Optimization Methods
Introduction

Jean-Louis Bouquard

Beijing Institute of Technology

Content

Thanks to Prof. Li Dongni

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

Thanks to Prof. Li Dongni

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

1 Introduction

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction
Optimization problems

- Optimization problems
- Find X such that $f(X)$ is minimal subject to $\mathcal{C}(X)$
- $f(X)$ is the Objective function or the Criterion
- defines what you are looking for
- $\mathcal{C}(X)$ 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

Introduction

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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the “usual” algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction
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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the “usual” algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction

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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the “usual” algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction
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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the “usual” algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction
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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the “usual” algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction

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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the “usual” algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction

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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the "usual" algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction
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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the “usual” algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction
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 $\Theta(n^2)$
- Moore-Dijsktra algorithm: $O(V^2)$
- Matrix Multiplication with the “usual” algorithm: $\Theta(n^3)$
- Matrix Multiplication with the Strassen algorithm: $\Theta(n^{\lg 7})$
- TSP with B&B procedure: $O(n!)$

Introduction
Algorithm Complexity

- 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 $\Theta(n)$?

Introduction

Algorithm Complexity

- 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 $\Theta(n)$?

Introduction
Algorithm Complexity

- 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 $\Theta(n)$?

Introduction

Algorithm Complexity

- 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 $\Theta(n)$?

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: $P \neq NP$ 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: $P \stackrel{?}{\neq} NP$ 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: $P \neq NP$ 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: $P \neq NP$ 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: $P \neq NP$ 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: $P \neq NP$ 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: $P \neq NP$ 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: $P \neq NP$ question since 1971 !!!
- Called *NP*-hard
- Any comparison sort algorithm is $\Omega(n \log n)$

Introduction
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 or
- Heuristics

Introduction
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 or
- Heuristics

Introduction

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 or
- Heuristics

Introduction

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 or
- Heuristics

Introduction
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 or
- Heuristics

Introduction
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 or
- Heuristics

2 A scheduling problem

A two-machine flowshop problem

- 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

A two-machine flowshop problem

- 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

A two-machine flowshop problem

- 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

A two-machine flowshop problem

- 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

A two-machine flowshop problem

- 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

A two-machine flowshop problem

- 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

A two-machine flowshop problem

- 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

A two-machine flowshop problem

- 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

A two-machine flowshop problem

- 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

A two-machine flowshop problem

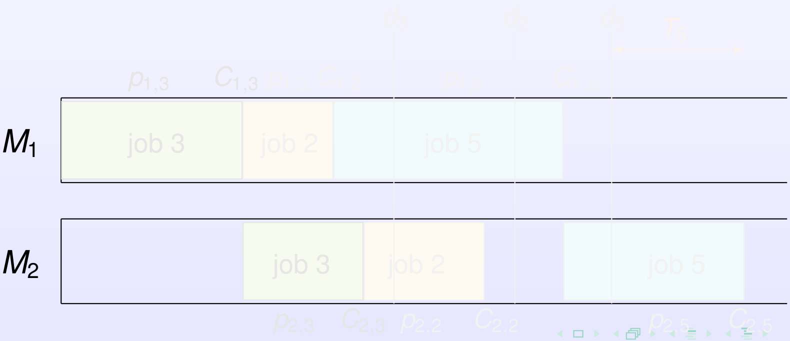
- There are n jobs
- Each job j $(1 \leq j \leq n)$ has to be processed
- on machine 1: processing time is $p_{1,j}$
- then on machine 2: processing time is $p_{2,j}$
- Each machine can process only one job at a time
- For each job, operation 2 cannot begin before operation 1 is completed
- If the completion 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

A two-machine flowshop problem: $F2//\overline{T}$

Job 3 early: $C_{2,3} \leq d_3$, $T_3 = 0$

Job 2 early: $C_{2,2} \leq d_2$, $T_2 = 0$

Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$

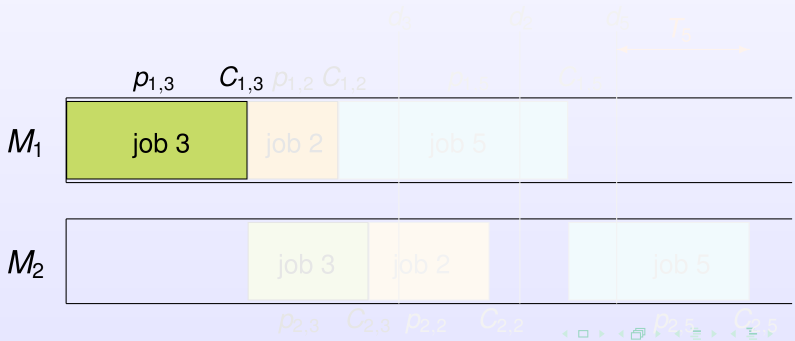


A two-machine flowshop problem: $F2//\overline{T}$

Job 3 early: $C_{2,3} \leq d_3$, $T_3 = 0$

Job 2 early: $C_{2,2} \leq d_2$, $T_2 = 0$

Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$

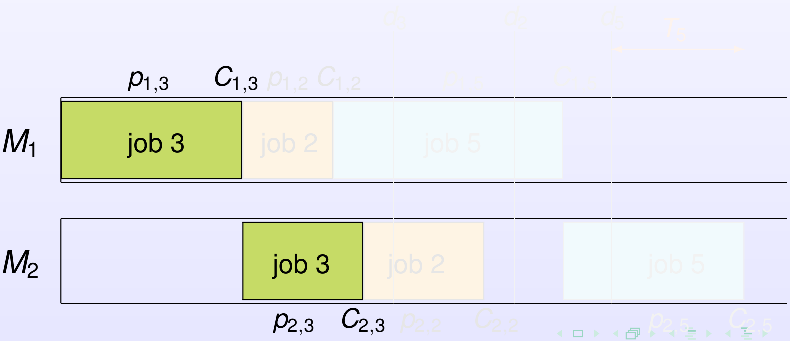


A two-machine flowshop problem: $F2//\overline{T}$

Job 3 early: $C_{2,3} \leq d_3$, $T_3 = 0$

Job 2 early: $C_{2,2} \leq d_2$, $T_2 = 0$

Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$

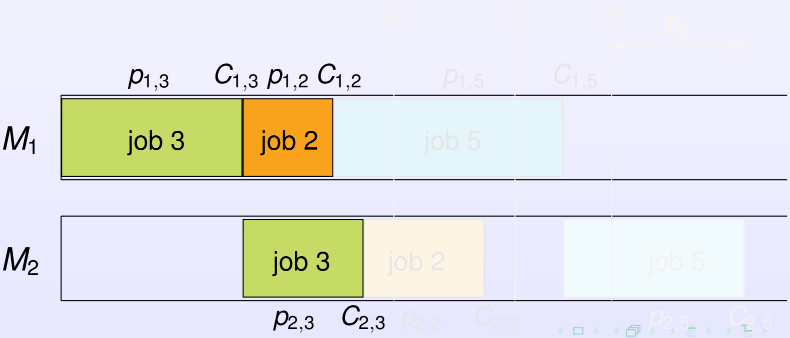


A two-machine flowshop problem: $F2//\overline{T}$

Job 3 early: $C_{2,3} \leq d_3$, $T_3 = 0$

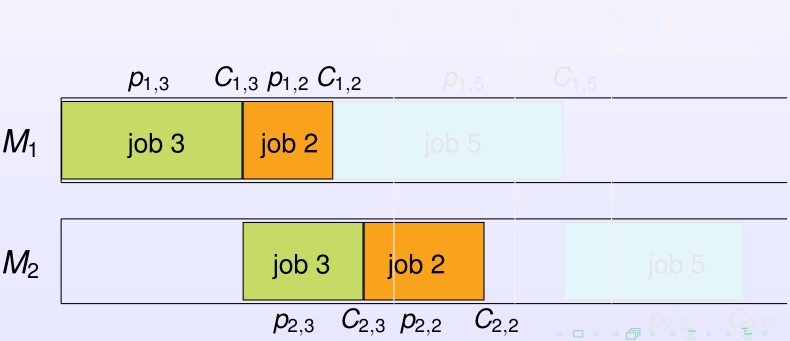
Job 2 early: $C_{2,2} \leq d_2$, $T_2 = 0$

Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$



A two-machine flowshop problem: $F2//\overline{T}$

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

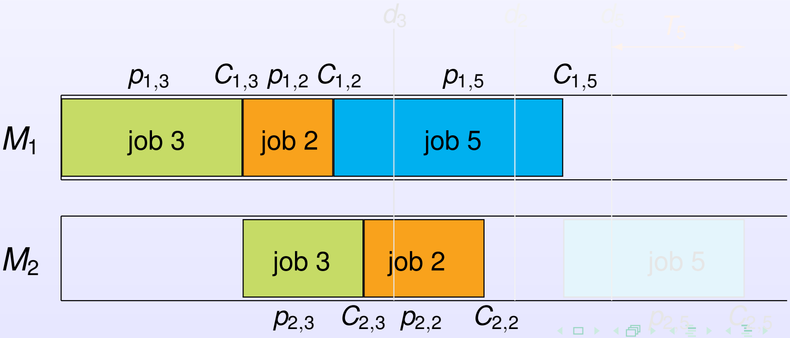


A two-machine flowshop problem: $F2//\overline{T}$

Job 3 early: $C_{2,3} \leq d_3$, $T_3 = 0$

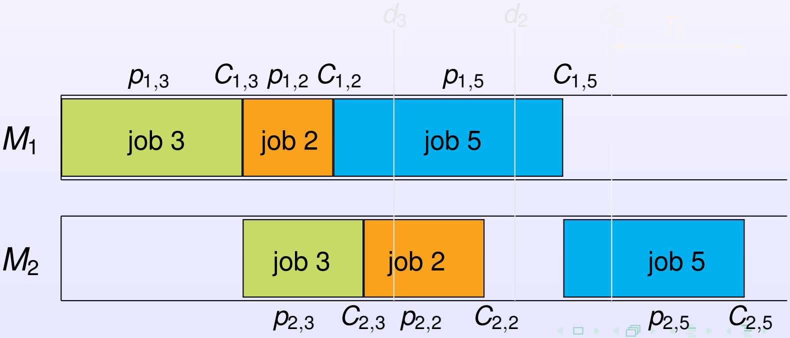
Job 2 early: $C_{2,2} \leq d_2$, $T_2 = 0$

Job 5 tardy: $C_{2,5} > d_5$, $T_5 = C_{2,5} - d_5 > 0$



A two-machine flowshop problem: $F2//\overline{T}$

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

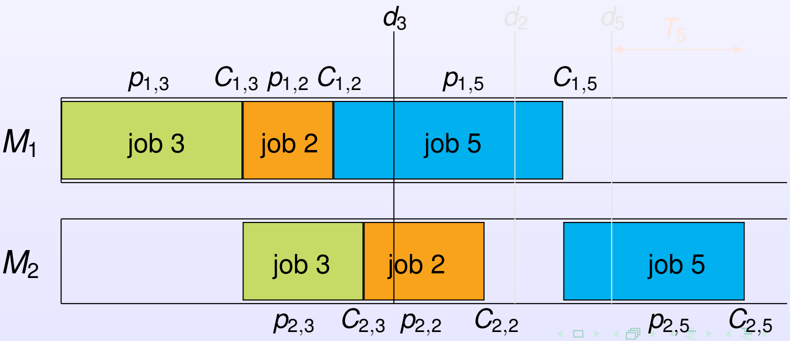


A two-machine flowshop problem: $F2//\overline{T}$

Job 3 early: $C_{2,3} \leq d_3,\ T_3 = 0$

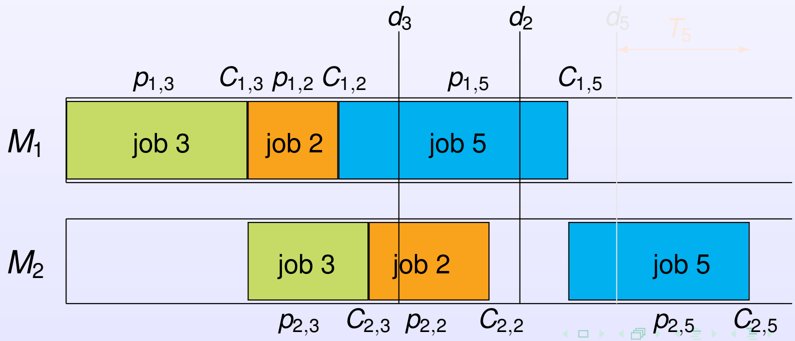
Job 2 early: $C_{2,2} \leq d_2,\ T_2 = 0$

Job 5 tardy: $C_{2,5} > d_5,\ T_5 = C_{2,5} - d_5 > 0$



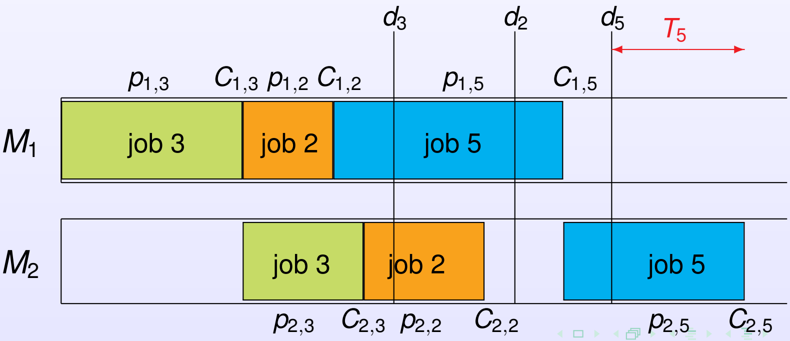
A two-machine flowshop problem: $F2//\overline{T}$

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



A two-machine flowshop problem: $F2//\overline{T}$

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



$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- The data are: n number of jobs
- $(p_{1,j}, p_{2,j}, d_j)$ for $j \in \{1, 2, \ldots, n\}$
- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \ldots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

- 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 $1 \leq k \leq n$)
- A solution is given by a permutation of $\{1, 2, \dots, n\}$
- $s(1)$ 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.

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

$F2//\overline{T}$ problem

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

3 Local search

Iterative improvement algorithms

- 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

Iterative improvement algorithms

- 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

Iterative improvement algorithms

- 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

Iterative improvement algorithms

- 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

Iterative improvement algorithms

- 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

Iterative improvement algorithms

- 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

Iterative improvement algorithms

- 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

- 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

- 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

- 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

- 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

- 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 for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \dots, n\}$
- 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 $k = 1, \dots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\begin{pmatrix} n \\ 2 \end{pmatrix} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- Here a solution is a permutation of $\{1, 2, \ldots, n\}$
- 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 $k = 1, \ldots, n-1$
- 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 $1 \leq k_1 < k_2 \leq n$
- The neighborhood has $\binom{n}{2} = \frac{n(n-1)}{2}$ elements

Local search for the $F2//\overline{T}$

- 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 $1 \leq k_1 < k_2 < k_3 \leq n$
- The neighborhood has $2\begin{pmatrix} n \\ 3 \end{pmatrix} = \frac{n(n-1)(n-2)}{3}$ elements

Local search for the $F2//\overline{T}$

- 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 $1 \leq k_1 < k_2 < k_3 \leq n$
- The neighborhood has $2\begin{pmatrix} n \\ 3 \end{pmatrix} = \frac{n(n-1)(n-2)}{3}$ elements

Local search for the $F2//\overline{T}$

- 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 $1 \leq k_1 < k_2 < k_3 \leq n$
- The neighborhood has $2\begin{pmatrix} n \\ 3 \end{pmatrix} = \frac{n(n-1)(n-2)}{3}$ elements

Local search for the $F2//\overline{T}$

- 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 $1 \leq k_1 < k_2 < k_3 \leq n$.
- The neighborhood has $2\begin{pmatrix} n \\ 3 \end{pmatrix} = \frac{n(n-1)(n-2)}{3}$ elements

Local search for the $F2//\overline{T}$

- 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 $1 \leq k_1 < k_2 < k_3 \leq n$
- The neighborhood has $2\begin{pmatrix} n \\ 3 \end{pmatrix} = \frac{n(n-1)(n-2)}{3}$ elements

Local search for the $F2//\overline{T}$

- 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 $1 \leq k_1 < k_2 < k_3 \leq n$
- The neighborhood has $2\begin{pmatrix} n \\ 3 \end{pmatrix} = \frac{n(n-1)(n-2)}{3}$ elements

Local search for the $F2//\overline{T}$

- 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 $1 \leq k_1 < k_2 < k_3 \leq n$
- The neighborhood has $2\binom{n}{3} = \frac{n(n-1)(n-2)}{3}$ elements

Local search for the $F2//\overline{T}$

- 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 $1 \leq k_1 < k_2 < k_3 \leq n$
- The neighborhood has $2\begin{pmatrix} n \\ 3 \end{pmatrix} = \frac{n(n-1)(n-2)}{3}$ elements

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

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

Local search for the $F2//\overline{T}$

- 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 5: Extraction and Forward Shift Reinsertion: EFSR

Local search for the $F2//\overline{T}$

- 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 5: Extraction and Forward Shift Reinsertion: EFSR

Local search for the $F2//\overline{T}$

- 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 5: Extraction and Forward Shift Reinsertion: EFSR

Local search for the $F2//\overline{T}$

- 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 5: Extraction and Forward Shift Reinsertion: EFSR

Local search for the $F2//\overline{T}$

- 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 5: Extraction and Forward Shift Reinsertion: EFSR

Local search for the $F2//\overline{T}$

- 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 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.

$currentvalue \leftarrow f(s)$
$bestvalue \leftarrow currentvalue$
$bestneighbor \leftarrow s$
for t **in** neighborhood(s) **do**
    if $(f(t) < bestvalue)$ **then**
        $bestvalue \leftarrow f(t)$
        $bestneighbor \leftarrow t$
    end if
end for
$Improved \leftarrow (bestvalue < currentvalue)$
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.

```
Improved ← true
while (Improved) do
    (Improved, bestneighbor) ← Neighbor(s)
end while
return bestneighbor
```