = s(k_1), t_2(k_3) = s(k_2)$
- For all $j \notin \{k_1, k_2, k_3\}, t_2(j) = s(j)$
- \circ 2 neighbors for the same $(K_1, K_2, K_1, K_2, K_2, K_3)$
- The neighborhood h

<ロ> < @ > < @ > < き > しき ● 9 < @ >

- Move 3: rotate the jobs $s(k_1)$, $s(k_2)$ and $s(k_3)$
- $t_1(k_1) = s(k_2), t_1(k_2) = s(k_3), t_1(k_3) = s(k_1)$
- For all $j \notin \{k_1, k_2, k_3\}, t_1(j) = s(j)$
- $t_2(k_1) = s(k_3), t_2(k_2) = s(k_1), t_2(k_3) = s(k_2)$
- For all $j \notin \{k_1, k_2, k_3\}, t_2(j) = s(j)$
- 2 neighbors for the same (k_1, k_2, k_3)

- Move 3: rotate the jobs $s(k_1)$, $s(k_2)$ and $s(k_3)$
- $t_1(k_1) = s(k_2), t_1(k_2) = s(k_3), t_1(k_3) = s(k_1)$
- For all $j \notin \{k_1, k_2, k_3\}, t_1(j) = s(j)$
- $t_2(k_1) = s(k_3), t_2(k_2) = s(k_1), t_2(k_3) = s(k_2)$
- For all $j \notin \{k_1, k_2, k_3\}, t_2(j) = s(j)$
- 2 neighbors for the same (k_1, k_2, k_3)
- For $1 \le k_1 < k_2 < k_3 \le n$



- Move 3: rotate the jobs $s(k_1)$, $s(k_2)$ and $s(k_3)$
- $t_1(k_1) = s(k_2), t_1(k_2) = s(k_3), t_1(k_3) = s(k_1)$
- For all $j \notin \{k_1, k_2, k_3\}, t_1(j) = s(j)$
- $t_2(k_1) = s(k_3), t_2(k_2) = s(k_1), t_2(k_3) = s(k_2)$
- For all $j \notin \{k_1, k_2, k_3\}, t_2(j) = s(j)$
- 2 neighbors for the same (k_1, k_2, k_3)
- For $1 < k_1 < k_2 < k_3 < n$
- The neighborhood has $2\binom{n}{3} = \frac{n(n-1)(n-2)}{3}$ elements



- Move 3: rotate the jobs $s(k_1)$, $s(k_2)$ and $s(k_3)$
- $t_1(k_1) = s(k_2), t_1(k_2) = s(k_3), t_1(k_3) = s(k_1)$
- For all $j \notin \{k_1, k_2, k_3\}, t_1(j) = s(j)$
- $t_2(k_1) = s(k_3), t_2(k_2) = s(k_1), t_2(k_3) = s(k_2)$
- For all $j \notin \{k_1, k_2, k_3\}, t_2(j) = s(j)$
- 2 neighbors for the same (k_1, k_2, k_3)
- For $1 \le k_1 < k_2 < k_3 \le n$

• The neighborhood has $2\binom{n}{3} = \frac{n(n-1)(n-2)}{3}$ elements



- Move 3: rotate the jobs $s(k_1)$, $s(k_2)$ and $s(k_3)$
- $t_1(k_1) = s(k_2), t_1(k_2) = s(k_3), t_1(k_3) = s(k_1)$
- For all $j \notin \{k_1, k_2, k_3\}, t_1(j) = s(j)$
- $t_2(k_1) = s(k_3), t_2(k_2) = s(k_1), t_2(k_3) = s(k_2)$
- For all $j \notin \{k_1, k_2, k_3\}, t_2(j) = s(j)$
- 2 neighbors for the same (k_1, k_2, k_3)
- For $1 \le k_1 < k_2 < k_3 \le n$
- The neighborhood has $2\binom{n}{3} = \frac{n(n-1)(n-2)}{3}$ elements



- Move 4: shift backward the job $s(k_2)$ in position k_1
- $t(k_1) = s(k_2)$
- For all j, $(k_1 + 1 \le j \le k_2)$, t(j) = s(j-1)
- ▶ For all j, $(1 \le j \le k_1 1)$ and $(k_2 + 1 \le j \le n)$, t(i) = s(i)
- For $1 < k_1 < k_2 < r_1$
- The neighborhood has $\binom{n}{n} = \binom{n-1}{n}$ elements

- Move 4: shift backward the job $s(k_2)$ in position k_1
- $t(k_1) = s(k_2)$
- For all j, $(k_1 + 1 \le j \le k_2)$, t(j) = s(j 1)
- For all j, $(1 \le j \le k_1 1)$ and $(k_2 + 1 \le j \le n)$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$
- The neighborhood has $\binom{n}{s} = \binom{n-1}{s}$ elements

- Move 4: shift backward the job $s(k_2)$ in position k_1
- $t(k_1) = s(k_2)$
- For all j, $(k_1 + 1 \le j \le k_2)$, t(j) = s(j 1)
- For all j, $(1 \le j \le k_1 1)$ and $(k_2 + 1 \le j \le n)$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$
- The neighborhood has $\binom{n}{n} = \binom{n-1}{n}$ elements

- Move 4: shift backward the job $s(k_2)$ in position k_1
- $\bullet t(k_1) = s(k_2)$
- For all j, $(k_1 + 1 \le j \le k_2)$, t(j) = s(j-1)
- For all j, $(1 \le j \le k_1 1)$ and $(k_2 + 1 \le j \le n)$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements



Optim. Meth.

- Move 4: shift backward the job $s(k_2)$ in position k_1
- $t(k_1) = s(k_2)$
- For all j, $(k_1 + 1 \le j \le k_2)$, t(j) = s(j-1)
- For all j, $(1 \le j \le k_1 1)$ and $(k_2 + 1 \le j \le n)$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

- Move 4: shift backward the job $s(k_2)$ in position k_1
- $t(k_1) = s(k_2)$
- For all j, $(k_1 + 1 \le j \le k_2)$, t(j) = s(j-1)
- For all j, $(1 \le j \le k_1 1)$ and $(k_2 + 1 \le j \le n)$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

- Move 5: shift forward the job $s(k_1)$ in position k_2
- $t(k_2) = s(k_1)$
- For all j, $k_1 \le j \le k_2 1$, t(j) = s(j + 1)
- For all j, $1 \le j \le k_1 1$ and $k_2 + 1 \le j \le n$, t(j) = s(j)
- For $1 \le k_1 < k_2 \le n$
- ullet The neighborhood has $\left(egin{array}{c} n\\ 2 \end{array}
 ight)=rac{n(n-1)}{2}$ elements



- Move 5: shift forward the job $s(k_1)$ in position k_2
- $t(k_2) = s(k_1)$
- For all j, $k_1 \le j \le k_2 1$, t(j) = s(j + 1)
- For all j, $1 \le j \le k_1 1$ and $k_2 + 1 \le j \le n$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

- Move 5: shift forward the job $s(k_1)$ in position k_2
- $t(k_2) = s(k_1)$
- For all j, $k_1 \le j \le k_2 1$, t(j) = s(j + 1)
- For all j, $1 \le j \le k_1 1$ and $k_2 + 1 \le j \le n$, t(j) = s(j)
- For $1 \le k_1 < k_2 \le n$
- The neighborhood has $\binom{n}{n} = \frac{n(n-1)}{n}$ elements

- Move 5: shift forward the job $s(k_1)$ in position k_2
- $t(k_2) = s(k_1)$
- For all j, $k_1 \le j \le k_2 1$, t(j) = s(j + 1)
- For all j, $1 \le j \le k_1 1$ and $k_2 + 1 \le j \le n$, t(j) = s(j)
- For $1 < k_1 < k_2 < n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements



- Move 5: shift forward the job $s(k_1)$ in position k_2
- $t(k_2) = s(k_1)$
- For all j, $k_1 \le j \le k_2 1$, t(j) = s(j + 1)
- For all j, $1 \le j \le k_1 1$ and $k_2 + 1 \le j \le n$, t(j) = s(j)
- For $1 \le k_1 < k_2 \le n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements



- Move 5: shift forward the job $s(k_1)$ in position k_2
- $t(k_2) = s(k_1)$
- For all j, $k_1 \le j \le k_2 1$, t(j) = s(j + 1)
- For all j, $1 \le j \le k_1 1$ and $k_2 + 1 \le j \le n$, t(j) = s(j)
- For $1 \le k_1 < k_2 \le n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements



- These moves can be used with every problem the solution is a permutation
- Move 1: consecutive swap k and k + 1
- Move 2: any swap k_1 and k_2 : 2-opt
- Move 3: rotation (k_1, k_2, k_3) :
- Move 4: Extraction and Backward Shift Reinsertion:
- Nove & Extraction and Forward Shift Rensention

 FFSR

- These moves can be used with every problem the solution is a permutation
- Move 1: consecutive swap k and k + 1
- Move 2: any swap k_1 and k_2 : 2-opt
- Move 3: rotation (k₁, k₂, k₃): 3-opt
- Move 4: Extraction and Backward Shift Reinsertion: ERSR
- Wove 5: Extraction and Forward Shift Remsention

 FFSR

- These moves can be used with every problem the solution is a permutation
- Move 1: consecutive swap k and k + 1
- Move 2: any swap k_1 and k_2 : 2-opt
- Move 3: rotation (k_1, k_2, k_3) : 3-opt
- Move 4: Extraction and Backward Shift Reinsertion: EBSR
- Move St. Extraction and Forward Shift Rensentant:

 FFSR

- These moves can be used with every problem the solution is a permutation
- Move 1: consecutive swap k and k + 1
- Move 2: any swap k_1 and k_2 : 2-opt
- Move 3: rotation (k₁, k₂, k₃): 3-opt
- Move 4: Extraction and Backward Shift Reinsertion: EBSR
- Move 5: Extraction and Forward Shift Reinsertion:

- These moves can be used with every problem the solution is a permutation
- Move 1: consecutive swap k and k + 1
- Move 2: any swap k_1 and k_2 : 2-opt
- Move 3: rotation (k₁, k₂, k₃): 3-opt
- Move 4: Extraction and Backward Shift Reinsertion: EBSR
- Move 5: Extraction and Forward Shift Reinsertion:

- These moves can be used with every problem the solution is a permutation
- Move 1: consecutive swap k and k + 1
- Move 2: any swap k_1 and k_2 : 2-opt
- Move 3: rotation (k₁, k₂, k₃): 3-opt
- Move 4: Extraction and Backward Shift Reinsertion: EBSR
- Move 5: Extraction and Forward Shift Reinsertion: EFSR

Algorithm of exploring a neighborhood

Neighborhood

```
function Neighbor(s)
```

Require: s is a solution of the problem

Ensure: The neighborhood of *s* is explored and *bestneighbor* is returned. A boolean value *Improved* is also returned.

- 1: $currentvalue \leftarrow f(s)$
- 2: bestvalue ← currentvalue
- 3: $bestneighbor \leftarrow s$
- 4: **for** *t* **in** neighborhood(*s*) **do**
- 5: **if** (f(t) < bestvalue) **then**
- 6: bestvalue $\leftarrow f(t)$
- 7: $bestneighbor \leftarrow t$
- 8: end if
- 9: end for
- 10: *Improved* ← (*bestvalue* < *currentvalue*)
- 11: return (Improved, bestneighbor)

Algorithm of the Iterated Local Search

Iterated Local Search

function IteratedLS(s)

Require: s is a solution of the problem

Ensure: The neighborhood of *s* is explored as long as an improvement is proved. Then the current best solution is returned.

1: Improved ← true

2: while (Improved) do

3: $(Improved, bestneighbor) \leftarrow Neighbor(s)$

4: end while

5: return bestneighbor