

Design of Heuristic Algorithms for Hard Optimization

Éric D. Taillard

Slides



based on the Book that can be downloaded from

<https://doi.org/10.1007/978-3-031-13714-3>

Updated version, Python codes

Updated version, C codes

Updated version, julia codes



Table of Content

| | | |
|----|--|-----|
| 1 | Elements of Graphs and Complexity Theory | 3 |
| 2 | Combinatorial Optimization Problems | 27 |
| 3 | Problem Modelling | 68 |
| 4 | Constructive Methods | 90 |
| 5 | Local Search | 114 |
| 6 | Decomposition Methods | 142 |
| 7 | Randomized Methods | 162 |
| 8 | Construction Learning | 184 |
| 9 | Local Search Learning | 200 |
| 10 | Population Management | 213 |
| 11 | Heuristics Design | 245 |
| 12 | Bibliography | 256 |

Chapter 1

Elements of Graphs and Complexity Theory

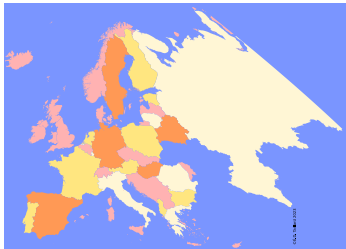
Chapter Content

| | | |
|---|--|----|
| 1 | Elements of Graphs and Complexity Theory | 3 |
| • | Combinatorial Optimization..... | 5 |
| • | Linear Programming | |
| • | A Small Glossary on Graphs and Networks | |
| • | Elements of Algorithmic Complexity and Complexity Theory | 12 |
| • | Algorithmic Complexity | |
| • | Bachmann-Landau Notation | |
| • | Encoding Scheme, Language and Turing Machine | |
| • | Basic Complexity Classes | |

1.1 Combinatorial Optimization

Combinatorial Optimization

An old European map coloured with 5 colours (taking the background into account)

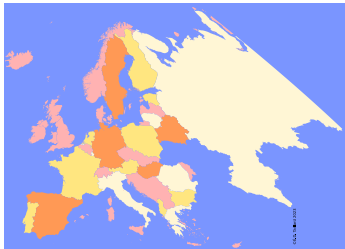


Questions

- How many colour are needed at minimum?
- How to formally model this problem?
- How to proceed to find a feasible colouring?

Combinatorial Optimization

An old European map coloured with 5 colours (taking the background into account)

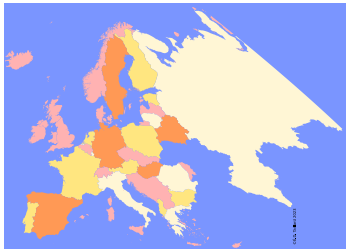


Questions

- How many colour are needed at minimum?
- How to formally model this problem?
- How to proceed to find a feasible colouring?

Combinatorial Optimization

An old European map coloured with 5 colours (taking the background into account)



Questions

- How many colour are needed at minimum?
- How to formally model this problem?
- How to proceed to find a feasible colouring?

Linear Programming

$$\begin{array}{ll}
 \text{Maximize} & z = c_1x_1 + c_2x_2 + \dots + c_nx_n \\
 \text{Subject} & a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1 \\
 \text{to:} & a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2 \\
 & \dots \\
 & a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m \\
 & x_j \geq 0 \ (j = 1, \dots, n)
 \end{array}$$

Formalizing the optimization version of the map colouring problem

$$\text{Minimize } z = \sum_{k=1}^n y_k$$

Subject to:

$$\sum_{k=1}^n x_{ik} = 1 \quad i = 1, \dots, n$$

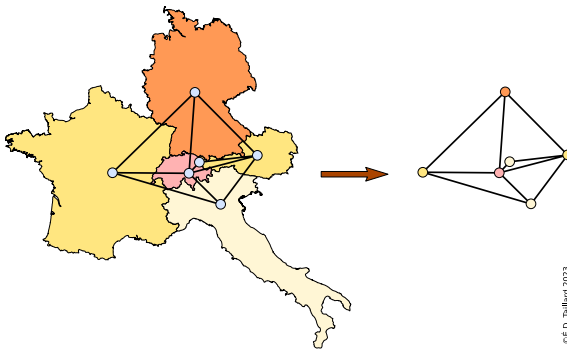
$$x_{ik} - y_k \leq 0 \quad i, k = 1, \dots, n$$

$$x_{ik} + x_{jk} \leq 1 \quad \forall (i, j) \text{ have a common border,} \\ k = 1, \dots, n$$

$$x_{ik}, y_k \in \{0, 1\}$$

Graph Colouring

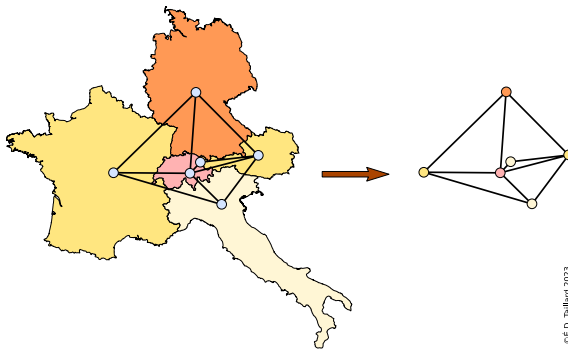
- Switzerland and its neighbour countries that we want to colour
- Each country is symbolized by a disk, and a common border is symbolized by a line connecting the corresponding countries
- The map colouring can be transformed into the colouring of the vertices of a graph



©É.D. Taillard 2023

Graph Colouring

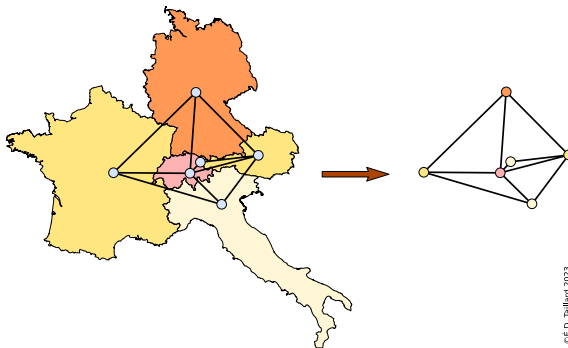
- Switzerland and its neighbour countries that we want to colour
- Each country is symbolized by a disk, and a common border is symbolized by a line connecting the corresponding countries
- The map colouring can be transformed into the colouring of the vertices of a graph



©É.D. Taillard 2023

Graph Colouring

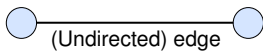
- Switzerland and its neighbour countries that we want to colour
- Each country is symbolized by a disk, and a common border is symbolized by a line connecting the corresponding countries
- The map colouring can be transformed into the colouring of the vertices of a graph



©É.D. Taillard 2023

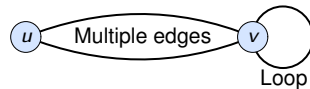
Graph, Vertex, Edge, Path

Vertex, node



$\deg(u) = 2$

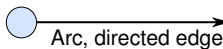
$\deg(v) = 4$



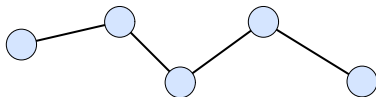
Tail

Head

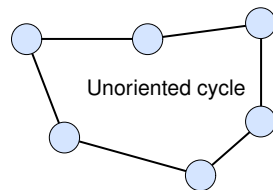
$\deg^+(w) = 0$



$\deg^-(w) = 1$

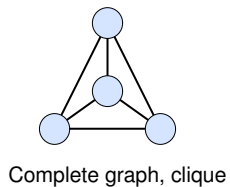
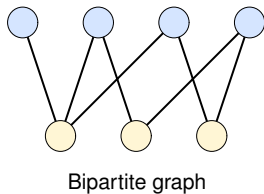
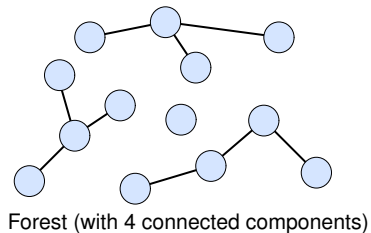
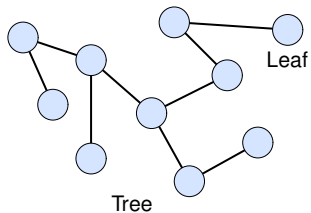


Elementary unoriented path of length 4



©É.D. Taillard 2023

Tree, Forest, Bipartite Graph, Clique

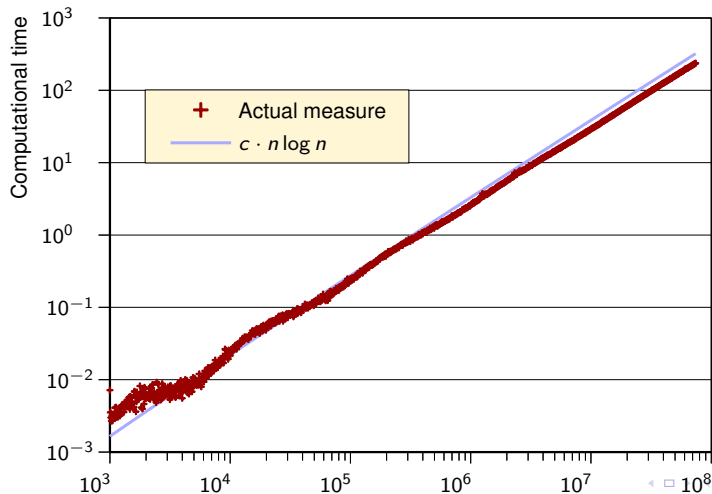


©É.D. Taillard 2023

1.2 Elements of Algorithmic Complexity and Complexity Theory

Empirical Computational Effort

Building a travelling salesman tour as a function of the number n of cities



©É.D. Taillard 2023

Bachmann-Landau Notations

• Big O notation

- $f(n) \in O(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) \in \Omega(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) \in \Theta(g(n))$ if $\exists n_0 > 0, \exists c_2 > c_1 > 0$ such that $\forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

• Little o notation

- $f(n) \in o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} > 0$
- $f(n) \in \omega(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0$
- $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Bachmann-Landau Notations

- Big O notation

- $f(n) \in O(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) \in \Omega(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) \in \Theta(g(n))$ if $\exists n_0 > 0, \exists c_2 > c_1 > 0$ such that $\forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

- Little o notation

- $f(n) \in o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- $f(n) \in \omega(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$
- $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Bachmann-Landau Notations

- Big O notation

- $f(n) \in O(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) \in \Omega(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) \in \Theta(g(n))$ if $\exists n_0 > 0, \exists c_2 > c_1 > 0$ such that $\forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

- Little o notation

- $f(n) \in o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- $f(n) \in \omega(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$
- $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Bachmann-Landau Notations

• Big O notation

- $f(n) \in O(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) \in \Omega(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) \in \Theta(g(n))$ if $\exists n_0 > 0, \exists c_2 > c_1 > 0$ such that $\forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

• Little o notation

- $f(n) \in o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- $f(n) \in \omega(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$
- $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Bachmann-Landau Notations

- Big O notation

- $f(n) \in O(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) \in \Omega(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) \in \Theta(g(n))$ if $\exists n_0 > 0, \exists c_2 > c_1 > 0$ such that $\forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

- Little o notation

- $f(n) \in o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} > 0$
- $f(n) \in \omega(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0$
- $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Bachmann-Landau Notations

- Big O notation

- $f(n) \in O(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) \in \Omega(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) \in \Theta(g(n))$ if $\exists n_0 > 0, \exists c_2 > c_1 > 0$ such that $\forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

- Little o notation

- $f(n) \in o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- $f(n) \in \omega(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0$
- $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Bachmann-Landau Notations

- Big O notation

- $f(n) \in O(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) \in \Omega(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) \in \Theta(g(n))$ if $\exists n_0 > 0, \exists c_2 > c_1 > 0$ such that $\forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

- Little o notation

- $f(n) \in o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- $f(n) \in \omega(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$
- $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Bachmann-Landau Notations

- Big O notation

- $f(n) \in O(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \leq c \cdot g(n)$
- $f(n) \in \Omega(g(n))$ if $\exists n_0 > 0, \exists c > 0$ such that $\forall n \geq n_0, f(n) \geq c \cdot g(n)$
- $f(n) \in \Theta(g(n))$ if $\exists n_0 > 0, \exists c_2 > c_1 > 0$ such that $\forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

- Little o notation

- $f(n) \in o(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- $f(n) \in \omega(g(n))$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$
- $f(n) \sim g(n)$ if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

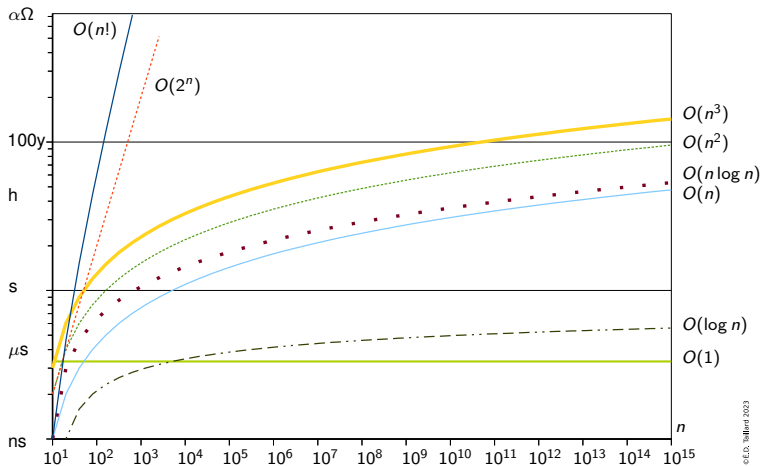
Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

Functions commonly encountered in algorithmic complexity

- $O(1)$: constant.
- $O(\log n)$: logarithmic; the base is not provided since $O(\log_a n) = O(\log_b n)$.
- $O(n^c)$: fractional power, with $0 < c < 1$.
- $O(n)$: linear.
- $O(n \log n)$: linearithmic.
- $O(n^2)$: quadratic.
- $O(n^3)$: cubic.
- $O(n^c)$: polynomial, with $c > 1$ constant.
- $O(n^{\log n})$: quasi-polynomial, super-polynomial, sub-exponential.
- $O(c^n)$: exponential, with $c > 1$ constant.
- $O(n!)$: factorial.

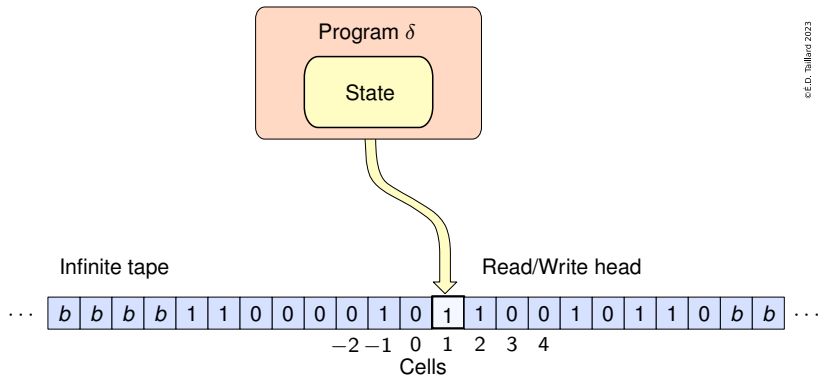
Functions commonly encountered in algorithmic complexity



©É.D. Taillard 2023

Deterministic Turing Machine

Schematic representation of a deterministic Turing machine, which allows modelling and formalizing a computer



Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Specification of a Deterministic Turing Machine

- A tape alphabet Γ containing at least
 - Σ , the set of symbols that encodes a decision problem instance
 - The special blank symbol b not belonging to Σ
 - Eventually other control symbols
- A set of states Q , containing at least
 - q_0 (the initial state)
 - q_Y (the final state indicating that the answer to the instance is “yes”)
 - q_N (the final state indicating that the answer is “no”)
- A transition function $\delta : Q \setminus \{q_Y, q_N\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, 1\}$.

Example of a Turing Machine Program

- Tape alphabet: $\Gamma = \{0, 1, b\}$
- Input alphabet: $\Sigma = \{0, 1\}$
- Set of states: $Q = \{q_0, q_1, q_2, q_3, q_Y, q_N\}$
- Transition function δ

| State | Symbol $\sigma \in \Gamma$ on the tape | | |
|-------|--|----------------|----------------|
| | 0 | 1 | b |
| q_0 | $(q_0, 0, 1)$ | $(q_0, 1, 1)$ | $(q_1, b, -1)$ |
| q_1 | $(q_2, b, -1)$ | $(q_3, b, -1)$ | $(q_N, b, -1)$ |
| q_2 | $(q_Y, b, -1)$ | $(q_N, b, -1)$ | $(q_N, b, -1)$ |
| q_3 | $(q_N, b, -1)$ | $(q_N, b, -1)$ | $(q_N, b, -1)$ |

Example of a Turing Machine Program

- Tape alphabet: $\Gamma = \{0, 1, b\}$
- Input alphabet: $\Sigma = \{0, 1\}$
- Set of states: $Q = \{q_0, q_1, q_2, q_3, q_Y, q_N\}$
- Transition function δ

| State | Symbol $\sigma \in \Gamma$ on the tape | | |
|-------|--|----------------|----------------|
| | 0 | 1 | b |
| q_0 | $(q_0, 0, 1)$ | $(q_0, 1, 1)$ | $(q_1, b, -1)$ |
| q_1 | $(q_2, b, -1)$ | $(q_3, b, -1)$ | $(q_N, b, -1)$ |
| q_2 | $(q_Y, b, -1)$ | $(q_N, b, -1)$ | $(q_N, b, -1)$ |
| q_3 | $(q_N, b, -1)$ | $(q_N, b, -1)$ | $(q_N, b, -1)$ |

Example of a Turing Machine Program

- Tape alphabet: $\Gamma = \{0, 1, b\}$
- Input alphabet: $\Sigma = \{0, 1\}$
- Set of states: $Q = \{q_0, q_1, q_2, q_3, q_Y, q_N\}$
- Transition function δ

| State | Symbol $\sigma \in \Gamma$ on the tape | | |
|-------|--|----------------|----------------|
| | 0 | 1 | b |
| q_0 | $(q_0, 0, 1)$ | $(q_0, 1, 1)$ | $(q_1, b, -1)$ |
| q_1 | $(q_2, b, -1)$ | $(q_3, b, -1)$ | $(q_N, b, -1)$ |
| q_2 | $(q_Y, b, -1)$ | $(q_N, b, -1)$ | $(q_N, b, -1)$ |
| q_3 | $(q_N, b, -1)$ | $(q_N, b, -1)$ | $(q_N, b, -1)$ |

Example of a Turing Machine Program

- Tape alphabet: $\Gamma = \{0, 1, b\}$
- Input alphabet: $\Sigma = \{0, 1\}$
- Set of states: $Q = \{q_0, q_1, q_2, q_3, q_Y, q_N\}$
- Transition function δ

| State | Symbol $\sigma \in \Gamma$ on the tape | | |
|-------|--|----------------|----------------|
| | 0 | 1 | b |
| q_0 | $(q_0, 0, 1)$ | $(q_0, 1, 1)$ | $(q_1, b, -1)$ |
| q_1 | $(q_2, b, -1)$ | $(q_3, b, -1)$ | $(q_N, b, -1)$ |
| q_2 | $(q_Y, b, -1)$ | $(q_N, b, -1)$ | $(q_N, b, -1)$ |
| q_3 | $(q_N, b, -1)$ | $(q_N, b, -1)$ | $(q_N, b, -1)$ |

Class P of languages

- The machine M *accepts* $x \in \Sigma^*$ if and only if M stops in the state q_Y
- The *language recognized* by M is the set of strings $x \in \Sigma^*$ such that M accepts x
- An *algorithm* is a program that stops for any string $x \in \Sigma^*$
- The *computational time* of an algorithm is the number of steps performed by the machine before it stops
- The *complexity* of a program M is the *largest* computational time $T_M(n)$ required by the machine to stop, whatever the string x of length n initially written on the tape is
- A deterministic Turing machine program is in *polynomial time* if there is a polynomial p such that $T_M(n) \leq p(n)$
- The *class P* of languages includes all the languages L such that there is a program for deterministic Turing machine recognizing L in polynomial time

Class P of languages

- The machine M *accepts* $x \in \Sigma^*$ if and only if M stops in the state q_Y
- The *language recognized* by M is the set of strings $x \in \Sigma^*$ such that M accepts x
- An *algorithm* is a program that stops for any string $x \in \Sigma^*$
- The *computational time* of an algorithm is the number of steps performed by the machine before it stops
- The *complexity* of a program M is the *largest* computational time $T_M(n)$ required by the machine to stop, whatever the string x of length n initially written on the tape is
- A deterministic Turing machine program is in *polynomial time* if there is a polynomial p such that $T_M(n) \leq p(n)$
- The *class P* of languages includes all the languages L such that there is a program for deterministic Turing machine recognizing L in polynomial time

Class P of languages

- The machine M *accepts* $x \in \Sigma^*$ if and only if M stops in the state q_Y
- The *language recognized* by M is the set of strings $x \in \Sigma^*$ such that M accepts x
- An *algorithm* is a program that stops for any string $x \in \Sigma^*$
- The *computational time* of an algorithm is the number of steps performed by the machine before it stops
- The *complexity* of a program M is the *largest* computational time $T_M(n)$ required by the machine to stop, whatever the string x of length n initially written on the tape is
- A deterministic Turing machine program is in *polynomial time* if there is a polynomial p such that $T_M(n) \leq p(n)$
- The *class P* of languages includes all the languages L such that there is a program for deterministic Turing machine recognizing L in polynomial time

Class P of languages

- The machine M *accepts* $x \in \Sigma^*$ if and only if M stops in the state q_Y
- The *language recognized* by M is the set of strings $x \in \Sigma^*$ such that M accepts x
- An *algorithm* is a program that stops for any string $x \in \Sigma^*$
- The *computational time* of an algorithm is the number of steps performed by the machine before it stops
- The *complexity* of a program M is the *largest* computational time $T_M(n)$ required by the machine to stop, whatever the string x of length n initially written on the tape is
- A deterministic Turing machine program is in *polynomial time* if there is a polynomial p such that $T_M(n) \leq p(n)$
- The *class P* of languages includes all the languages L such that there is a program for deterministic Turing machine recognizing L in polynomial time

Class P of languages

- The machine M *accepts* $x \in \Sigma^*$ if and only if M stops in the state q_Y
- The *language recognized* by M is the set of strings $x \in \Sigma^*$ such that M accepts x
- An *algorithm* is a program that stops for any string $x \in \Sigma^*$
- The *computational time* of an algorithm is the number of steps performed by the machine before it stops
- The *complexity* of a program M is the *largest* computational time $T_M(n)$ required by the machine to stop, whatever the string x of length n initially written on the tape is
- A deterministic Turing machine program is in *polynomial time* if there is a polynomial p such that $T_M(n) \leq p(n)$
- The *class P* of languages includes all the languages L such that there is a program for deterministic Turing machine recognizing L in polynomial time

Class P of languages

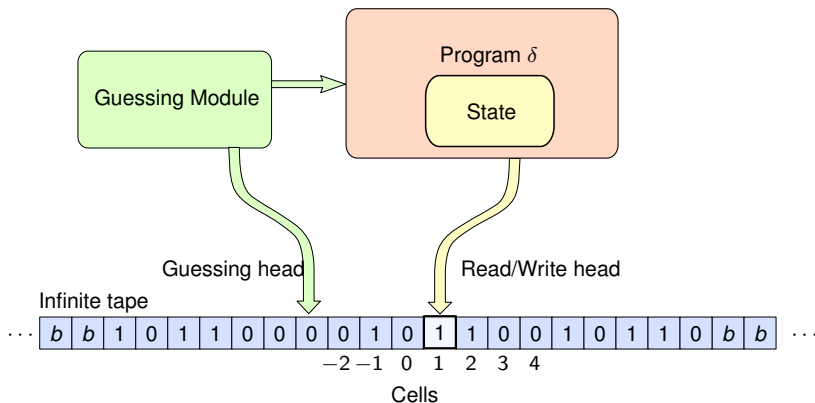
- The machine M *accepts* $x \in \Sigma^*$ if and only if M stops in the state q_Y
- The *language recognized* by M is the set of strings $x \in \Sigma^*$ such that M accepts x
- An *algorithm* is a program that stops for any string $x \in \Sigma^*$
- The *computational time* of an algorithm is the number of steps performed by the machine before it stops
- The *complexity* of a program M is the *largest* computational time $T_M(n)$ required by the machine to stop, whatever the string x of length n initially written on the tape is
- A deterministic Turing machine program is in *polynomial time* if there is a polynomial p such that $T_M(n) \leq p(n)$
- The *class P* of languages includes all the languages L such that there is a program for deterministic Turing machine recognizing L in polynomial time

Class P of languages

- The machine M *accepts* $x \in \Sigma^*$ if and only if M stops in the state q_Y
- The *language recognized* by M is the set of strings $x \in \Sigma^*$ such that M accepts x
- An *algorithm* is a program that stops for any string $x \in \Sigma^*$
- The *computational time* of an algorithm is the number of steps performed by the machine before it stops
- The *complexity* of a program M is the *largest* computational time $T_M(n)$ required by the machine to stop, whatever the string x of length n initially written on the tape is
- A deterministic Turing machine program is in *polynomial time* if there is a polynomial p such that $T_M(n) \leq p(n)$
- The *class P* of languages includes all the languages L such that there is a program for deterministic Turing machine recognizing L in polynomial time

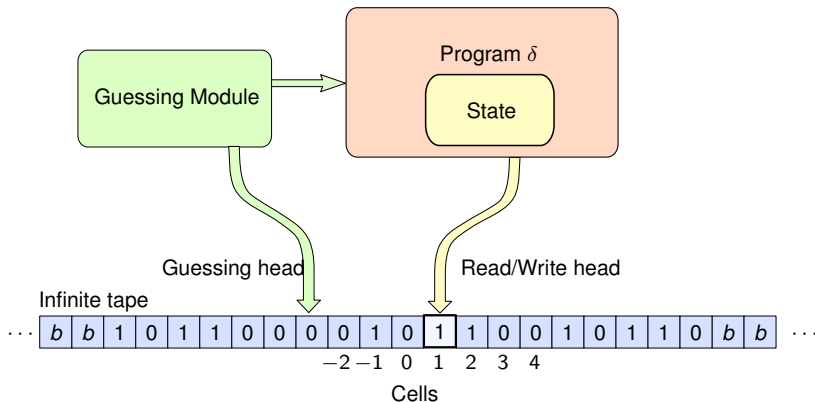
Non-deterministic Turing machine

- Schematic representation of a non-deterministic Turing machine
- This machine allows formalizing the *NP* class, but does not exist in the real world



Non-deterministic Turing machine

- Schematic representation of a non-deterministic Turing machine
- This machine allows formalizing the NP class, but does not exist in the real world



© 2023 Tailored 2023

Class NP of languages

Languages L for which there exists a program M for a non-deterministic Turing machine so that M recognizes L in **polynomial** time

- The guessing module writes the solution of the problem into the negative index cells of the tape
- The program has just to verify that the solution is correct
- Informally the NP class contains all problems for which a solution can be verified in polynomial time

Class NP of languages

Languages L for which there exists a program M for a non-deterministic Turing machine so that M recognizes L in **polynomial** time

- The guessing module writes the solution of the problem into the negative index cells of the tape
- The program has just to verify that the solution is correct
- Informally the *NP* class contains all problems for which a solution can be verified in polynomial time

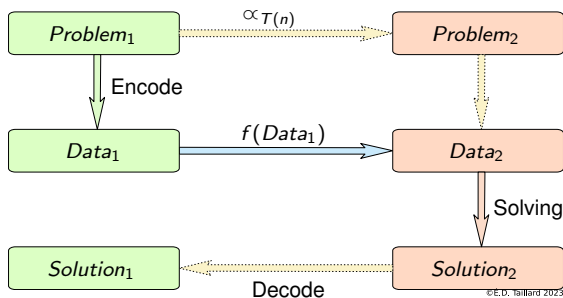
Class NP of languages

Languages L for which there exists a program M for a non-deterministic Turing machine so that M recognizes L in **polynomial** time

- The guessing module writes the solution of the problem into the negative index cells of the tape
- The program has just to verify that the solution is correct
- Informally the NP class contains all problems for which a solution can be verified in polynomial time

Polynomial-Time Reduction

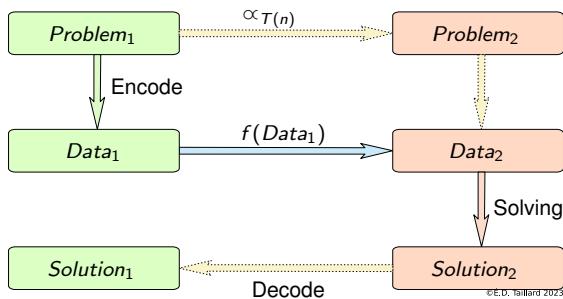
- Reduction of $Problem_1$ to $Problem_2$ in time $T(n)$ where
- If $T(n)$ is a polynomial and $Problem_1$ has a solution $\iff Problem_2$ has a solution, then we have a *polynomial-time reduction*
- The theory only requires to be able to carry out the operations represented with solid line arrows



©É.D. Taillard 2023

Polynomial-Time Reduction

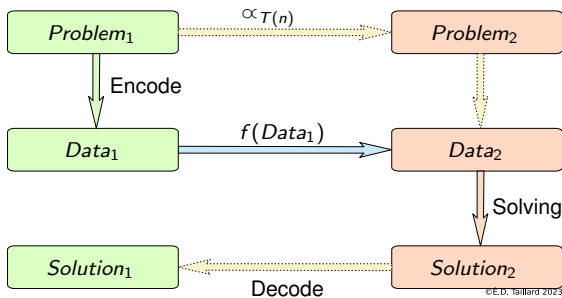
- Reduction of $Problem_1$ to $Problem_2$ in time $T(n)$ where
- If $T(n)$ is a polynomial and $Problem_1$ has a solution $\iff Problem_2$ has a solution, then we have a *polynomial-time reduction*
- The theory only requires to be able to carry out the operations represented with solid line arrows



©É.D. Taillard 2023

Polynomial-Time Reduction

- Reduction of $Problem_1$ to $Problem_2$ in time $T(n)$ where
- If $T(n)$ is a polynomial and $Problem_1$ has a solution $\iff Problem_2$ has a solution, then we have a *polynomial-time reduction*
- The theory only requires to be able to carry out the operations represented with solid line arrows



©É.D. Taillard 2023

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

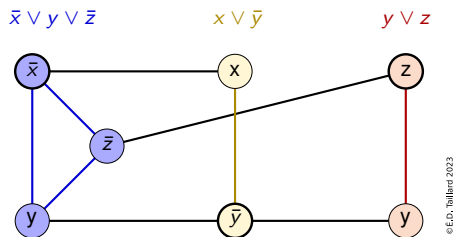
Class NP-Complete

A problem π belongs to the *class NP-Complete* if π belongs to NP and every problem of NP can be polynomially reduced to π

- If π is NP-complete and π can be solved in polynomial time, then $P = NP$
- If π is NP-complete and π does not belong to P , then $P \neq NP$
- If π_1 polynomially reduces to π_2 and π_2 polynomially reduces to π_3 , then π_1 polynomially reduces to π_3
- If π_1 is NP-complete, π_2 belongs to NP and π_1 polynomially reduces to π_2 , then π_2 is NP-complete
- (Cook) satisfiability is NP-complete; using the properties above, it is quite easy to show that other problems are NP-complete:
 - Stable set
 - Graph colouring
 - Integer linear programming
 - Travelling salesman
 - etc.

Polynomial Reduction of Satisfiability to the Stable Set Problem

$$(\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y}) \wedge (y \vee z)$$



©É.D. Taillard 2023

Other Complexity Classes

- *NP-Hard* Extension of NP-complete to non-decision problems
 - Optimization version of NP-complete problems
- *P-SPACE* Can be solved with a memory limited by a polynomial
 - Determine if a two-player game is fair
- *Class L* Can be solved with an additional memory limited by a logarithmic of the data size
 - Processing a database not fitting in RAM
- *Class NC* Can be solved in polylogarithmic time on a machine including a polynomial number of processors
 - Sorting the elements of an array

Other Complexity Classes

- *NP-Hard* Extension of NP-complete to non-decision problems
 - Optimization version of NP-complete problems
- *P-SPACE* Can be solved with a memory limited by a polynomial
 - Determine if a two-player game is fair
- *Class L* Can be solved with an additional memory limited by a logarithmic of the data size
 - Processing a database not fitting in RAM
- *Class NC* Can be solved in polylogarithmic time on a machine including a polynomial number of processors
 - Sorting the elements of an array

Other Complexity Classes

- *NP-Hard* Extension of NP-complete to non-decision problems
 - Optimization version of NP-complete problems
- *P-SPACE* Can be solved with a memory limited by a polynomial
 - Determine if a two-player game is fair
- *Class L* Can be solved with an additional memory limited by a logarithmic of the data size
 - Processing a database not fitting in RAM
- *Class NC* Can be solved in polylogarithmic time on a machine including a polynomial number of processors
 - Sorting the elements of an array

Other Complexity Classes

- *NP-Hard* Extension of NP-complete to non-decision problems
 - Optimization version of NP-complete problems
- *P-SPACE* Can be solved with a memory limited by a polynomial
 - Determine if a two-player game is fair
- *Class L* Can be solved with an additional memory limited by a logarithmic of the data size
 - Processing a database not fitting in RAM
- *Class NC* Can be solved in polylogarithmic time on a machine including a polynomial number of processors
 - Sorting the elements of an array

Other Complexity Classes

- *NP-Hard* Extension of NP-complete to non-decision problems
 - Optimization version of NP-complete problems
- *P-SPACE* Can be solved with a memory limited by a polynomial
 - Determine if a two-player game is fair
- *Class L* Can be solved with an additional memory limited by a logarithmic of the data size
 - Processing a database not fitting in RAM
- *Class NC* Can be solved in polylogarithmic time on a machine including a polynomial number of processors
 - Sorting the elements of an array

Other Complexity Classes

- *NP-Hard* Extension of NP-complete to non-decision problems
 - Optimization version of NP-complete problems
- *P-SPACE* Can be solved with a memory limited by a polynomial
 - Determine if a two-player game is fair
- *Class L* Can be solved with an additional memory limited by a logarithmic of the data size
 - Processing a database not fitting in RAM
- *Class NC* Can be solved in polylogarithmic time on a machine including a polynomial number of processors
 - Sorting the elements of an array

Other Complexity Classes

- *NP-Hard* Extension of NP-complete to non-decision problems
 - Optimization version of NP-complete problems
- *P-SPACE* Can be solved with a memory limited by a polynomial
 - Determine if a two-player game is fair
- *Class L* Can be solved with an additional memory limited by a logarithmic of the data size
 - Processing a database not fitting in RAM
- *Class NC* Can be solved in polylogarithmic time on a machine including a polynomial number of processors
 - Sorting the elements of an array

Other Complexity Classes

- *NP-Hard* Extension of NP-complete to non-decision problems
 - Optimization version of NP-complete problems
- *P-SPACE* Can be solved with a memory limited by a polynomial
 - Determine if a two-player game is fair
- *Class L* Can be solved with an additional memory limited by a logarithmic of the data size
 - Processing a database not fitting in RAM
- *Class NC* Can be solved in polylogarithmic time on a machine including a polynomial number of processors
 - Sorting the elements of an array

Chapter 2

Combinatorial Optimization Problems

Chapter Content

| | | |
|---|---|----|
| 2 | Combinatorial Optimization Problems | 27 |
| • | Optimal Trees | 29 |
| • | Steiner Tree | |
| • | Optimal Paths | 33 |
| • | Shortest Path | |
| • | Elementary Shortest Path | |
| • | Vehicle Routing | |
| • | Scheduling | 41 |
| • | Permutation Flow Shop Scheduling | |
| • | Job Shop Scheduling | |
| • | Flows in networks | 47 |
| • | Assignment Problems | 53 |
| • | Linear Assignment | |
| • | Knapsack | |
| • | Quadratic Assignment | |
| • | Stable Set | 59 |
| • | Clustering | |

2.1 Optimal Trees

Kruskal Algorithm for Finding a Minimum Spanning Tree

For each edge, decide whether it is kept or not

- There are $2^{|E|}$ potential solutions
- This is a combinatorial problem

Data: Undirected connected network $R = (V, E, w)$

Result: Minimum spanning tree $T = (V, E_T)$

1 Sort and renumber the edges by nondecreasing weight $w(e_1) \leq w(e_2) \leq \dots \leq w(e_{|E|})$

2 $E_T = \emptyset$

3 **for** $k = 1 \dots |E|$ **do**

4 **if** $E_T \cup \{e_k\}$ has no cycle **then**

5 $E_T \leftarrow E_T \cup \{e_k\}$

- This is a greedy algorithm
- The algorithm is polynomial, so the problem is simple

Kruskal Algorithm for Finding a Minimum Spanning Tree

For each edge, decide whether it is kept or not

- There are $2^{|E|}$ potential solutions
- This is a combinatorial problem

Data: Undirected connected network $R = (V, E, w)$

Result: Minimum spanning tree $T = (V, E_T)$

1 Sort and renumber the edges by nondecreasing weight $w(e_1) \leq w(e_2) \leq \dots \leq w(e_{|E|})$

2 $E_T = \emptyset$

3 **for** $k = 1 \dots |E|$ **do**

4 **if** $E_T \cup \{e_k\}$ has no cycle **then**

5 $E_T \leftarrow E_T \cup \{e_k\}$

- This is a **greedy** algorithm
- The algorithm is polynomial, so the problem is simple

Kruskal Algorithm for Finding a Minimum Spanning Tree

For each edge, decide whether it is kept or not

- There are $2^{|E|}$ potential solutions
- This is a combinatorial problem

Data: Undirected connected network $R = (V, E, w)$

Result: Minimum spanning tree $T = (V, E_T)$

1 Sort and renumber the edges by nondecreasing weight $w(e_1) \leq w(e_2) \leq \dots \leq w(e_{|E|})$

2 $E_T = \emptyset$

3 **for** $k = 1 \dots |E|$ **do**

4 **if** $E_T \cup \{e_k\}$ has no cycle **then**

5 $E_T \leftarrow E_T \cup \{e_k\}$

- This is a **greedy** algorithm
- The algorithm is polynomial, so the problem is simple

Kruskal Algorithm for Finding a Minimum Spanning Tree

For each edge, decide whether it is kept or not

- There are $2^{|E|}$ potential solutions
- This is a combinatorial problem

Data: Undirected connected network $R = (V, E, w)$

Result: Minimum spanning tree $T = (V, E_T)$

1 Sort and renumber the edges by nondecreasing weight $w(e_1) \leq w(e_2) \leq \dots \leq w(e_{|E|})$

2 $E_T = \emptyset$

3 **for** $k = 1 \dots |E|$ **do**

4 **if** $E_T \cup \{e_k\}$ has no cycle **then**

5 $E_T \leftarrow E_T \cup \{e_k\}$

- This is a **greedy** algorithm
- The algorithm is polynomial, so the problem is simple

Jarník Algorithm for Finding a Minimum Spanning Tree

Known Under Prim or Prim-Dijkstra Algorithm

Data: Undirected connected network $R = (V, E, w)$, a given vertex $s \in V$

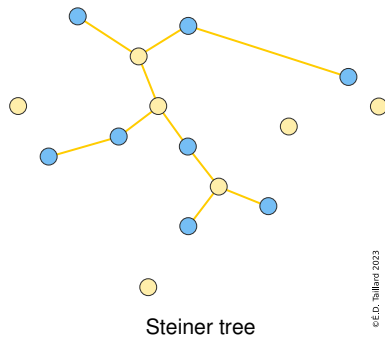
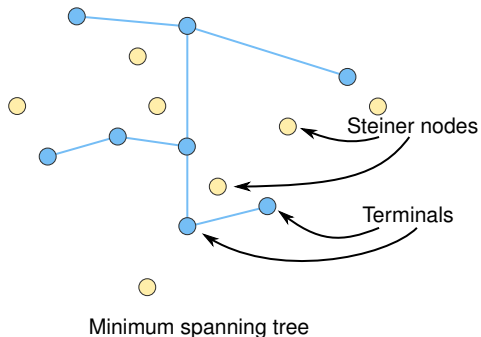
Result: Minimum spanning tree $T = (V, E_T)$

```

1 forall Vertex  $i \in V$  do
2    $\lambda_i \leftarrow \infty$                                      // Cost for introducing  $i$  into  $T$ 
3    $pred_i \leftarrow \emptyset$                              // Predecessor of  $i$ 
4  $\lambda_s = 0$ ;  $E_T \leftarrow \emptyset$ 
5  $L \leftarrow V$                                            // List of vertices to introduce in  $T$ 
6 while  $L \neq \emptyset$  do
7   Remove the vertex  $i$  with the smallest  $\lambda_i$  from  $L$ 
8   if  $i \neq s$  then
9      $E_T \leftarrow E_T \cup \{pred_i, i\}$ 
10    forall Vertex  $j$  adjacent to  $i$  do
11      if  $j \in L$  and  $\lambda_j > w(\{i, j\})$  then
12         $\lambda_j \leftarrow w(\{i, j\})$ 
13         $pred_j \leftarrow i$ 
14
15

```

Steiner Tree



©É.D. Taillard 2023

The combinatorial nature of the optional (Steiner) vertices to be introduced makes the problem NP-hard

2.2 Optimal Paths

Dijkstra Algorithm for Finding a Shortest Path

Data: Directed Network $R = (V, E, w)$ with $w(e) \geq 0 \forall e \in E$, given by successor lists $\text{succ}(i)$ for each vertex $i \in V$, a given vertex s

Result: Immediate predecessor pred_j of j on a shortest path from s to j , $\forall j \in V$ and length λ_j of the shortest path from s to j

```

1 forall Vertex  $i \in V$  do
2    $\lambda_i \leftarrow \infty$ 
3    $\text{pred}_i \leftarrow \emptyset$ 
4  $\lambda_s = 0$ 
5  $L \leftarrow V$  // Vertices for which the shortest path is not definitive
6 repeat
7   Remove vertex  $i$  with smallest  $\lambda_i$  value from  $L$ 
8   forall Vertices  $j \in \text{succ}(i)$  do
9     if  $j \in L$  and  $\lambda_j > \lambda_i + w(i, j)$  then
10       $\lambda_j \leftarrow \lambda_i + w(i, j)$ 
11       $\text{pred}_j \leftarrow i$ 
12
13
14 until  $L \neq \emptyset$ 
  
```

Bellman-Ford Algorithm for the Shortest Path

Data: Directed network $R = (V, E, w)$ given with an arc list, a starting node s

Result: Immediate predecessor $pred_j$ of j on a shortest path from s to j with its length λ_j , $\forall j \in V$, or: warning message of the existence of a negative length circuit

```

1 forall  $i \in V$  do
2    $\lambda_i \leftarrow \infty$  ;  $pred_i \leftarrow \emptyset$ 
3  $\lambda_s \leftarrow 0$ 
4  $k \leftarrow 0$ 
5  $continue \leftarrow true$ 
6 while  $k < |V|$  and  $continue$  do
7    $continue \leftarrow false$ 
8    $k \leftarrow k + 1$ 
9   forall  $arc(i, j) \in E$  do
10    if  $\lambda_j > \lambda_i + w(i, j)$  then
11       $\lambda_j \leftarrow \lambda_i + w(i, j)$ 
12       $pred_j \leftarrow i$ 
13       $continue \leftarrow true$ 
14 if  $k = |V|$  then
15   Warning: there is a negative length circuit that can be reached from  $s$ 

```

// Step counter

// At least one λ modified at last step

Operating Differences Between Dijkstra et Bellman Algorithms

Kruskal, Prim et Dijkstra are greedy algorithms

- A choice made at a step is never questioned
- The choices to be made may differ from one algorithm to another
 - Kruskal: one edge is introduced at each step
 - Prim: a vertex is introduced at each step
- Prim and Dijkstra can be formulated almost identically

Bellman-Ford works on the principle of local improvements

- We start with a solution that is not necessarily good
- At each step, the solution is slightly improved

Operating Differences Between Dijkstra et Bellman Algorithms

Kruskal, Prim et Dijkstra are greedy algorithms

- A choice made at a step is never questioned
- The choices to be made may differ from one algorithm to another
 - Kruskal: one edge is introduced at each step
 - Prim: a vertex is introduced at each step
- Prim and Dijkstra can be formulated almost identically

Bellman-Ford works on the principle of local improvements

- We start with a solution that is not necessarily good
- At each step, the solution is slightly improved

Operating Differences Between Dijkstra et Bellman Algorithms

Kruskal, Prim et Dijkstra are greedy algorithms

- A choice made at a step is never questioned
- The choices to be made may differ from one algorithm to another
 - Kruskal: one edge is introduced at each step
 - Prim: a vertex is introduced at each step
- Prim and Dijkstra can be formulated almost identically

Bellman-Ford works on the principle of local improvements

- We start with a solution that is not necessarily good
- At each step, the solution is slightly improved

Operating Differences Between Dijkstra et Bellman Algorithms

Kruskal, Prim et Dijkstra are greedy algorithms

- A choice made at a step is never questioned
- The choices to be made may differ from one algorithm to another
 - Kruskal: one edge is introduced at each step
 - Prim: a vertex is introduced at each step
- Prim and Dijkstra can be formulated almost identically

Bellman-Ford works on the principle of local improvements

- We start with a solution that is not necessarily good
- At each step, the solution is slightly improved

Operating Differences Between Dijkstra et Bellman Algorithms

Kruskal, Prim et Dijkstra are greedy algorithms

- A choice made at a step is never questioned
- The choices to be made may differ from one algorithm to another
 - Kruskal: one edge is introduced at each step
 - Prim: a vertex is introduced at each step
- Prim and Dijkstra can be formulated almost identically

Bellman-Ford works on the principle of local improvements

- We start with a solution that is not necessarily good
- At each step, the solution is slightly improved

Operating Differences Between Dijkstra et Bellman Algorithms

Kruskal, Prim et Dijkstra are greedy algorithms

- A choice made at a step is never questioned
- The choices to be made may differ from one algorithm to another
 - Kruskal: one edge is introduced at each step
 - Prim: a vertex is introduced at each step
- Prim and Dijkstra can be formulated almost identically

Bellman-Ford works on the principle of local improvements

- We start with a solution that is not necessarily good
- At each step, the solution is slightly improved

Operating Differences Between Dijkstra et Bellman Algorithms

Kruskal, Prim et Dijkstra are greedy algorithms

- A choice made at a step is never questioned
- The choices to be made may differ from one algorithm to another
 - Kruskal: one edge is introduced at each step
 - Prim: a vertex is introduced at each step
- Prim and Dijkstra can be formulated almost identically

Bellman-Ford works on the principle of local improvements

- We start with a solution that is not necessarily good
- At each step, the solution is slightly improved

Formulation of the shortest path problem in terms of a linear program

Primal:

$$\begin{aligned}
 &\text{Minimize } z = \sum_{i,j} w(i,j)x_{ij} \\
 &\text{s.t. } \sum_{i=1}^n x_{ij} - \sum_{k=1}^n x_{jk} = 0 \quad \forall j \neq s, j \neq t \\
 &\quad \sum_{i=1}^n x_{it} - \sum_{k=1}^n x_{tk} = 1 \\
 &\quad x_{ij} \geq 0 \quad \forall i,j
 \end{aligned}$$

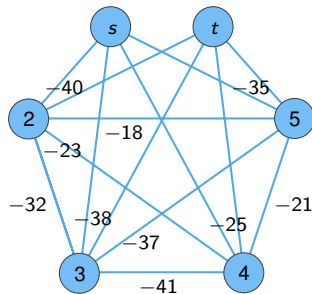
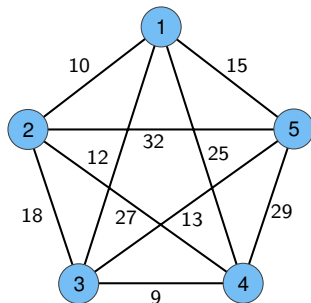
Dual:

$$\begin{aligned}
 &\text{Maximize } \lambda_t \\
 &\text{s.t. } \lambda_j - \lambda_i \leq w(i,j) \quad \forall i,j \\
 &\quad \lambda_s = 0
 \end{aligned}$$

Polynomial Reduction of Travelling Salesman to Elementary Path

Travelling Salesman Problem (TSP)

- Find the shortest Hamiltonian cycle in a weighted network
- It has been proved that the search for a Hamiltonian path is NP-complete
- From this result, it is easy to prove that the TSP is difficult

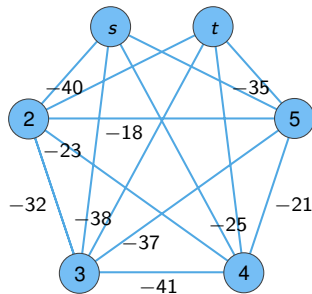
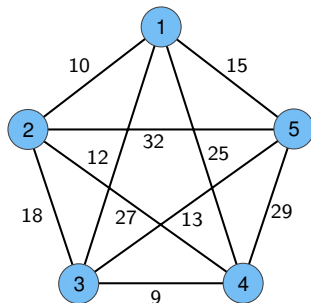


©É.D. Taillard 2023

Polynomial Reduction of Travelling Salesman to Elementary Path

Travelling Salesman Problem (TSP)

- Find the shortest Hamiltonian cycle in a weighted network
- It has been proved that the search for a Hamiltonian path is NP-complete
- From this result, it is easy to prove that the TSP is difficult

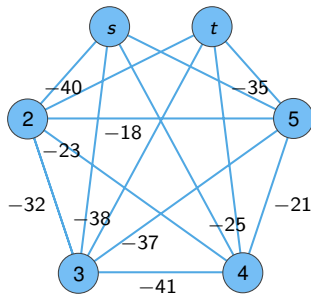
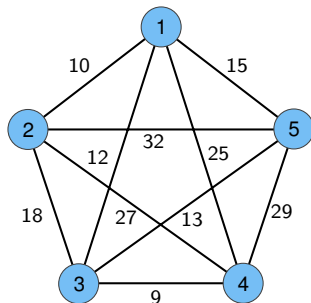


©É.D. Taillard 2023

Polynomial Reduction of Travelling Salesman to Elementary Path

Travelling Salesman Problem (TSP)

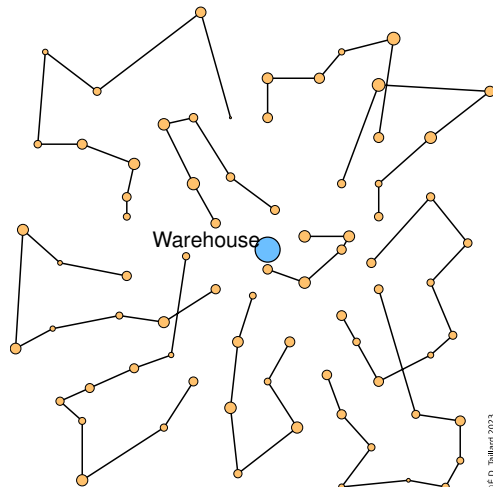
- Find the shortest Hamiltonian cycle in a weighted network
- It has been proved that the search for a Hamiltonian path is NP-complete
- From this result, it is easy to prove that the TSP is difficult



©É.D. Taillard 2023

Vehicle Routing

- Set V of customers requiring quantities q_i of goods ($i = 1, \dots, |V|$)
- A vehicle of capacity Q
- A warehouse d
- Split V into V_1, \dots, V_m such that $\sum_{i \in V_j} q_i \leq Q$
- Solve a TSP for each $V_j \cup \{d\}$
- Minimize the total distance performed by the vehicle



©É.D. Taillard 2023

Vehicle Routing

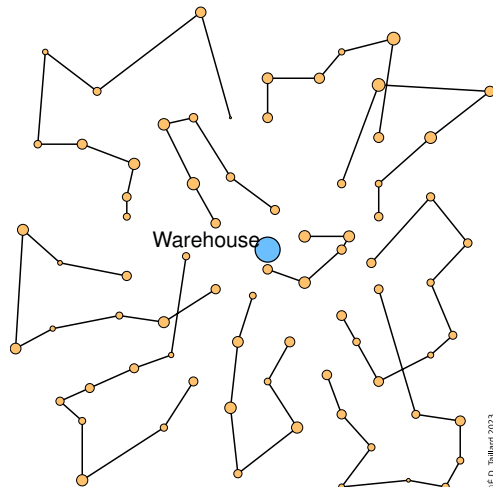
- Set V of customers requiring quantities q_i of goods ($i = 1, \dots, |V|$)
- A vehicle of capacity Q
- A warehouse d
- Split V into V_1, \dots, V_m such that $\sum_{i \in V_j} q_i \leq Q$
- Solve a TSP for each $V_j \cup \{d\}$
- Minimize the total distance performed by the vehicle



©É.D. Taillard 2023

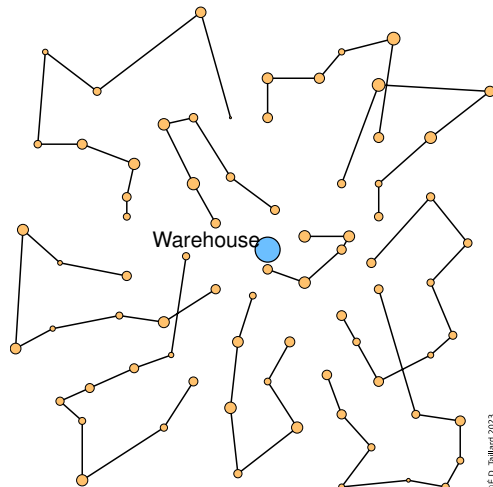
Vehicle Routing

- Set V of customers requiring quantities q_i of goods ($i = 1, \dots, |V|$)
- A vehicle of capacity Q
- A warehouse d
- Split V into V_1, \dots, V_m such that $\sum_{i \in V_j} q_i \leq Q$
- Solve a TSP for each $V_j \cup \{d\}$
- Minimize the total distance performed by the vehicle



Vehicle Routing

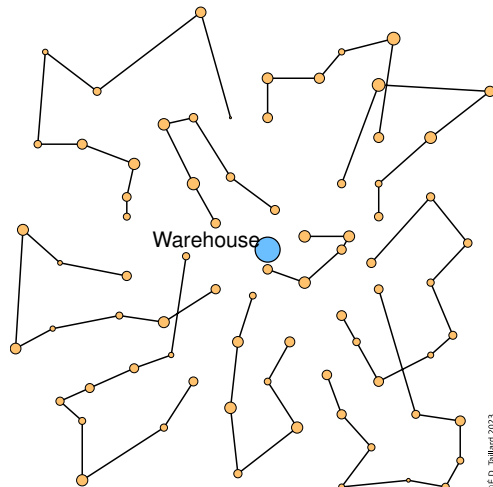
- Set V of customers requiring quantities q_i of goods ($i = 1, \dots, |V|$)
- A vehicle of capacity Q
- A warehouse d
- Split V into V_1, \dots, V_m such that $\sum_{i \in V_j} q_i \leq Q$
- Solve a TSP for each $V_j \cup \{d\}$
- Minimize the total distance performed by the vehicle



©É.D. Taillard 2023

Vehicle Routing

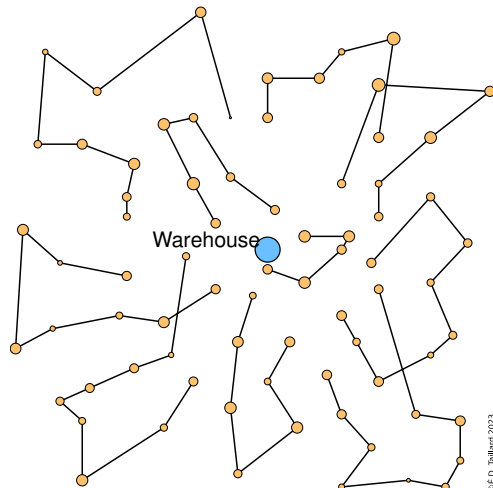
- Set V of customers requiring quantities q_i of goods ($i = 1, \dots, |V|$)
- A vehicle of capacity Q
- A warehouse d
- Split V into V_1, \dots, V_m such that $\sum_{i \in V_j} q_i \leq Q$
- Solve a TSP for each $V_j \cup \{d\}$
- Minimize the total distance performed by the vehicle



©É.D. Taillard 2023

Vehicle Routing

- Set V of customers requiring quantities q_i of goods ($i = 1, \dots, |V|$)
- A vehicle of capacity Q
- A warehouse d
- Split V into V_1, \dots, V_m such that $\sum_{i \in V_j} q_i \leq Q$
- Solve a TSP for each $V_j \cup \{d\}$
- Minimize the total distance performed by the vehicle



©É.D. Taillard 2023

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

Variants of Vehicle Routing Problems

- The number m of tours can be *a priori* set or minimized
- The maximum length of tours can be limited
- Customers specify one or more time windows during which they want the vehicle to visit
- Orders can be split up as required, which means several visits to the same customer
- A tour combines collection and distribution
- We have more than one warehouse
- Depots house a fleet of heterogeneous or non-homogeneous vehicles
- The position of warehouses can be chosen
- Etc.

2.3 Scheduling

Scheduling

Determine the order in which to perform operations. Many parameters:

- Ressource** An operation must take place on a given resource or subset of resources or involve several resources simultaneously
- Duration** It takes a certain amount of time to complete an operation, which may depend on the resource used
- Setup** Before performing an operation, the resource requires a set-up time depending on the previously performed operation
- Interrupt** Once an operation has started, it can be put on hold (or not) before it ends
- Preemption** A resource can interrupt (or not) an operation to perform another one
- Wait** There may or may not be a dead time between two successive operations of the same task
- Release** An operation cannot take place before a certain time
- Deadline** An operation cannot take place after a certain hour

Scheduling

Determine the order in which to perform operations. Many parameters:

- Ressource** An operation must take place on a given resource or subset of resources or involve several resources simultaneously
- Duration** It takes a certain amount of time to complete an operation, which may depend on the resource used
- Setup** Before performing an operation, the resource requires a set-up time depending on the previously performed operation
- Interrupt** Once an operation has started, it can be put on hold (or not) before it ends
- Preemption** A resource can interrupt (or not) an operation to perform another one
- Wait** There may or may not be a dead time between two successive operations of the same task
- Release** An operation cannot take place before a certain time
- Deadline** An operation cannot take place after a certain hour

Scheduling

Determine the order in which to perform operations. Many parameters:

- Ressource** An operation must take place on a given resource or subset of resources or involve several resources simultaneously
- Duration** It takes a certain amount of time to complete an operation, which may depend on the resource used
- Setup** Before performing an operation, the resource requires a set-up time depending on the previously performed operation
- Interrupt** Once an operation has started, it can be put on hold (or not) before it ends
- Preemption** A resource can interrupt (or not) an operation to perform another one
- Wait** There may or may not be a dead time between two successive operations of the same task
- Release** An operation cannot take place before a certain time
- Deadline** An operation cannot take place after a certain hour

Scheduling

Determine the order in which to perform operations. Many parameters:

- Ressource** An operation must take place on a given resource or subset of resources or involve several resources simultaneously
- Duration** It takes a certain amount of time to complete an operation, which may depend on the resource used
- Setup** Before performing an operation, the resource requires a set-up time depending on the previously performed operation
- Interrupt** Once an operation has started, it can be put on hold (or not) before it ends
- Preemption** A resource can interrupt (or not) an operation to perform another one
- Wait** There may or may not be a dead time between two successive operations of the same task
- Release** An operation cannot take place before a certain time
- Deadline** An operation cannot take place after a certain hour

Scheduling

Determine the order in which to perform operations. Many parameters:

- Ressource** An operation must take place on a given resource or subset of resources or involve several resources simultaneously
- Duration** It takes a certain amount of time to complete an operation, which may depend on the resource used
- Setup** Before performing an operation, the resource requires a set-up time depending on the previously performed operation
- Interrupt** Once an operation has started, it can be put on hold (or not) before it ends
- Preemption** A resource can interrupt (or not) an operation to perform another one
- Wait** There may or may not be a dead time between two successive operations of the same task
- Release** An operation cannot take place before a certain time
- Deadline** An operation cannot take place after a certain hour

Scheduling

Determine the order in which to perform operations. Many parameters:

- Ressource** An operation must take place on a given resource or subset of resources or involve several resources simultaneously
- Duration** It takes a certain amount of time to complete an operation, which may depend on the resource used
- Setup** Before performing an operation, the resource requires a set-up time depending on the previously performed operation
- Interrupt** Once an operation has started, it can be put on hold (or not) before it ends
- Preemption** A resource can interrupt (or not) an operation to perform another one
- Wait** There may or may not be a dead time between two successive operations of the same task
- Release** An operation cannot take place before a certain time
- Deadline** An operation cannot take place after a certain hour

Scheduling

Determine the order in which to perform operations. Many parameters:

- Ressource** An operation must take place on a given resource or subset of resources or involve several resources simultaneously
- Duration** It takes a certain amount of time to complete an operation, which may depend on the resource used
- Setup** Before performing an operation, the resource requires a set-up time depending on the previously performed operation
- Interrupt** Once an operation has started, it can be put on hold (or not) before it ends
- Preemption** A resource can interrupt (or not) an operation to perform another one
- Wait** There may or may not be a dead time between two successive operations of the same task
- Release** An operation cannot take place before a certain time
- Deadline** An operation cannot take place after a certain hour

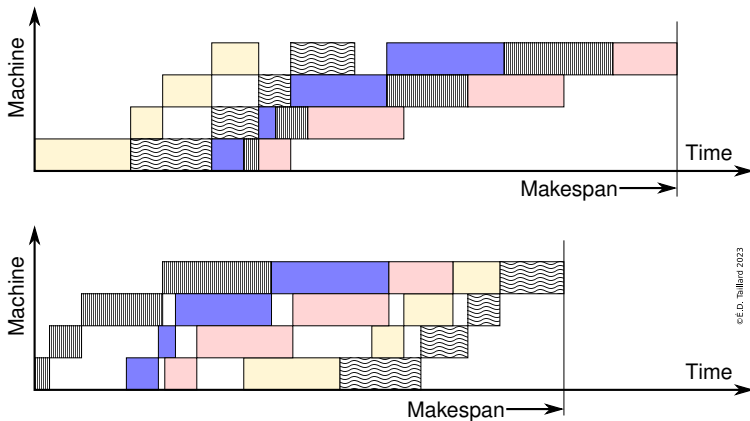
Scheduling

Determine the order in which to perform operations. Many parameters:

- Ressource** An operation must take place on a given resource or subset of resources or involve several resources simultaneously
- Duration** It takes a certain amount of time to complete an operation, which may depend on the resource used
- Setup** Before performing an operation, the resource requires a set-up time depending on the previously performed operation
- Interrupt** Once an operation has started, it can be put on hold (or not) before it ends
- Preemption** A resource can interrupt (or not) an operation to perform another one
- Wait** There may or may not be a dead time between two successive operations of the same task
- Release** An operation cannot take place before a certain time
- Deadline** An operation cannot take place after a certain hour

Permutation Flow Shop Scheduling

Application: Assembly line with m machines, each object to undergo m operations, one per machine, without interruption or pre-emption



PL formulation of the Permutation Flow Shop Scheduling

Minimize d_ω

$$d_{mj} + t_{mj} \leq d_\omega \quad (j = 1 \dots n)$$

$$d_{ij} + t_{ij} \leq d_{i+1j} \quad (i = 1, \dots, m-1, j = 1 \dots n)$$

$$d_{ij} + t_{ij} \leq d_{ik} + M \cdot (1 - y_{jk}) \quad (i = 1, \dots, m, j = 1 \dots n, j < k = 2 \dots n)$$

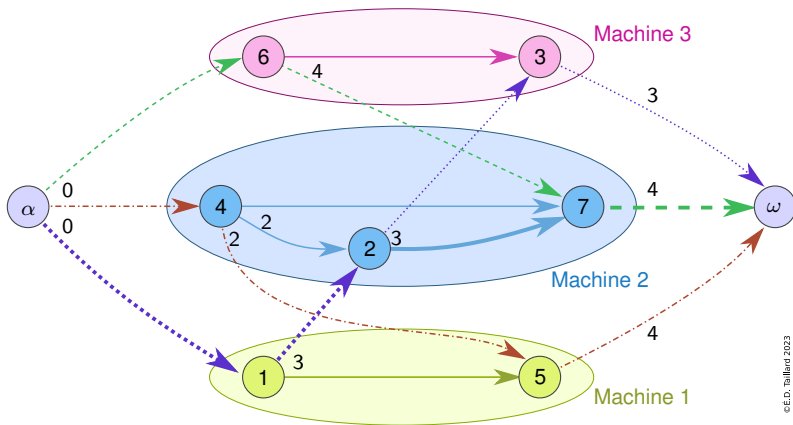
$$d_{ik} + t_{ik} \leq d_{ij} + M \cdot y_{jk} \quad (i = 1, \dots, m, j = 1 \dots n, j < k = 2 \dots n)$$

$$d_{ij} \geq 0 \quad (i = 1, \dots, m, j = 1 \dots n)$$

$$y_{jk} \in \{0, 1\} \quad (j = 1 \dots n, j < k = 2 \dots n)$$

Job Shop Scheduling

Example of modelling with 3 machines and 3 objects



Problem: Orient the edges of the cliques associated with the operations on each machine to minimize the longest path from α to ω

PL formulation of the Job Shop Scheduling

Minimize d_ω

$$d_i + t_i \leq d_j \quad \forall (i, j)$$

$$d_i + t_i \leq d_\omega \quad \forall i$$

$$d_i + t_i \leq d_k + M \cdot (1 - y_{ik}) \quad \forall i, k \text{ on the same machine}$$

$$d_k + t_k \leq d_i + M \cdot y_{ik} \quad \forall i, k \text{ on the same machine}$$

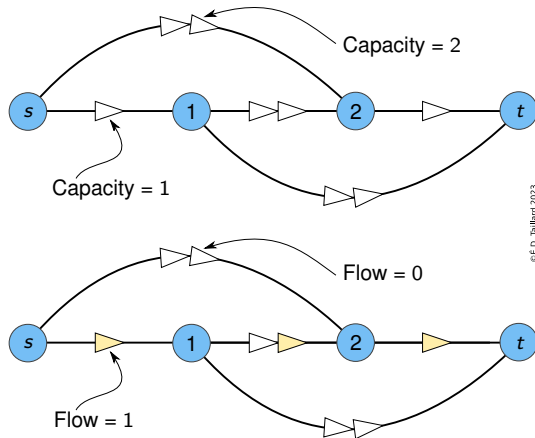
$$d_i \geq 0 \quad \forall i$$

$$y_{ik} \in \{0, 1\} \quad \forall i, k \text{ on the same machine}$$

2.4 Flows in networks

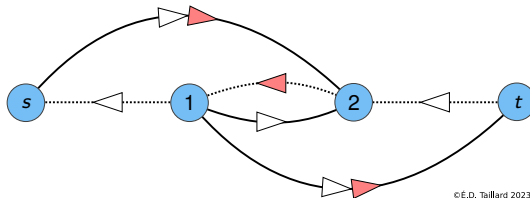
Search for a maximum flow (1): blocking flow

By increasing the flow by one unit on the path $s - 1 - 2 - t$, there is no longer a path with a possible direct increase



Searching for a Maximum Flow (2): Residual Network

Idea: we can increase the flow from 2 to 1 by decreasing the flow from 1 to 2



Ford-Fulkerson Algorithm

Search for a maximum flow from s to t

Input: Oriented network $R = (V, E, w)$, a source-node s and a sink-node t

Result: Maximum flow from s to t

1 Starts with a null flow in all arcs

2 **repeat**

3 Build the residual network R^* corresponding to the current flow

4 **if** *There is a path from s to t in R^** **then**

5 Find the maximal possible flow from s to t in R^* along this path

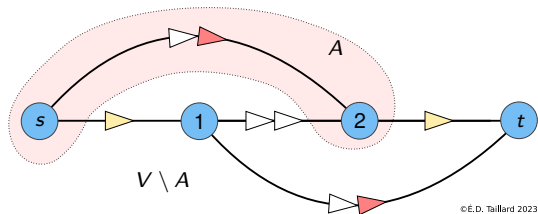
6 Superimpose this flow on the current flow (diminish the flow in the arcs (i, j) of R appearing as (j, i) arcs on the path in R^*)

7 **until** *there is no path from s to t in R^**

Local Improvement Algorithm

Searching for a maximum flow (3): proof of optimality

We can exhibit a subset A separating s from t such that the sum of the capacities leaving A is equal to the value of the flow found



Busacker-Gowen Algorithm for a Maximum Flow with Minimum Cost

Input: Oriented network $R = (V, E, w, c)$ without negative circuit, a source-node s and a sink-node t

Result: Maximum flow with minimum cost from s to t

1 Start with a null flow in all the arcs

2 **repeat**

3 Build the residual network R^* relative to the current flow

4 **if** *A path from s to t exists in R^** **then**

5 Find the maximal possible flow through the **shortest** path from s to t in R^*

6 Superimpose this flow on the current flow

7 **until** *there is no path from s to t in R^**

2.5 Assignment Problems

Linear Assignment in Terms of Linear Programming

Minimize
$$\sum_{i=1}^n \sum_{u=1}^n c_{iu} x_{iu}$$

Such that

$$\sum_{i=1}^n x_{iu} = 1 \quad u = 1, \dots, n$$

$$\sum_{u=1}^n x_{iu} = 1 \quad i = 1, \dots, n$$

$$x_{iu} \in \{0, 1\} \quad (i, u = 1, \dots, n)$$

The problem can be polynomially solved with a network flow model

Generalized Assignment

$$\text{Minimize} \quad \sum_{i=1}^n \sum_{u=1}^m c_{iu} x_{iu}$$

Such that

$$\sum_{i=1}^n w_{iu} x_{iu} \leq t_u \quad u = 1, \dots, m$$

$$\sum_{u=1}^m x_{iu} = 1 \quad i = 1, \dots, n$$

$$x_{iu} \in \{0, 1\} \quad (i, u = 1, \dots, m)$$

NP-hard

2.5 Knapsack

- Special case of the generalized assignment
- Objects of volume v_i with values c_i , ($i = 1, \dots, n$)
- Find the maximum value load of the knapsack
- Not exceeding the volume V of the sack
- The simplest *NP*-hard problem that can be expressed as a integer linear program:

$$\text{Maximize} \quad \sum_{i=1}^n c_i \cdot x_i$$

Such that

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

$$x_i \in \{0, 1\} \quad (i = 1, \dots, n)$$

2.5 Knapsack

- Special case of the generalized assignment
- Objects of volume v_i with values c_i , ($i = 1, \dots, n$)
- Find the maximum value load of the knapsack
- Not exceeding the volume V of the sack
- The simplest *NP*-hard problem that can be expressed as a integer linear program:

$$\text{Maximize} \quad \sum_{i=1}^n c_i \cdot x_i$$

Such that

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

$$x_i \in \{0, 1\} \quad (i = 1, \dots, n)$$

2.5 Knapsack

- Special case of the generalized assignment
- Objects of volume v_i with values c_i , ($i = 1, \dots, n$)
- Find the maximum value load of the knapsack
- Not exceeding the volume V of the sack
- The simplest *NP*-hard problem that can be expressed as a integer linear program:

$$\text{Maximize} \quad \sum_{i=1}^n c_i \cdot x_i$$

Such that

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

$$x_i \in \{0, 1\} \quad (i = 1, \dots, n)$$

2.5 Knapsack

- Special case of the generalized assignment
- Objects of volume v_i with values c_i , ($i = 1, \dots, n$)
- Find the maximum value load of the knapsack
- Not exceeding the volume V of the sack
- The simplest *NP*-hard problem that can be expressed as a integer linear program:

$$\text{Maximize} \quad \sum_{i=1}^n c_i \cdot x_i$$

Such that

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

$$x_i \in \{0, 1\} \quad (i = 1, \dots, n)$$

2.5 Knapsack

- Special case of the generalized assignment
- Objects of volume v_i with values c_i , ($i = 1, \dots, n$)
- Find the maximum value load of the knapsack
- Not exceeding the volume V of the sack
- The simplest *NP*-hard problem that can be expressed as a integer linear program:

$$\text{Maximize} \quad \sum_{i=1}^n c_i \cdot x_i$$

Such that

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

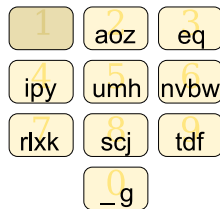
$$x_i \in \{0, 1\} \quad (i = 1, \dots, n)$$

Standard and optimised mobile phone keyboard

- 1 unit of time to press a key
- 2 time units for moving from one key to another
- Wait 6 time units to enter a new symbol on the same key



(a) Standard keyboard



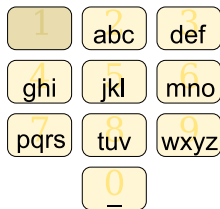
(b) Optimized keyboard

©É.D. Taillard 2023

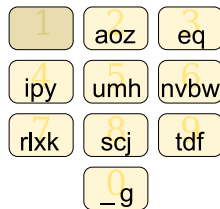
- 70 units of time to type “a ce soir bisous”
- Only 51 time units with an optimised keyboard

Standard and optimised mobile phone keyboard

- 1 unit of time to press a key
- 2 time units for moving from one key to another
- Wait 6 time units to enter a new symbol on the same key



(a) Standard keyboard



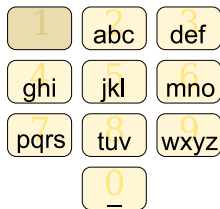
(b) Optimized keyboard

©É.D. Taillard 2023

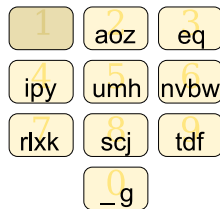
- 70 units of time to type “a ce soir bisous”
- Only 51 time units with an optimised keyboard

Standard and optimised mobile phone keyboard

- 1 unit of time to press a key
- 2 time units for moving from one key to another
- Wait 6 time units to enter a new symbol on the same key



(a) Standard keyboard



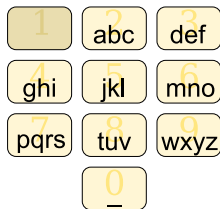
(b) Optimized keyboard

©É.D. Taillard 2023

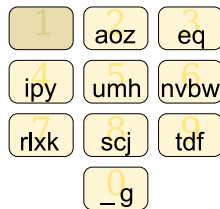
- 70 units of time to type “a ce soir bisous”
- Only 51 time units with an optimised keyboard

Standard and optimised mobile phone keyboard

- 1 unit of time to press a key
- 2 time units for moving from one key to another
- Wait 6 time units to enter a new symbol on the same key



(a) Standard keyboard



(b) Optimized keyboard

©É.D. Taillard 2023

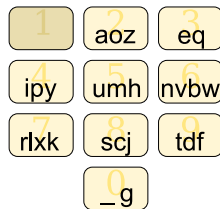
- 70 units of time to type “a ce soir bisous”
- Only 51 time units with an optimised keyboard

Standard and optimised mobile phone keyboard

- 1 unit of time to press a key
- 2 time units for moving from one key to another
- Wait 6 time units to enter a new symbol on the same key



(a) Standard keyboard



(b) Optimized keyboard

©É.D. Taillard 2023

- 70 units of time to type “a ce soir bisous”
- Only 51 time units with an optimised keyboard

Quadratic Assignment Problem

$$\text{Minimize} \quad \sum_{i=1}^n \sum_{j=1}^n \sum_{u=1}^n \sum_{v=1}^n a_{ij} b_{uv} x_{iu} x_{jv}$$

Such that

$$\sum_{i=1}^n x_{iu} = 1 \quad u = 1, \dots, n$$

$$\sum_{u=1}^n x_{iu} = 1 \quad i = 1, \dots, n$$

$$x_{iu} \in \{0, 1\} \quad (i, u = 1, \dots, n)$$

Other formulation: find a permutation p of n elements minimizing $\sum_{i=1}^n \sum_{j=1}^n a_{ij} \cdot b_{p_i p_j}$

- a_{ij} Frequency of occurrence of the symbol j after the symbol i in a sample text
- b_{uv} Delay between typing a symbol in position u and another in position v

2.7 Clustering

2.7 Clustering

- Set of elements E
- Dissimilarity measure $d(i, j) \geq 0$ between elements i and $j \in E$
- Group similar elements and separate dissimilar elements

Homogeneity measure of a cluster $G \subset E$

Diameter Maximal value $d(i, j)$ for 2 elements i and $j \in G$: $\max_{i, j \in G} d(i, j)$

Radius Maximal dissimilarity between the most central element and another in G :
 $\min_{i \in G} \max_{j \in G} d(i, j)$

Clique Sum of dissimilarities between all pair of elements of G : $\sum_{i \in G} \sum_{j \in G} d(i, j)$

Star Sum of dissimilarities between the most central element of G and the others:
 $\min_i \sum_{j \in G} d(i, j)$

2.7 Clustering

- Set of elements E
- Dissimilarity measure $d(i, j) \geq 0$ between elements i and $j \in E$
- Group similar elements and separate dissimilar elements

Homogeneity measure of a cluster $G \subset E$

Diameter Maximal value $d(i, j)$ for 2 elements i and $j \in G$: $\max_{i, j \in G} d(i, j)$

Radius Maximal dissimilarity between the most central element and another in G :
 $\min_{i \in G} \max_{j \in G} d(i, j)$

Clique Sum of dissimilarities between all pair of elements of G : $\sum_{i \in G} \sum_{j \in G} d(i, j)$

Star Sum of dissimilarities between the most central element of G and the others:
 $\min_i \sum_{j \in G} d(i, j)$

2.7 Clustering

- Set of elements E
- Dissimilarity measure $d(i, j) \geq 0$ between elements i and $j \in E$
- Group similar elements and separate dissimilar elements

Homogeneity measure of a cluster $G \subset E$

Diameter Maximal value $d(i, j)$ for 2 elements i and $j \in G$: $\max_{i, j \in G} d(i, j)$

Radius Maximal dissimilarity between the most central element and another in G :
 $\min_{i \in G} \max_{j \in G} d(i, j)$

Clique Sum of dissimilarities between all pair of elements of G : $\sum_{i \in G} \sum_{j \in G} d(i, j)$

Star Sum of dissimilarities between the most central element of G and the others:
 $\min_i \sum_{j \in G} d(i, j)$

2.7 Clustering

- Set of elements E
- Dissimilarity measure $d(i, j) \geq 0$ between elements i and $j \in E$
- Group similar elements and separate dissimilar elements

Homogeneity measure of a cluster $G \subset E$

Diameter Maximal value $d(i, j)$ for 2 elements i and $j \in G$: $\max_{i, j \in G} d(i, j)$

Radius Maximal dissimilarity between the most central element and another in G :
 $\min_{i \in G} \max_{j \in G} d(i, j)$

Clique Sum of dissimilarities between all pair of elements of G : $\sum_{i \in G} \sum_{j \in G} d(i, j)$

Star Sum of dissimilarities between the most central element of G and the others:
 $\min_i \sum_{j \in G} d(i, j)$

2.7 Clustering

- Set of elements E
- Dissimilarity measure $d(i, j) \geq 0$ between elements i and $j \in E$
- Group similar elements and separate dissimilar elements

Homogeneity measure of a cluster $G \subset E$

Diameter Maximal value $d(i, j)$ for 2 elements i and $j \in G$: $\max_{i, j \in G} d(i, j)$

Radius Maximal dissimilarity between the most central element and another in G :
 $\min_{i \in G} \max_{j \in G} d(i, j)$

Clique Sum of dissimilarities between all pair of elements of G : $\sum_{i \in G} \sum_{j \in G} d(i, j)$

Star Sum of dissimilarities between the most central element of G and the others:
 $\min_i \sum_{j \in G} d(i, j)$

2.7 Clustering

- Set of elements E
- Dissimilarity measure $d(i, j) \geq 0$ between elements i and $j \in E$
- Group similar elements and separate dissimilar elements

Homogeneity measure of a cluster $G \subset E$

Diameter Maximal value $d(i, j)$ for 2 elements i and $j \in G$: $\max_{i, j \in G} d(i, j)$

Radius Maximal dissimilarity between the most central element and another in G :
 $\min_{i \in G} \max_{j \in G} d(i, j)$

Clique Sum of dissimilarities between all pair of elements of G : $\sum_{i \in G} \sum_{j \in G} d(i, j)$

Star Sum of dissimilarities between the most central element of G and the others:
 $\min_i \sum_{j \in G} d(i, j)$

2.7 Clustering

- Set of elements E
- Dissimilarity measure $d(i, j) \geq 0$ between elements i and $j \in E$
- Group similar elements and separate dissimilar elements

Homogeneity measure of a cluster $G \subset E$

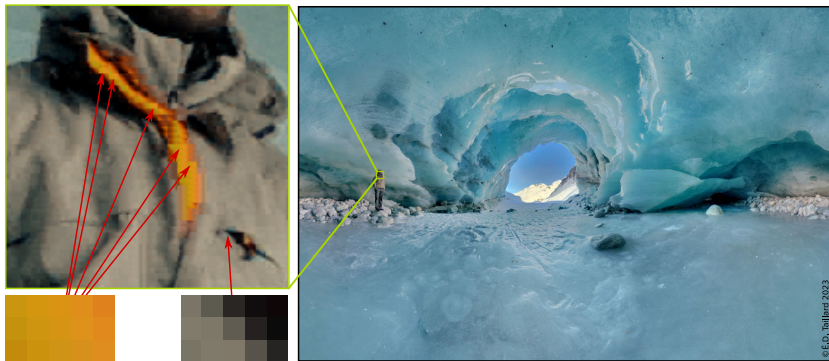
Diameter Maximal value $d(i, j)$ for 2 elements i and $j \in G$: $\max_{i, j \in G} d(i, j)$

Radius Maximal dissimilarity between the most central element and another in G :
 $\min_{i \in G} \max_{j \in G} d(i, j)$

Clique Sum of dissimilarities between all pair of elements of G : $\sum_{i \in G} \sum_{j \in G} d(i, j)$

Star Sum of dissimilarities between the most central element of G and the others:
 $\min_i \sum_{j \in G} d(i, j)$

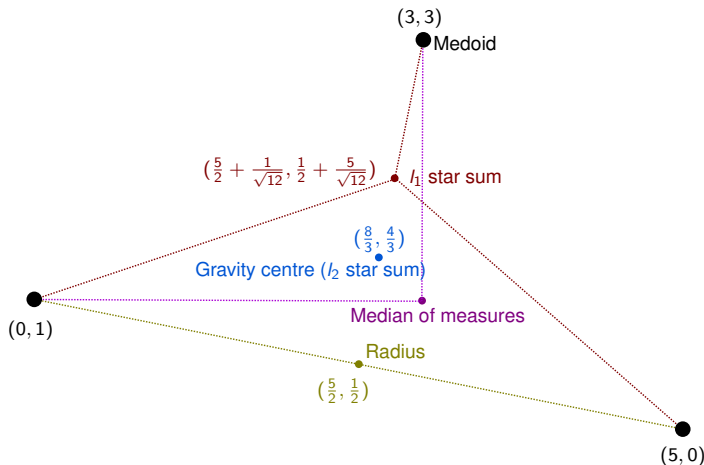
Example of a large unsupervised clustering problem: compression by quantization



With blocks of 15 pixels, clustering over millions of elements and $2^{14} = 16384$ centres, we can build a decent image with less than one bit per pixel

Position of optimal points according to the dissimilarity measure

For the points of the Euclidean plane: $(0, 1)$, $(3, 3)$, $(5, 0)$



© E.D. Taillard 2023

Heterogeneity measures between two clusters G and H

Separation Minimum distance between two elements belonging to different groups:

$$\min_{i \in G, j \in H} d(i, j)$$

Cut Sum of dissimilarities between elements of two different clusters:

$$\sum_{i \in G} \sum_{j \in H} d(i, j)$$

Normalized cut Average dissimilarity between elements of two different clusters:

$$\sum_{i \in G} \sum_{j \in H} d(i, j) / (|G| \cdot |H|)$$

Heterogeneity measures between two clusters G and H

Separation Minimum distance between two elements belonging to different groups:

$$\min_{i \in G, j \in H} d(i, j)$$

Cut Sum of dissimilarities between elements of two different clusters:

$$\sum_{i \in G} \sum_{j \in H} d(i, j)$$

Normalized cut Average dissimilarity between elements of two different clusters:

$$\sum_{i \in G} \sum_{j \in H} d(i, j) / (|G| \cdot |H|)$$

Heterogeneity measures between two clusters G and H

Separation Minimum distance between two elements belonging to different groups:

$$\min_{i \in G, j \in H} d(i, j)$$

Cut Sum of dissimilarities between elements of two different clusters:

$$\sum_{i \in G} \sum_{j \in H} d(i, j)$$

Normalized cut Average dissimilarity between elements of two different clusters:

$$\sum_{i \in G} \sum_{j \in H} d(i, j) / (|G| \cdot |H|)$$

Global Clustering Criteria

- Maximize the smallest separation (or the smallest cut) between elements of 2 different clusters
- Minimize the largest diameter (or the largest radius, or the largest clique or the largest star) of a cluster
- Minimize the star sum (or the diameter, radius, clique sum)

Global Clustering Criteria

- Maximize the smallest separation (or the smallest cut) between elements of 2 different clusters
- Minimize the largest diameter (or the largest radius, or the largest clique or the largest star) of a cluster
- Minimize the star sum (or the diameter, radius, clique sum)

Global Clustering Criteria

- Maximize the smallest separation (or the smallest cut) between elements of 2 different clusters
- Minimize the largest diameter (or the largest radius, or the largest clique or the largest star) of a cluster
- Minimize the star sum (or the diameter, radius, clique sum)

k -Medoids or p -Median: Partition Around Medoids (PAM)

Minimize the star sum relatively to elements of E

Input: Set E of items with a dissimilarity function $d(i, j)$ between items i and j ; k medoids $c_1, \dots, c_k \in E$

Result: Clusters $G_1, \dots, G_k \subset E$

```
1 repeat
2   forall item  $i \in E$  do
3     | Assign  $i$  to the closest medoid, creating clusters  $G_1, \dots, G_k \subset E$ 
4   forall medoid  $c_j$  do
5     | forall item  $i \in E$  do
6       | | Compute the improvement (or the loss) of a solution where  $c_j$  is moved on item  $i$ 
7   if A strictly positive improvement is found then
8     | Move the medoid on the item inducing the largest improvement
9 until no strict improvement is found
```

Local Improvement Algorithm

k -Means

Minimize the star sum (square distance) relatively to gravity center of clusters

Input: Set E of items in \mathbb{R}^d with l_2 norm measuring the dissimilarity between items; k centres $c_1, \dots, c_k \in \mathbb{R}^d$

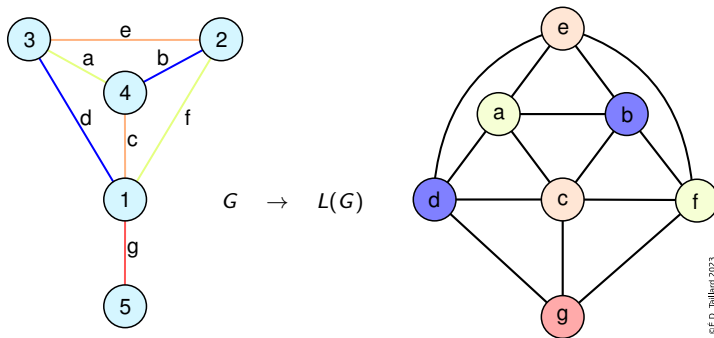
Result: Clusters $G_1, \dots, G_k \subset E$

```
1 repeat
2   forall item  $i \in E$  do
3     | Assign each item  $i \in E$  to its nearest centre, creating clusters  $G_1, \dots, G_k \subset E$ 
4   forall  $j \in 1, \dots, k$  do
5     |  $c_j$  = gravity center of  $G_j$ 
6 until no centre has moved
```

Local Improvement Algorithm

2.8 Graph Colouring Problem

- Either the vertices or the edges of a graph can be coloured
- Edge colouring can be transformed into vertex colouring of the corresponding line graph

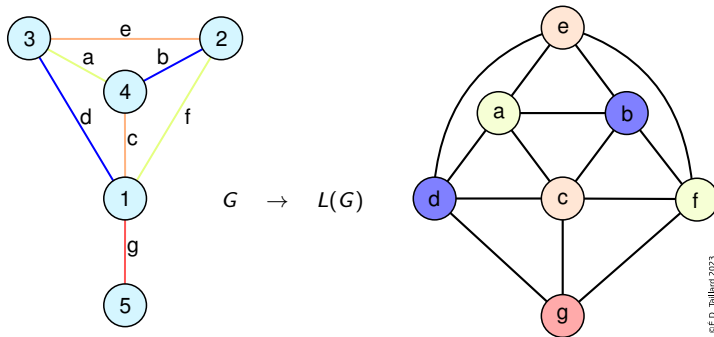


NP-hard problem

However, colouring the edges of a bipartite graph is a simple problem

2.8 Graph Colouring Problem

- Either the vertices or the edges of a graph can be coloured
- Edge colouring can be transformed into vertex colouring of the corresponding line graph



NP-hard problem

However, colouring the edges of a bipartite graph is a simple problem

Chapter 3

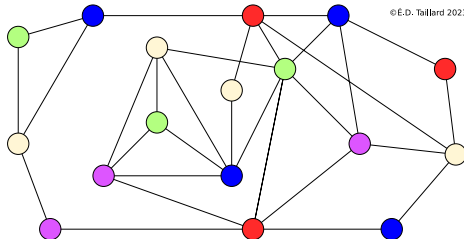
Problem Modelling

Chapter Content

| | | |
|---|---|----|
| 3 | Problem Modelling | 68 |
| • | Objective Function and Fitness Function | 70 |
| • | Lagrangian Relaxation | |
| • | Multi-Objectives Optimization | 78 |
| • | Pareto Set | |
| • | Scalarizing | |
| • | Sub-goals to Reach | |
| • | Practical Applications Modelled as Classical Problems | 84 |
| • | Linear Assignment Modelled by Minimum Cost Flow | |
| • | Map Labelling Modelled by Stable Set | |

3.1 Objective Function and Fitness Function

Natural objective function for graph colouring

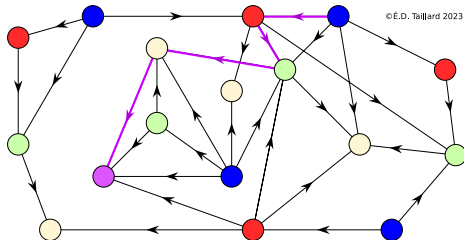


Minimize : $c = f_1(s)$

S.t. : 2 adjacent vertices have different colours

and : Number of colours used by $s \leq c$

Less Intuitive Fitness Function: Edge Orientation



Minimize : $f_2(s) = \text{Longest Path in } G \text{ oriented}$

S.t. : The edge orientation G has no circuit

Lagrangian Relaxation

Principle

- Introduce one or more constraints into the objective

Minimize $f(s)$

S.t. $s \in S$ Easy constraint

and $g(s) \leq 0$ Makes the problem hard

- With a penalty λ

Minimize $f(s) + \lambda \cdot \max(g(s), 0)$ with λ , a parameter

S.t. $s \in S$

Lagrangian Relaxation

Principle

- Introduce one or more constraints into the objective

Minimize $f(s)$

S.t. $s \in S$ Easy constraint

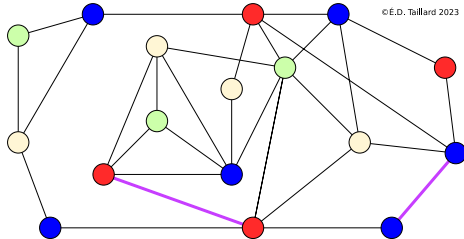
and $g(s) \leq 0$ Makes the problem hard

- With a penalty λ

Minimize $f(s) + \lambda \cdot \max(g(s), 0)$ with λ , a parameter

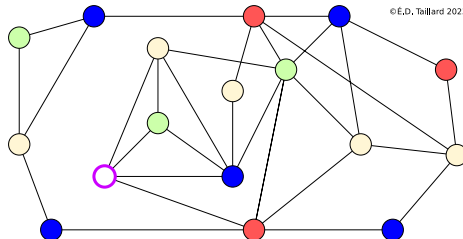
S.t. $s \in S$

Fitness Function Obtained with Lagrangean Relaxation



Minimize : $f_3(s) = c + \lambda \cdot \text{Number of edges with both end with the same colour}$
S.t. : Number of colours used $s \leq c$

Partial Colouring



Minimize : $f_4(s) = \text{Number of uncoloured vertices}$

S.t. : 2 adjacent vertices have different colours

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Lagrangian Relaxation for the TSP with 1-tree

- A 1-tree in a graph with vertices numbered from 1 to n is a tree on vertices 2 to n with 2 edges added from vertex 1
- TSP modelling in terms of 1-tree

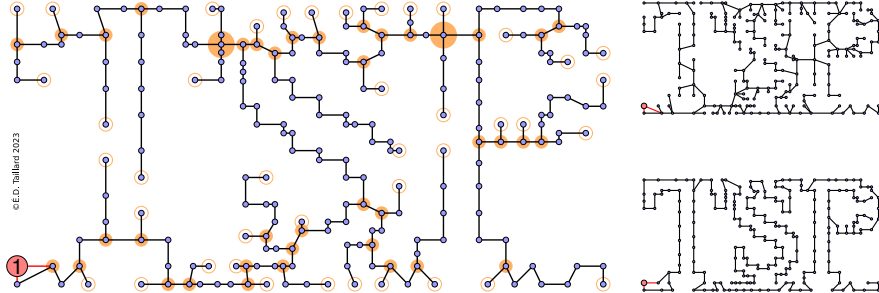
$$\begin{array}{ll} \min z = & \sum_{(i,j) \in H} d_{ij} x_{ij} \\ \text{S.t.} & \sum_{j=1}^n x_{ij} = 2 \quad (i = 1, \dots, n) \\ \text{et} & H \text{ is a 1-tree} \end{array}$$

- Lagrangian relaxation:

$$\begin{array}{ll} \min z(\lambda) = & \sum_{(i,j) \in H} d_{ij} x_{ij} + \sum_{i=1}^n \lambda_i (\sum_{j=1}^n x_{ij} - 2) \\ \text{S.t.} & H \text{ is a 1-tree} \end{array}$$

- The objective can be rewritten with modified distances $d_{ij} + \lambda_i + \lambda_j$
- For λ known, the problem is to find a minimum 1-tree
- $z(\lambda)$ provides a lower bound to the TSP tour
- If the 1-tree is not a tour for given λ , the lasts can be modified
 - Increase λ_i if the degree of vertex i is larger than 2
 - Decrease λ_i if the degree is 1

Applying the 1-tree relaxation on the TSP



- Left: 1-tree on the TSP instance *tsp225* obtained with all λ values set to 0
- Top right: the 1-tree obtained with the first modification of the λ
- Bottom right: the 1-tree obtained after iterating the process

3.2 Multi-Objectives Optimization

Multi-Objectives Models

K objectives

- $f_1(s), f_2(s), \dots, f_K(s)$
- The objectives are partially antagonist (price/time/durability)

“Minimize” : $\vec{f}(s) = (f_1(s), \dots, f_K(s))$

S.t. $s \in S$

- A solution s_1 **dominate** a solution s_2 if s_1 is better than s_2 on one objective at least (it can be worse than s_2 on the other objectives)
- A non-dominated solution is **Pareto-optimal** or **efficient**
- A solution on the convex hull of the Pareto set is **supported**

Multi-Objectives Models

K objectives

- $f_1(s), f_2(s), \dots, f_K(s)$
- The objectives are partially antagonist (price/time/durability)

“Minimize” : $\vec{f}(s) = (f_1(s), \dots, f_K(s))$

S.t. $s \in S$

- A solution s_1 **dominate** a solution s_2 if s_1 is better than s_2 on one objective at least (it can be worse than s_2 on the other objectives)
- A non-dominated solution is **Pareto-optimal** or **efficient**
- A solution on the convex hull of the Pareto set is **supported**

Multi-Objectives Models

K objectives

- $f_1(s), f_2(s), \dots, f_K(s)$
- The objectives are partially antagonist (price/time/durability)

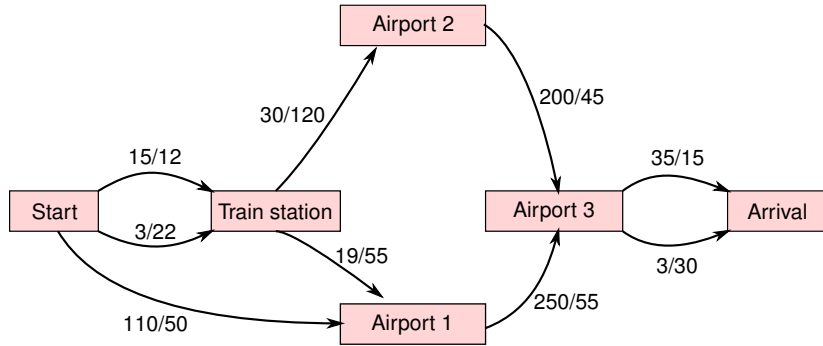
“Minimize” : $\vec{f}(s) = (f_1(s), \dots, f_K(s))$

S.t. $s \in S$

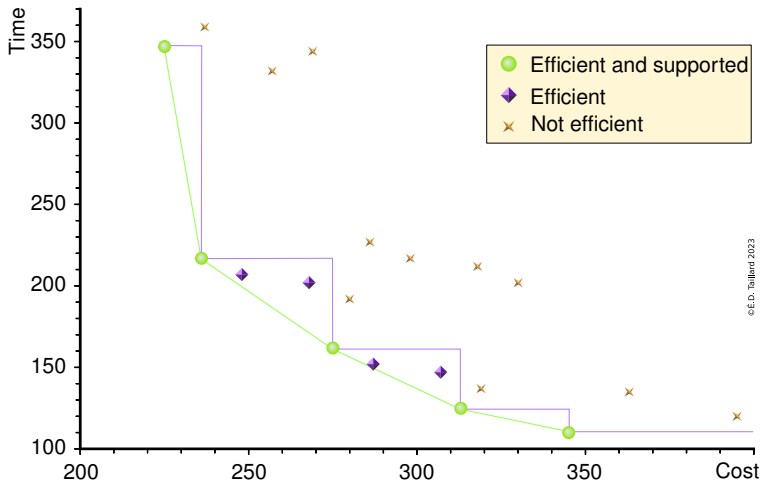
- A solution s_1 **dominate** a solution s_2 if s_1 is better than s_2 on one objective at least (it can be worse than s_2 on the other objectives)
- A non-dominated solution is **Pareto-optimal** or **efficient**
- A solution on the convex hull of the Pareto set is **supported**

Example: Cheap and Short Path

- Cost of the path
- Duration of the path



Representation of the solutions in a cost/time diagram



© E.D. Taillard 2023

Scalarizing

Reduce to a single-objective problem by assigning weights w_1, \dots, w_K for each objective

$$\begin{aligned} \text{Minimize : } & \sum_{i=1}^K w_i \cdot f_i(s) \\ \text{S.t. } & s \in S \end{aligned}$$

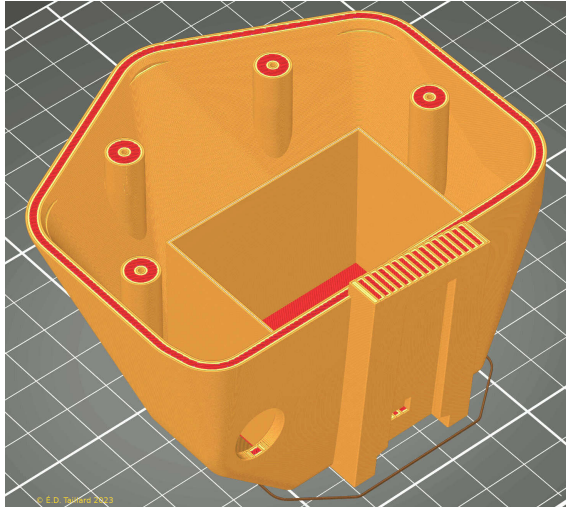
Shortcoming: Unsupported effective solutions are not achievable

Sub-goals to Reach

$$\begin{array}{ll}\text{Minimize :} & \vec{f}(s) = (f_2(s), \dots, f_K(s)) \\ \text{S.t.} & f_1(s) \leq v_1 \\ \text{with} & s \in S\end{array}$$

3.3 Practical Applications Modelled as Classical Problems

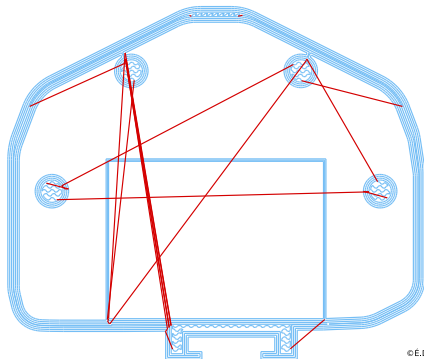
Industrial TSP Application: 3D-Printing



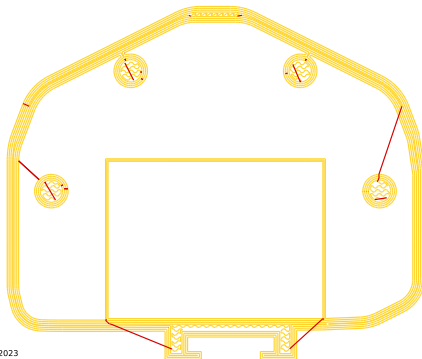
3D-Printing

Improductive moves of the extrusion head

Solution obtained with *PrusaSlicer* and possible improvement by solving a TSP



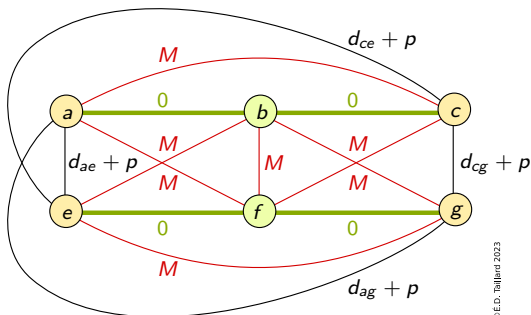
©E.D. Taillard 2023



Minimising unproductive moves in 3D printing

Representation of the printing of 2 segments with ends $a - c$ and $e - g$

- Force the head to print the segment by introducing an intermediate node reasonably accessible only from the ends of the segment
- Pay a penalty (e.g. $p = M/10$) if we change the layer
- A Hamiltonian path (passing through all vertices) of minimum cost corresponds to the printing of all segments; all segments of a layer are printed before moving to another layer; unproductive moves is minimized

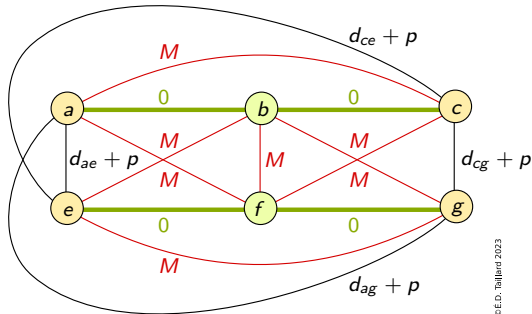


© E.D. Taillard 2023

Minimising unproductive moves in 3D printing

Representation of the printing of 2 segments with ends $a - c$ and $e - g$

- Force the head to print the segment by introducing an intermediate node reasonably accessible only from the ends of the segment
- Pay a penalty (e.g. $p = M/10$) if we change the layer
- A Hamiltonian path (passing through all vertices) of minimum cost corresponds to the printing of all segments; all segments of a layer are printed before moving to another layer; unproductive moves is minimized

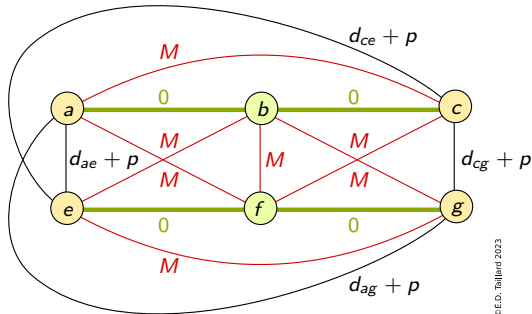


© E.D. Taillard 2023

Minimising unproductive moves in 3D printing

Representation of the printing of 2 segments with ends $a - c$ and $e - g$

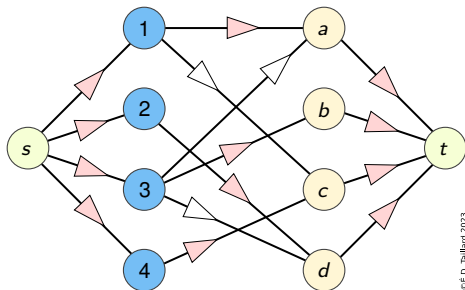
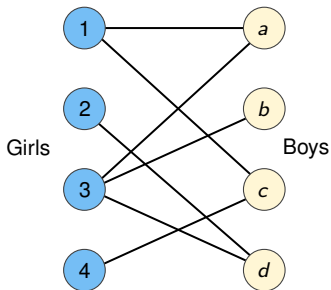
- Force the head to print the segment by introducing an intermediate node reasonably accessible only from the ends of the segment
- Pay a penalty (e.g. $p = M/10$) if we change the layer
- A Hamiltonian path (passing through all vertices) of minimum cost corresponds to the printing of all segments; all segments of a layer are printed before moving to another layer; unproductive moves is minimized



© E.D. Taillard 2023

Linear Assignment Modelled by Minimum Cost Flow

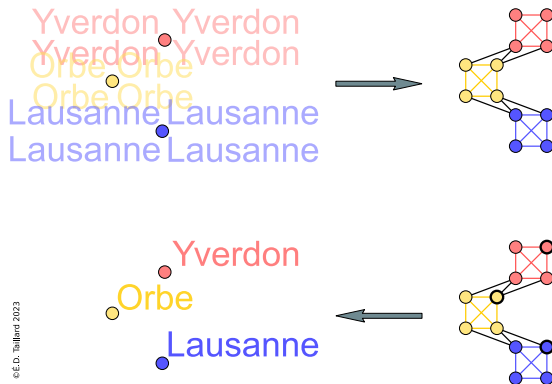
What is the maximum number of pairs that can be formed between girls and boys?
A compatible pair is represented by an edge in a bipartite graph



©É.D. Taillard 2023

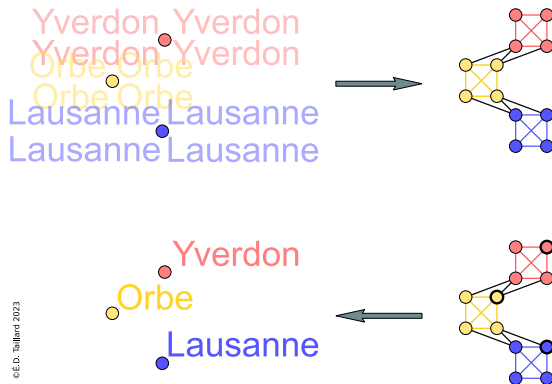
Map Labelling Modelled by Stable Set

- Create as many vertices as there are possible positions for each object to label
- Connect 2 incompatible vertices with an edge



Map Labelling Modelled by Stable Set

- Create as many vertices as there are possible positions for each object to label
- Connect 2 incompatible vertices with an edge



Chapter 4

Constructive Methods

Chapter Content

| | | |
|---|---------------------------------------|-----|
| 4 | Constructive Methods..... | 90 |
| • | Systematic Enumeration..... | 92 |
| • | Branch and Bound | |
| • | Random Building..... | 97 |
| • | Greedy Algorithms..... | 101 |
| • | Greedy Heuristics for the TSP | |
| • | Greedy Heuristics for Graph Colouring | |
| • | Improvement of Greedy Procedures..... | 109 |
| • | Beam Search | |
| • | Pilot Method | |

4.1 Systematic Enumeration

Systematic Enumeration

- 0 – 1 knapsack instance with two constraints with n variables: has 2^n potential solutions

$$\begin{aligned}
 \max r = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 4x_1 + 3x_2 + 5x_3 + 2x_4 + x_5 \leq 10 \\
 & 4x_1 + 2x_2 + 3x_3 + 2x_4 + x_5 \leq 7 \\
 & x_i \in \{0, 1\} (i = 1, \dots, 5)
 \end{aligned}$$

- The solution can be enumerated by splitting the problem into two sub-problems

$$\begin{aligned}
 \max r = & 0 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 3x_2 + 5x_3 + 2x_4 + x_5 \leq 10 \\
 & 2x_2 + 3x_3 + 2x_4 + x_5 \leq 7 \\
 & x_i \in \{0, 1\} (i = 2, \dots, 5)
 \end{aligned}$$

$$\begin{aligned}
 \max r = & 9 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 3x_2 + 5x_3 + 2x_4 + x_5 \leq 6 \\
 & 2x_2 + 3x_3 + 2x_4 + x_5 \leq 3 \\
 & x_i \in \{0, 1\} (i = 2, \dots, 5)
 \end{aligned}$$

Systematic Enumeration

- 0 – 1 knapsack instance with two constraints with n variables: has 2^n potential solutions

$$\begin{aligned}
 \max r = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 4x_1 + 3x_2 + 5x_3 + 2x_4 + x_5 \leq 10 \\
 & 4x_1 + 2x_2 + 3x_3 + 2x_4 + x_5 \leq 7 \\
 & x_i \in \{0, 1\} (i = 1, \dots, 5)
 \end{aligned}$$

- The solution can be enumerated by splitting the problem into two sub-problems

$$\begin{aligned}
 \max r = & 0 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 3x_2 + 5x_3 + 2x_4 + x_5 \leq 10 \\
 & 2x_2 + 3x_3 + 2x_4 + x_5 \leq 7 \\
 & x_i \in \{0, 1\} (i = 2, \dots, 5)
 \end{aligned}$$

$$\begin{aligned}
 \max r = & 9 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 3x_2 + 5x_3 + 2x_4 + x_5 \leq 6 \\
 & 2x_2 + 3x_3 + 2x_4 + x_5 \leq 3 \\
 & x_i \in \{0, 1\} (i = 2, \dots, 5)
 \end{aligned}$$

Branch and Bound

Before separating, it is assessed whether the problem can potentially be solved by relaxing it

- Integrity:

$$\begin{array}{ll} \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\ \text{S.t.} & 4x_1 + 3x_2 + 5x_3 + 2x_4 + x_5 \leq 10 \\ & 4x_1 + 2x_2 + 3x_3 + 2x_4 + x_5 \leq 7 \\ & 0 \leq x_i \leq 1 (i = 1, \dots, 5) \end{array}$$

- Aggregation of constraints:

$$\begin{array}{ll} \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\ \text{S.t.} & 8x_1 + 5x_2 + 8x_3 + 4x_4 + 2x_5 \leq 17 \\ & x_i \in \{0, 1\} (i = 1, \dots, 5) \end{array}$$

- Combination of relaxations:

$$\begin{array}{ll} \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\ \text{S.t.} & 8x_1 + 5x_2 + 8x_3 + 4x_4 + 2x_5 \leq 17 \\ & 0 \leq x_i \leq 1 (i = 1, \dots, 5) \end{array}$$

Branch and Bound

Before separating, it is assessed whether the problem can potentially be solved by relaxing it

- Integrity:

$$\begin{aligned}
 \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 4x_1 + 3x_2 + 5x_3 + 2x_4 + x_5 \leq 10 \\
 & 4x_1 + 2x_2 + 3x_3 + 2x_4 + x_5 \leq 7 \\
 & 0 \leq x_i \leq 1 (i = 1, \dots, 5)
 \end{aligned}$$

- Aggregation of constraints:

$$\begin{aligned}
 \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 8x_1 + 5x_2 + 8x_3 + 4x_4 + 2x_5 \leq 17 \\
 & x_i \in \{0, 1\} (i = 1, \dots, 5)
 \end{aligned}$$

- Combination of relaxations:

$$\begin{aligned}
 \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t. } & 8x_1 + 5x_2 + 8x_3 + 4x_4 + 2x_5 \leq 17 \\
 & 0 \leq x_i \leq 1 (i = 1, \dots, 5)
 \end{aligned}$$

Branch and Bound

Before separating, it is assessed whether the problem can potentially be solved by relaxing it

- Integrity:

$$\begin{array}{ll}
 \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t.} & 4x_1 + 3x_2 + 5x_3 + 2x_4 + x_5 \leq 10 \\
 & 4x_1 + 2x_2 + 3x_3 + 2x_4 + x_5 \leq 7 \\
 & 0 \leq x_i \leq 1 (i = 1, \dots, 5)
 \end{array}$$

- Aggregation of constraints:

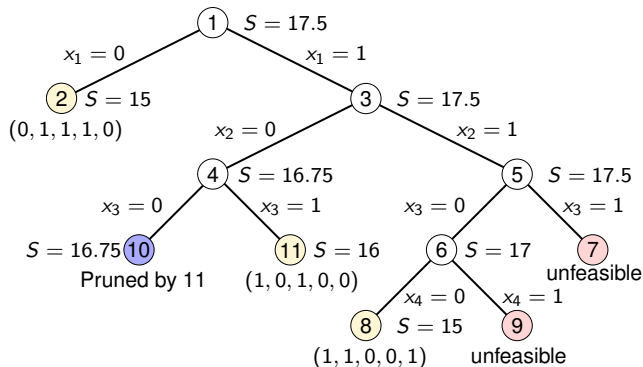
$$\begin{array}{ll}
 \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t.} & 8x_1 + 5x_2 + 8x_3 + 4x_4 + 2x_5 \leq 17 \\
 & x_i \in \{0, 1\} (i = 1, \dots, 5)
 \end{array}$$

- Combination of relaxations:

$$\begin{array}{ll}
 \max S = & 9x_1 + 5x_2 + 7x_3 + 3x_4 + x_5 \\
 \text{S.t.} & 8x_1 + 5x_2 + 8x_3 + 4x_4 + 2x_5 \leq 17 \\
 & 0 \leq x_i \leq 1 (i = 1, \dots, 5)
 \end{array}$$

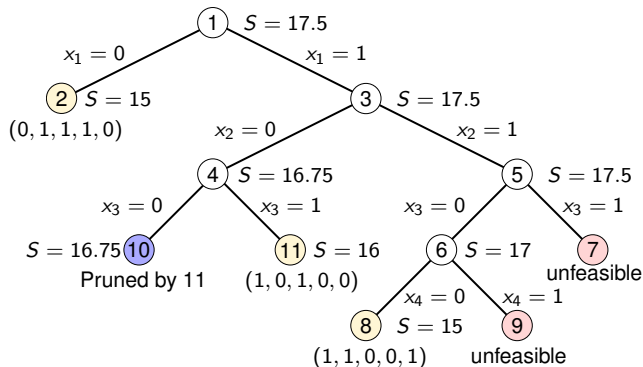
Policy Example of a Branch and Bound

- Q manage as a stack
- Branching according to the order of the indices
- Constraint aggregation and variable integrity relaxation



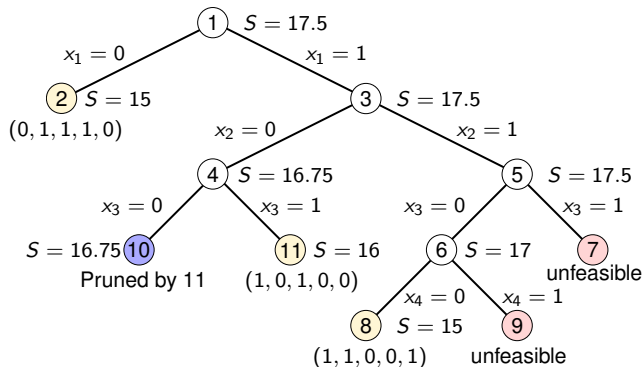
Policy Example of a Branch and Bound

- Q manage as a stack
- Branching according to the order of the indices
- Constraint aggregation and variable integrity relaxation



Policy Example of a Branch and Bound

- Q manage as a stack
- Branching according to the order of the indices
- Constraint aggregation and variable integrity relaxation



Branch and Bound Frame

Input: A problem with n variables x_1, \dots, x_n , policy α for managing sub-problems, relaxation method β , branching method

Result: An optimal solution x^* of maximal value f^*

```

1   $f^* \leftarrow -\infty$                                      // Value of best solution found
2   $F \leftarrow \emptyset$                                 // Set of fixed variables
3   $L \leftarrow \{x_1, \dots, x_n\}$                       // Set of free variables
4   $Q \leftarrow \{(F, L)\}$                              // Set of sub-problems to solve
5  while  $Q \neq \emptyset$  do
6      Remove a problem  $P = (F, L)$  from  $Q$  according to policy  $\alpha$ 
7      if  $P$  can potentially have feasible solutions with values already fixed in  $F$  then
8          Compute a solution  $x$  to a relaxation of  $P$  with method  $\beta$ , modifying only variables  $x_k \in L$ 
9          if  $x$  is feasible for the initial problem and  $f^* < f(x)$  then Store the improved solution
10              $x^* \leftarrow x$ 
11              $f^* \leftarrow f(x)$ 
12         else if  $f(x) > f^*$  then Expand the branch
13             Choose  $x_k \in L$  according to policy  $\gamma$ 
14             forall possible value  $v$  of  $x_k$  do
15                  $Q \leftarrow Q \cup \{(F \cup \{x_k = v\}, L \setminus \{x_k\})\}$ 
16             else No solution better than  $x^*$  can be obtained
17                 Prune the branch
18 
```

4.2 Random Building

Random Building

Idea

- Draw a solution randomly, uniformly in the space of admissible solutions
- Benefit: One of the simplest methods
- Drawbacks
 - A uniform construction is not necessarily easy to achieve
 - The quality of the solution is poor
- Example: Draw a random permutation of n elements

Random Building

Idea

- Draw a solution randomly, uniformly in the space of admissible solutions
- Benefit: One of the simplest methods
- Drawbacks
 - A uniform construction is not necessarily easy to achieve
 - The quality of the solution is poor
- Example: Draw a random permutation of n elements

Random Building

Idea

- Draw a solution randomly, uniformly in the space of admissible solutions
- Benefit: One of the simplest methods
- Drawbacks
 - A uniform construction is not necessarily easy to achieve
 - The quality of the solution is poor
- Example: Draw a random permutation of n elements

Random Building

Idea

- Draw a solution randomly, uniformly in the space of admissible solutions
- Benefit: One of the simplest methods
- Drawbacks
 - A uniform construction is not necessarily easy to achieve
 - The quality of the solution is poor
- Example: Draw a random permutation of n elements

Random Building

Idea

- Draw a solution randomly, uniformly in the space of admissible solutions
- Benefit: One of the simplest methods
- Drawbacks
 - A uniform construction is not necessarily easy to achieve
 - The quality of the solution is poor
- Example: Draw a random permutation of n elements

Random Building

Idea

- Draw a solution randomly, uniformly in the space of admissible solutions
- Benefit: One of the simplest methods
- Drawbacks
 - A uniform construction is not necessarily easy to achieve
 - The quality of the solution is poor
- Example: Draw a random permutation of n elements

Bad algorithm for constructing a random permutation (1)

Input: A set of n elements e_1, \dots, e_n

Result: A permutation p of the elements

```
1  $i \leftarrow 0$                                      // Number of element already chosen
2 while  $i \neq n$  do
3   Draw a random number  $u$  uniformly between 1 and  $n$ 
4   if  $e_u$  is not already chosen then
5      $i \leftarrow i + 1$ 
6      $p_i \leftarrow e_u$ 
```

Bad algorithm for constructing a random permutation (2)

Input: A set of n elements e_1, \dots, e_n

Result: A permutation p of the elements

```
1  $i \leftarrow 0$  // Number of element already chosen
2 while  $i \neq n$  do
3   Draw a random number  $u$  uniformly between 1 and  $n$ 
4    $i \leftarrow i + 1$ 
5   if  $e_u$  is already chosen then
6     Find the next  $u'$  such that  $e_{u'}$  is not chosen
7      $p_i \leftarrow e_{u'}$ 
8   else
9      $p_i \leftarrow e_u$ 
```

4.3 Greedy Algorithms

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

General idea of greedy methods

Imitating what works with Prim and Dijkstra

- Construct a solution, element by element
 - Add the element that seems most appropriate at each step
 - Do not question a choice
-
- Example of elements that can be added to a solution
 - A city for a travelling salesman
 - An edge for a travelling salesman or for a Steiner tree
 - A vertex with a given colour for graph colouring
 - Orientation of an edge for graph colouring
 - When adding an element e to a partial solution s , an incremental cost $c(s, e)$ is taken into account, and this can lead to restrictions on the elements that remain to be added

Greedy Algorithm

Input: A trivial partial solution s (generally \emptyset); set E of elements constituting a solution;
incremental cost function $c(s, e)$

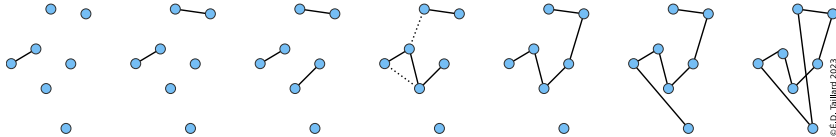
Result: Complete solution s

```
1  $R \leftarrow E$  // Elements that can be added to  $s$ 
2 while  $R \neq \emptyset$  do
3    $\forall e \in R$ , compute  $c(s, e)$ 
4   Choose  $e'$  optimizing  $c(s, e')$ 
5    $s \leftarrow s \cup e'$  // Include  $e'$  in the partial solution  $s$ 
6   Remove from  $R$  the elements that cannot be added any more to  $s$ 
```


Greedy on the Edge Weight for the TSP

Same as Kruskal, but without allowing a vertex degree larger than 2

- Element e to be added: an edge
- Incremental cost: edge cost e
- Elements to remove: do not create degree 3 vertices or sub-cycles

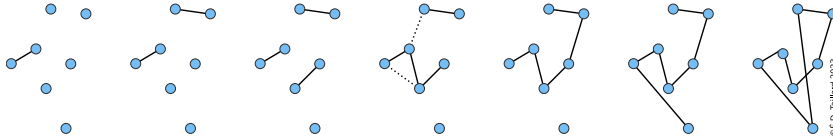


© E.D. Taillard 2023

Greedy on the Edge Weight for the TSP

Same as Kruskal, but without allowing a vertex degree larger than 2

- Element e to be added: an edge
- Incremental cost: edge cost e
- Elements to remove: do not create degree 3 vertices or sub-cycles

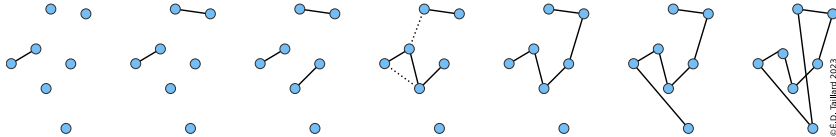


© E.D. Taillard 2023

Greedy on the Edge Weight for the TSP

Same as Kruskal, but without allowing a vertex degree larger than 2

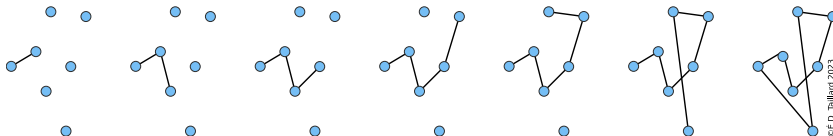
- Element e to be added: an edge
- Incremental cost: edge cost e
- Elements to remove: do not create degree 3 vertices or sub-cycles



Nearest Neighbour for the TSP

Same as Prim, but the next vertex must be connected to the last one added

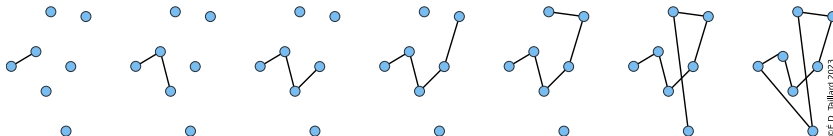
- Start from any city
- Element e to be added: a city
- Incremental cost: edge cost to reach e from the last added city



Nearest Neighbour for the TSP

Same as Prim, but the next vertex must be connected to the last one added

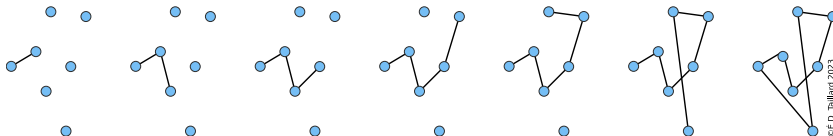
- Start from any city
- Element e to be added: a city
- Incremental cost: edge cost to reach e from the last added city



Nearest Neighbour for the TSP

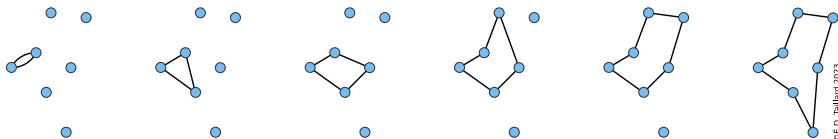
Same as Prim, but the next vertex must be connected to the last one added

- Start from any city
- Element e to be added: a city
- Incremental cost: edge cost to reach e from the last added city



Cheapest Insertion for the TSP

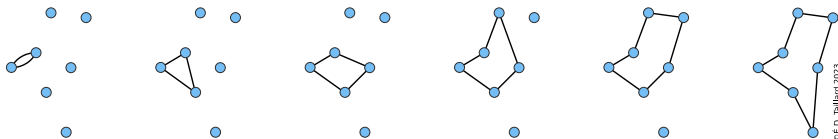
- Start from a 2-city tour
- Element e to be added: a city
- Incremental cost: cost of a minimum detour to add e to the partial tour
- Choose the city with the lowest incremental cost



©É.D. Taillard 2023

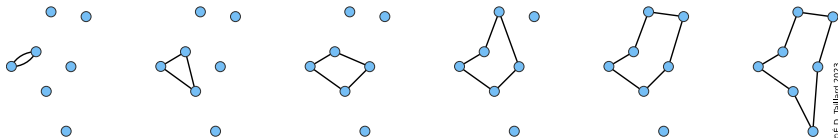
Cheapest Insertion for the TSP

- Start from a 2-city tour
- Element e to be added: a city
- Incremental cost: cost of a minimum detour to add e to the partial tour
- Choose the city with the lowest incremental cost



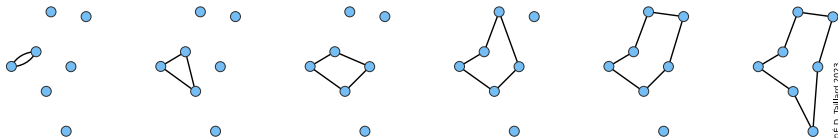
Cheapest Insertion for the TSP

- Start from a 2-city tour
- Element e to be added: a city
- Incremental cost: cost of a minimum detour to add e to the partial tour
- Choose the city with the lowest incremental cost



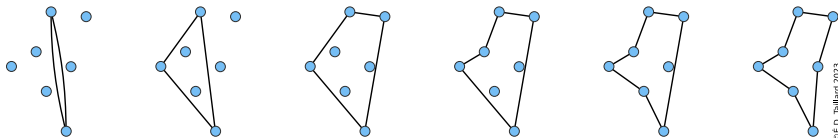
Cheapest Insertion for the TSP

- Start from a 2-city tour
- Element e to be added: a city
- Incremental cost: cost of a minimum detour to add e to the partial tour
- Choose the city with the lowest incremental cost



Farthest Insertion for the TSP

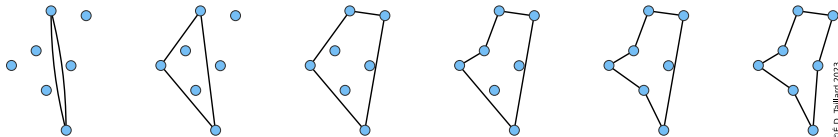
- Start from a 2-city tour
- Element e to be added: a city
- Incremental cost: cost of a minimum detour to add e to the partial tour
- Choose the city with the largest incremental cost



©É.D. Taillard 2023

Farthest Insertion for the TSP

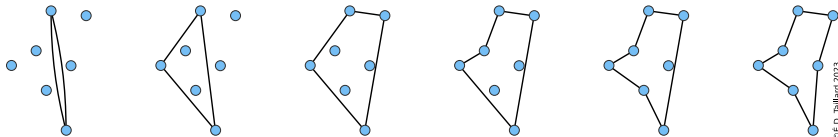
- Start from a 2-city tour
- Element e to be added: a city
- Incremental cost: cost of a minimum detour to add e to the partial tour
- Choose the city with the largest incremental cost



©É.D. Taillard 2023

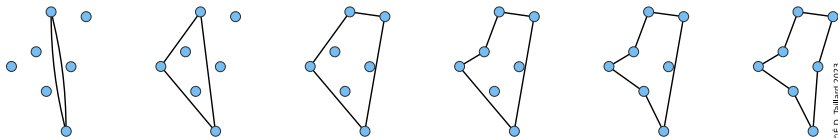
Farthest Insertion for the TSP

- Start from a 2-city tour
- Element e to be added: a city
- Incremental cost: cost of a minimum detour to add e to the partial tour
- Choose the city with the largest incremental cost



Farthest Insertion for the TSP

- Start from a 2-city tour
- Element e to be added: a city
- Incremental cost: cost of a minimum detour to add e to the partial tour
- Choose the city with the largest incremental cost



©É.D. Taillard 2023

DSATUR Heuristics for Graph Colouring

Idea: calculate the degree of saturation $DS(v)$ of each vertex v (the number of different colours used by adjacent vertices of v)

Input: Undirected graph $G = (V, E)$;

Result: Vertex colouring

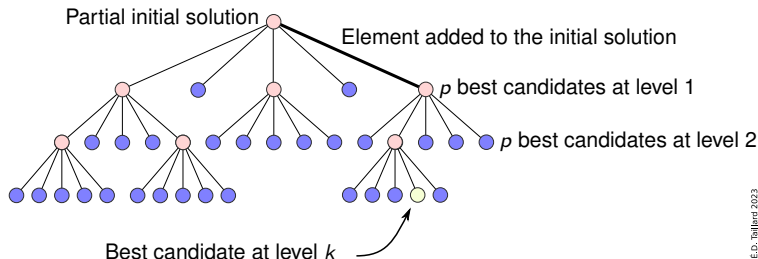
```
1 Colour with 1 the vertex  $v$  with the highest degree
2  $R \leftarrow V \setminus v$ 
3  $colours \leftarrow 1$ 
4 while  $R \neq \emptyset$  do
5    $\forall v \in R$ , compute  $DS(v)$ 
6   Choose  $v'$  maximizing  $DS(v')$ , with the highest possible degree
7   Find the smallest  $k$  ( $1 \leq k \leq colours + 1$ ) such that colour  $k$  is feasible for  $v'$ 
8   Assign colour  $k$  to  $v'$ 
9   if  $k > colours$  then
10      $colours = k$ 
11    $R \leftarrow R \setminus v'$ 
```

4.4 Improvement of Greedy Procedures

Beam Search

Identical to an enumeration method, but avoid node number exponential growth

- Select only the p best candidates at each level
- Stop k levels below the last added element
- Add the element that results in the best partial solution at the last level evaluated

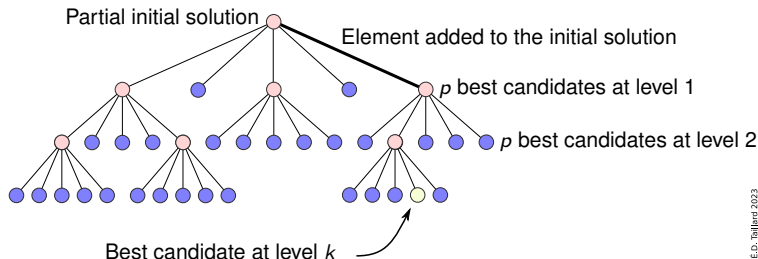


© E.D. Taillard 2023

Beam Search

Identical to an enumeration method, but avoid node number exponential growth

- Select only the p best candidates at each level
- Stop k levels below the last added element
- Add the element that results in the best partial solution at the last level evaluated

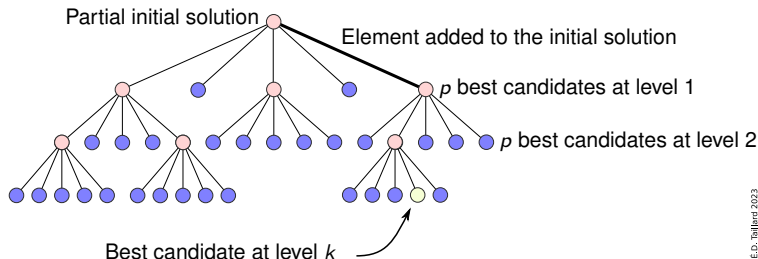


© E.D. Taillard 2023

Beam Search

Identical to an enumeration method, but avoid node number exponential growth

- Select only the p best candidates at each level
- Stop k levels below the last added element
- Add the element that results in the best partial solution at the last level evaluated



© E.D. Taillard 2023

Pilot Method

Try all possibilities to complete a partial solution with one element, then launch a heuristic method (pilot)

- The pilot heuristic starts with various partial solutions
- Run the pilot heuristic until complete solutions are obtained
- The element to be added to the partial solution is the one that led to the best complete solution

Pilot Method

Try all possibilities to complete a partial solution with one element, then launch a heuristic method (pilot)

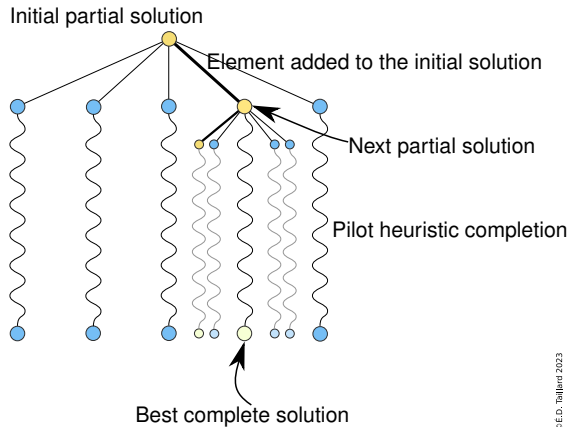
- The pilot heuristic starts with various partial solutions
- Run the pilot heuristic until complete solutions are obtained
- The element to be added to the partial solution is the one that led to the best complete solution

Pilot Method

Try all possibilities to complete a partial solution with one element, then launch a heuristic method (pilot)

- The pilot heuristic starts with various partial solutions
- Run the pilot heuristic until complete solutions are obtained
- The element to be added to the partial solution is the one that led to the best complete solution

Pilot Method: Illustration



© E.D. Taillard 2023

Pilot Method Algorithm

Input: s_p trivial partial solution; set E of elements constituting a solution; pilot heuristic $h(s_e)$ for completing a partial solution s_e ; fitness function $f(s)$

Result: Complete solution s^*

```

1  $R \leftarrow E$  // Elements that can be added to  $s$ 
2 while  $R \neq \emptyset$  do
3    $v \leftarrow \infty$ 
4   forall  $e \in R$  do
5     Complete  $s_p$  with  $e$  to get  $s_e$ 
6     Apply  $h(s_e)$  to get a complete solution  $s$ 
7     if  $f(s) \leq v$  then
8        $v \leftarrow f(s)$ 
9        $s_r \leftarrow s_e$ 
10      if  $s$  is better than  $s^*$  then Store the improved solution
11       $s^* \leftarrow s$ 

```


Chapter 5

Local Search

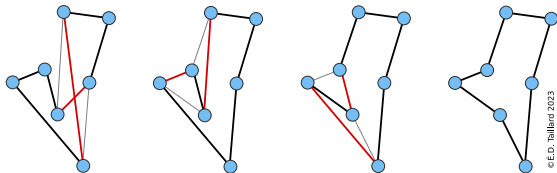
Chapter Content

| | | |
|---|---------------------------------------|-----|
| 5 | Local Search | 114 |
| • | Local Search Frame | 117 |
| • | Policy of the first improving move | |
| • | Best Move Policy | |
| • | Local, Global Minima, Plateau | |
| • | Neighbourhood Properties | |
| • | Limiting the Neighbourhood Size | 128 |
| • | Candidate List | |
| • | Neighbourhood Extension | 130 |
| • | Filter and Fan | |
| • | Ejection Chains | |
| • | Multi-Objective Local Search | 137 |
| • | Scalarization | |
| • | Pareto Local Search | |

Local Search

General Idea

- Start from a solution obtained with a constructive method
- Modify it locally to improve it

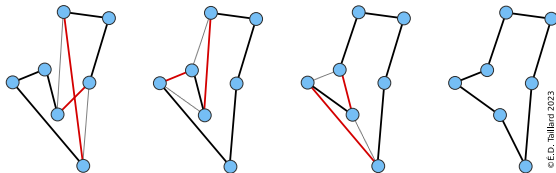


Successive improvements of a TSP solution with a 2-opt local search
Two edges are replaced by two others whose sum of lengths is smaller

Local Search

General Idea

- Start from a solution obtained with a constructive method
- Modify it locally to improve it



Successive improvements of a TSP solution with a 2-opt local search
Two edges are replaced by two others whose sum of lengths is smaller

5.1 Local Search Frame

Input: Solution s , method modifying a solution

Result: Improved solution s

```
1 repeat  
2   | if there is a modification of  $s$  into  $s'$  improving  $s$  then  
3   |   |  $s \leftarrow s'$   
4 until no improvement of  $s$  is found
```

Neighbourhood Definition

For each solution $s \in S$, define

- $V(s) \subset S$ a subset of *neighbour solutions* of s
- Often, $V(s)$ is defined by modifications brought to s
- When one moves from a solution s to a solution $s' \in V(s)$, one performs a *move* m

Example of Moves for a TSP

- m is specified by a pair $[i, j]$
- Consists of replacing the edges $[i, s_i]$ and $[j, s_j]$ by the edges $[i, j]$ and $[s_i, s_j]$
- Set of 2-opt moves: $M(s) = \{[i, j], i, j \in s, i \neq j, j \neq s_i, i \neq s_j\}$
- Definition of $V(s) = \{s' | s' = s \oplus m, m \in M(s), s \in S\}$

Neighbourhood Definition

For each solution $s \in S$, define

- $V(s) \subset S$ a subset of *neighbour solutions* of s
- Often, $V(s)$ is defined by modifications brought to s
- When one moves from a solution s to a solution $s' \in V(s)$, one performs a *move* m

Example of Moves for a TSP

- m is specified by a pair $[i, j]$
- Consists of replacing the edges $[i, s_i]$ and $[j, s_j]$ by the edges $[i, j]$ and $[s_i, s_j]$
- Set of 2-opt moves: $M(s) = \{[i, j], i, j \in s, i \neq j, j \neq s_i, i \neq s_j\}$
- Definition of $V(s) = \{s' | s' = s \oplus m, m \in M(s), s \in S\}$

Neighbourhood Definition

For each solution $s \in S$, define

- $V(s) \subset S$ a subset of *neighbour solutions* of s
- Often, $V(s)$ is defined by modifications brought to s
- When one moves from a solution s to a solution $s' \in V(s)$, one performs a *move* m

Example of Moves for a TSP

- m is specified by a pair $[i, j]$
- Consists of replacing the edges $[i, s_i]$ and $[j, s_j]$ by the edges $[i, j]$ and $[s_i, s_j]$
- Set of 2-opt moves: $M(s) = \{[i, j], i, j \in s, i \neq j, j \neq s_i, i \neq s_j\}$
- Definition of $V(s) = \{s' | s' = s \oplus m, m \in M(s), s \in S\}$

Neighbourhood Definition

For each solution $s \in S$, define

- $V(s) \subset S$ a subset of *neighbour solutions* of s
- Often, $V(s)$ is defined by modifications brought to s
- When one moves from a solution s to a solution $s' \in V(s)$, one performs a *move* m

Example of Moves for a TSP

- m is specified by a pair $[i, j]$
- Consists of replacing the edges $[i, s_i]$ and $[j, s_j]$ by the edges $[i, j]$ and $[s_i, s_j]$
- Set of 2-opt moves: $M(s) = \{[i, j], i, j \in s, i \neq j, j \neq s_i, i \neq s_j\}$
- Definition of $V(s) = \{s' | s' = s \oplus m, m \in M(s), s \in S\}$

Neighbourhood Definition

For each solution $s \in S$, define

- $V(s) \subset S$ a subset of *neighbour solutions* of s
- Often, $V(s)$ is defined by modifications brought to s
- When one moves from a solution s to a solution $s' \in V(s)$, one performs a *move* m

Example of Moves for a TSP

- m is specified by a pair $[i, j]$
- Consists of replacing the edges $[i, s_i]$ and $[j, s_j]$ by the edges $[i, j]$ and $[s_i, s_j]$
- Set of 2-opt moves: $M(s) = \{[i, j], i, j \in s, i \neq j, j \neq s_i, i \neq s_j\}$
- Definition of $V(s) = \{s' | s' = s \oplus m, m \in M(s), s \in S\}$

Neighbourhood Definition

For each solution $s \in S$, define

- $V(s) \subset S$ a subset of *neighbour solutions* of s
- Often, $V(s)$ is defined by modifications brought to s
- When one moves from a solution s to a solution $s' \in V(s)$, one performs a *move* m

Example of Moves for a TSP

- m is specified by a pair $[i, j]$
- Consists of replacing the edges $[i, s_i]$ and $[j, s_j]$ by the edges $[i, j]$ and $[s_i, s_j]$
- Set of 2-opt moves: $M(s) = \{[i, j], i, j \in s, i \neq j, j \neq s_i, i \neq s_j\}$
- Definition of $V(s) = \{s' | s' = s \oplus m, m \in M(s), s \in S\}$

Neighbourhood Definition

For each solution $s \in S$, define

- $V(s) \subset S$ a subset of *neighbour solutions* of s
- Often, $V(s)$ is defined by modifications brought to s
- When one moves from a solution s to a solution $s' \in V(s)$, one performs a *move* m

Example of Moves for a TSP

- m is specified by a pair $[i, j]$
- Consists of replacing the edges $[i, s_i]$ and $[j, s_j]$ by the edges $[i, j]$ and $[s_i, s_j]$
- Set of 2-opt moves: $M(s) = \{[i, j], i, j \in s, i \neq j, j \neq s_i, i \neq s_j\}$
- Definition of $V(s) = \{s' | s' = s \oplus m, m \in M(s), s \in S\}$

Policy of the first improving move

Input: Solution s , neighbourhood specification $N(\cdot)$, fitness function $f(\cdot)$ to minimize.

Result: Improved solution s

```
1 forall  $s' \in N(s)$  do  
2   if  $f(s') < f(s)$  then Move to  $s'$ , break the loop and initiate the next one  
3    $s \leftarrow s'$ 
```

Algorithm Complexity

- at most $O(|S|)$ moves
 - There is a finite number of solutions
 - One goes from a solution to a strictly better one
- The algorithm terminates after a time bounded by the number of solutions (in practice: greatly overestimated)

Policy of the first improving move

Input: Solution s , neighbourhood specification $N(\cdot)$, fitness function $f(\cdot)$ to minimize.

Result: Improved solution s

```
1 forall  $s' \in N(s)$  do  
2   if  $f(s') < f(s)$  then Move to  $s'$ , break the loop and initiate the next one  
3    $s \leftarrow s'$ 
```

Algorithm Complexity

- at most $O(|S|)$ moves
 - There is a finite number of solutions
 - One goes from a solution to a strictly better one
- The algorithm terminates after a time bounded by the number of solutions (in practice: greatly overestimated)

Policy of the first improving move

Input: Solution s , neighbourhood specification $N(\cdot)$, fitness function $f(\cdot)$ to minimize.

Result: Improved solution s

```
1 forall  $s' \in N(s)$  do  
2   if  $f(s') < f(s)$  then Move to  $s'$ , break the loop and initiate the next one  
3    $s \leftarrow s'$ 
```

Algorithm Complexity

- at most $O(|S|)$ moves
 - There is a finite number of solutions
 - One goes from a solution to a strictly better one
- The algorithm terminates after a time bounded by the number of solutions (in practice: greatly overestimated)

Policy of the first improving move

Input: Solution s , neighbourhood specification $N(\cdot)$, fitness function $f(\cdot)$ to minimize.

Result: Improved solution s

```
1 forall  $s' \in N(s)$  do  
2   if  $f(s') < f(s)$  then Move to  $s'$ , break the loop and initiate the next one  
3    $s \leftarrow s'$ 
```

Algorithm Complexity

- at most $O(|S|)$ moves
 - There is a finite number of solutions
 - One goes from a solution to a strictly better one
- The algorithm terminates after a time bounded by the number of solutions (in practice: greatly overestimated)

Best Move Policy

Input: Solution s , neighbourhood specification $N(\cdot)$, fitness function $f(\cdot)$ to minimize.

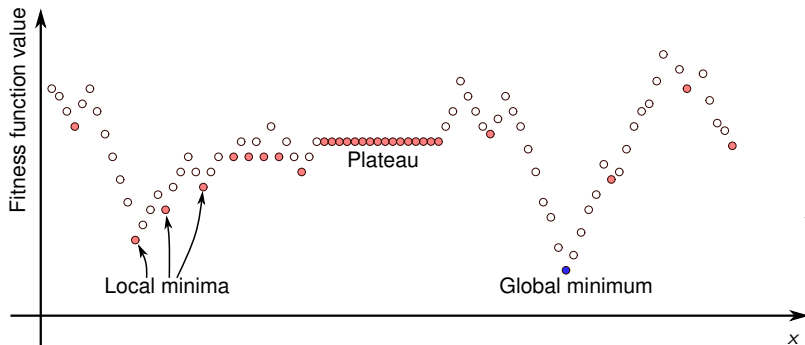
Result: Improved solution s

```
1 repeat
2    $end \leftarrow \text{true}$ 
3    $best\_neighbour\_value \leftarrow \infty$ 
4   forall  $s' \in N(s)$  do
5     if  $f(s') < best\_neighbour\_value$  then A better neighbour is found
6        $best\_neighbour\_value \leftarrow f(s')$ 
7        $best\_neighbour \leftarrow s'$ 
8   if  $best\_neighbour\_value < f(s)$  then Move to the improved solution
9      $s \leftarrow best\_neighbour$ 
10     $end \leftarrow \text{false}$ 
1 until end
```

Local, Global Minima, Plateau

Solution obtained by local improvement method

- *Local optimum*, not necessarily the best possible solution (*global optimum*)
- The quality of a local optimum depends on the starting solution and the definition of the neighbourhood



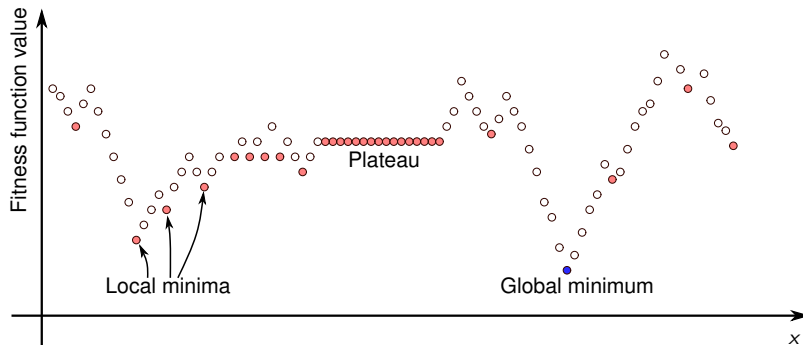
©É.D. Taillard 2023

Neighbourhood: modify x by one unit

Local, Global Minima, Plateau

Solution obtained by local improvement method

- *Local optimum*, not necessarily the best possible solution (*global optimum*)
- The quality of a local optimum depends on the starting solution and the definition of the neighbourhood

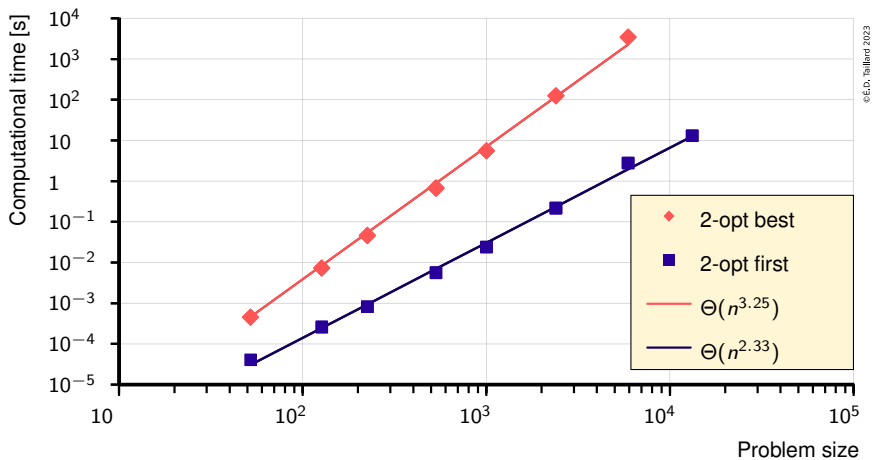


©É.D. Taillard 2023

Neighbourhood: modify x by one unit

Empirical Complexity of 2-opt Neighbourhood

Comparison between the best and first improvement

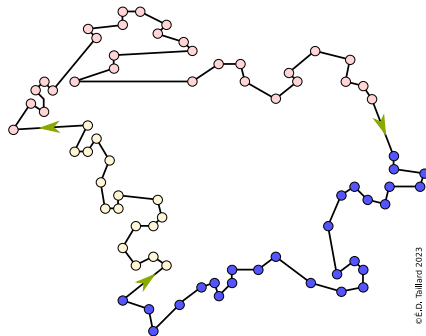
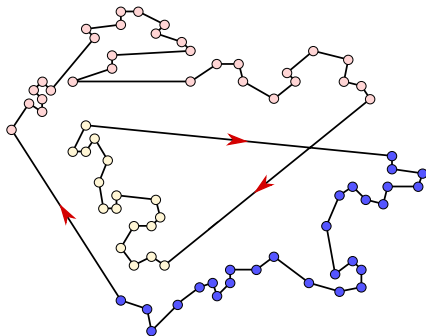


3-opt Neighbourhood

Remplace 3 edges by 3 others

$(i \rightarrow s_i), (j \rightarrow s_j), (k \rightarrow s_k)$ replaced by: $(i \rightarrow s_j), (j \rightarrow s_k), (k \rightarrow s_i)$

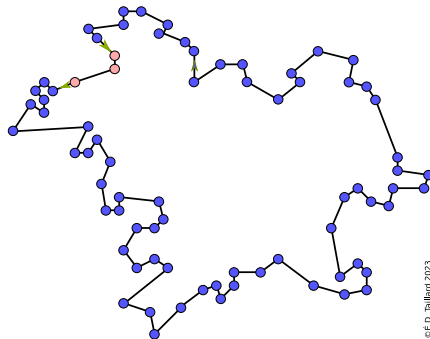
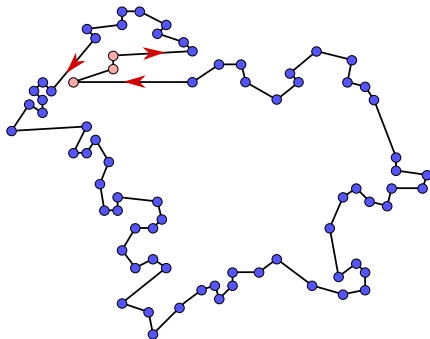
Respects the visiting order of the cities on the other arcs, unlike 2-opt



© E.D. Taillard 2023

Or-opt Neighbourhood

Move sub-sequences of 3, 2 or 1 cities



©É.D. Taillard 2023

Neighbourhood Properties

Connectivity Any feasible solution can reach at least one globally optimal solution

Low Diameter Not too many steps are needed to connect any 2 solutions

Low Ruggedness Fitness function value does not change too much from one solution to its neighbour, few local optima

Small Size Not too many neighbour solutions should be evaluated

Fast Evaluation Fitness function evaluation of a neighbour solution must be fast

These properties are often antagonistic!

Neighbourhood Properties

Connectivity Any feasible solution can reach at least one globally optimal solution

Low Diameter Not too many steps are needed to connect any 2 solutions

Low Ruggedness Fitness function value does not change too much from one solution to its neighbour, few local optima

Small Size Not too many neighbour solutions should be evaluated

Fast Evaluation Fitness function evaluation of a neighbour solution must be fast

These properties are often antagonistic!

Neighbourhood Properties

Connectivity Any feasible solution can reach at least one globally optimal solution

Low Diameter Not too many steps are needed to connect any 2 solutions

Low Ruggedness Fitness function value does not change too much from one solution to its neighbour, few local optima

Small Size Not too many neighbour solutions should be evaluated

Fast Evaluation Fitness function evaluation of a neighbour solution must be fast

These properties are often antagonistic!

Neighbourhood Properties

Connectivity Any feasible solution can reach at least one globally optimal solution

Low Diameter Not too many steps are needed to connect any 2 solutions

Low Ruggedness Fitness function value does not change too much from one solution to its neighbour, few local optima

Small Size Not too many neighbour solutions should be evaluated

Fast Evaluation Fitness function evaluation of a neighbour solution must be fast

These properties are often antagonistic!

Neighbourhood Properties

Connectivity Any feasible solution can reach at least one globally optimal solution

Low Diameter Not too many steps are needed to connect any 2 solutions

Low Ruggedness Fitness function value does not change too much from one solution to its neighbour, few local optima

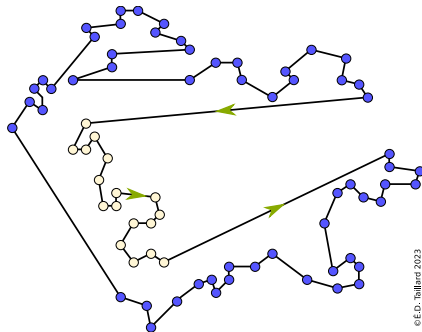
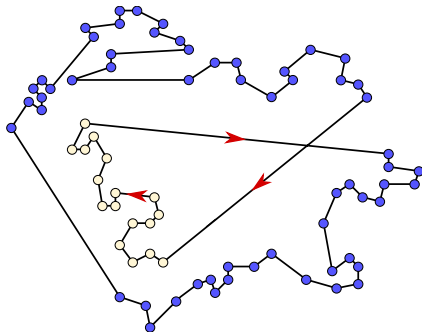
Small Size Not too many neighbour solutions should be evaluated

Fast Evaluation Fitness function evaluation of a neighbour solution must be fast

These properties are often antagonistic!

Properties of 2-opt Neighbourhood

- **Connectivity:** Easy to verify; in exercise
- **Graph Diameter:** $O(n)$
- **Low Ruggedness:** This criterion is not fulfilled for very asymmetric problems
A significant portion of the tour may be reversed by a 2-opt move

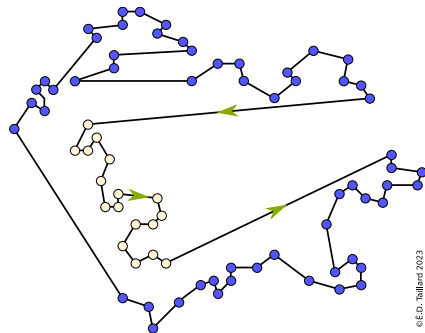
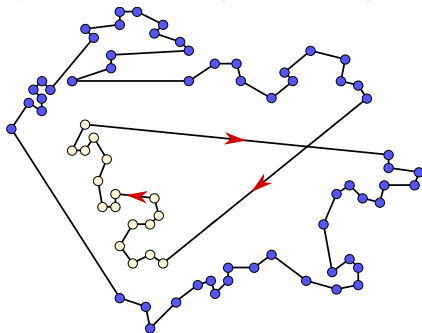


©É.D. Taillard 2023

- **Limited Size:** $O(n^2)$
- **Fast Evaluation:** For symmetrical instances: a neighbour solution can be evaluated in constant time
Modification of a solution in $O(n)$ (without a special data structure)

Properties of 2-opt Neighbourhood

- **Connectivity:** Easy to verify; in exercise
- **Graph Diameter:** $O(n)$
- **Low Ruggedness:** This criterion is not fulfilled for very asymmetric problems
A significant portion of the tour may be reversed by a 2-opt move

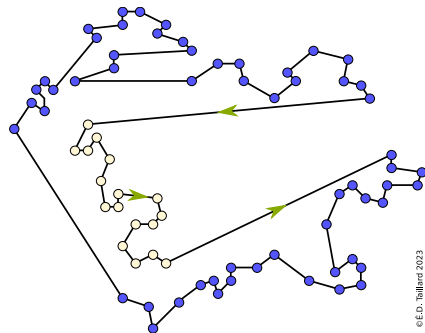
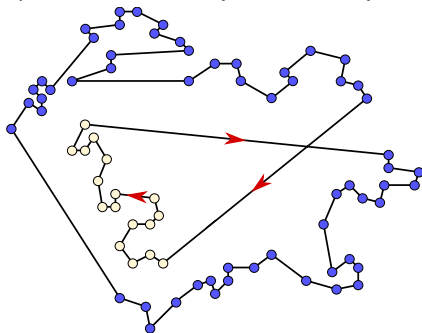


©É.D. Taillard 2023

- **Limited Size:** $O(n^2)$
- **Fast Evaluation:** For symmetrical instances: a neighbour solution can be evaluated in constant time
Modification of a solution in $O(n)$ (without a special data structure)

Properties of 2-opt Neighbourhood

- **Connectivity:** Easy to verify; in exercise
- **Graph Diameter:** $O(n)$
- **Low Ruggedness:** This criterion is not fulfilled for very asymmetric problems
A significant portion of the tour may be reversed by a 2-opt move

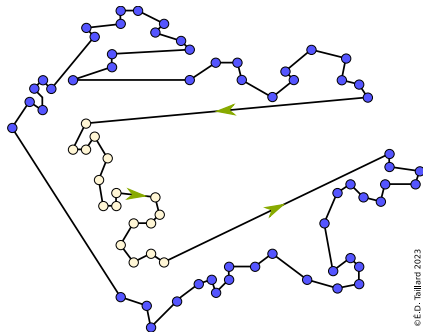
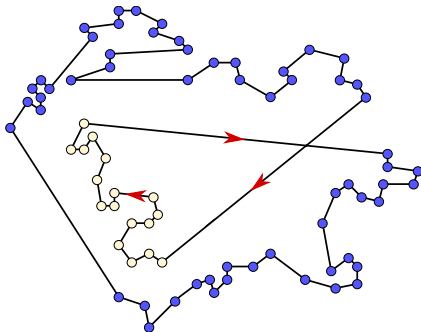


©É.D. Taillard 2023

- **Limited Size:** $O(n^2)$
- **Fast Evaluation:** For symmetrical instances: a neighbour solution can be evaluated in constant time
Modification of a solution in $O(n)$ (without a special data structure)

Properties of 2-opt Neighbourhood

- **Connectivity:** Easy to verify; in exercise
- **Graph Diameter:** $O(n)$
- **Low Ruggedness:** This criterion is not fulfilled for very asymmetric problems
A significant portion of the tour may be reversed by a 2-opt move

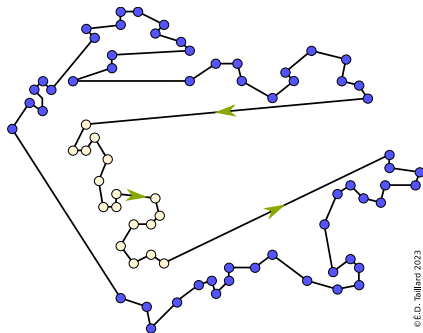
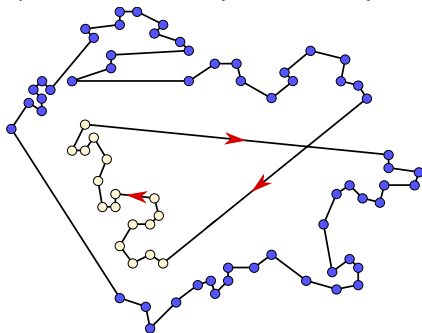


©É.D. Taillard 2023

- **Limited Size:** $O(n^2)$
- **Fast Evaluation:** For symmetrical instances: a neighbour solution can be evaluated in constant time
Modification of a solution in $O(n)$ (without a special data structure)

Properties of 2-opt Neighbourhood

- **Connectivity:** Easy to verify; in exercise
- **Graph Diameter:** $O(n)$
- **Low Ruggedness:** This criterion is not fulfilled for very asymmetric problems
A significant portion of the tour may be reversed by a 2-opt move

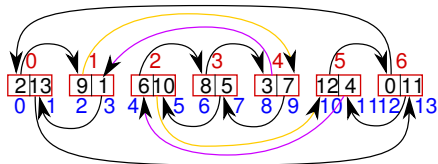
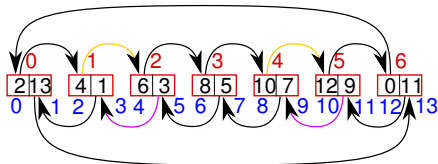
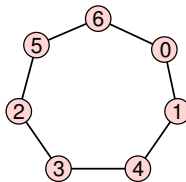
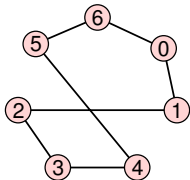


©É.D. Taillard 2023

- **Limited Size:** $O(n^2)$
- **Fast Evaluation:** For symmetrical instances: a neighbour solution can be evaluated in constant time
Modification of a solution in $O(n)$ (without a special data structure)

Data structure to perform a 2-opt move in $O(1)$

A portion of the tour must be reversed in constant time

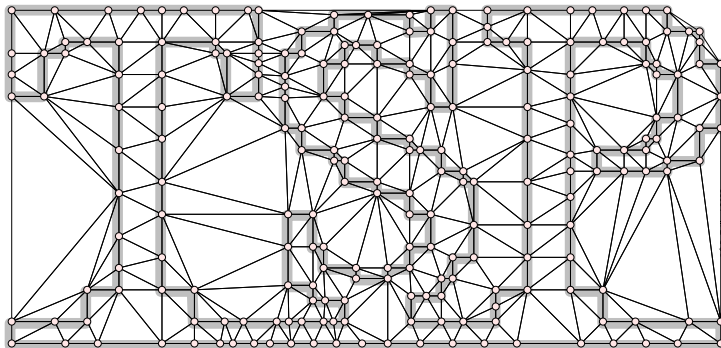


5.2 Limiting the Neighbourhood Size

Limiting the Neighbourhood Size: Candidate List

Idea: do not consider all neighbour solutions, but only a potentially interesting subset

Example for the Euclidean TSP: restrict to the edges contained in a Delaunay triangulation



5.3 Neighbourhood Extension

Neighbourhood Extension

- A simple neighbourhood, such as 2-opt for the TSP, has many local optima from which it is difficult to escape
- The changes made to the solution only marginally alter its structure
- Using a larger neighbourhood (4-opt, 5-opt) the computational effort becomes prohibitive
- It is not necessarily obvious how to define more complicated neighbourhoods
- How to extend a simple neighbourhood to make more significant changes to the solution
- Without increasing the complexity too much?

Neighbourhood Extension

- A simple neighbourhood, such as 2-opt for the TSP, has many local optima from which it is difficult to escape
- The changes made to the solution only marginally alter its structure
- Using a larger neighbourhood (4-opt, 5-opt) the computational effort becomes prohibitive
- It is not necessarily obvious how to define more complicated neighbourhoods
- How to extend a simple neighbourhood to make more significant changes to the solution
- Without increasing the complexity too much?

Neighbourhood Extension

- A simple neighbourhood, such as 2-opt for the TSP, has many local optima from which it is difficult to escape
- The changes made to the solution only marginally alter its structure
- Using a larger neighbourhood (4-opt, 5-opt) the computational effort becomes prohibitive
- It is not necessarily obvious how to define more complicated neighbourhoods
- How to extend a simple neighbourhood to make more significant changes to the solution
- Without increasing the complexity too much?

Neighbourhood Extension

- A simple neighbourhood, such as 2-opt for the TSP, has many local optima from which it is difficult to escape
- The changes made to the solution only marginally alter its structure
- Using a larger neighbourhood (4-opt, 5-opt) the computational effort becomes prohibitive
- It is not necessarily obvious how to define more complicated neighbourhoods
- How to extend a simple neighbourhood to make more significant changes to the solution
- Without increasing the complexity too much?

Neighbourhood Extension

- A simple neighbourhood, such as 2-opt for the TSP, has many local optima from which it is difficult to escape
- The changes made to the solution only marginally alter its structure
- Using a larger neighbourhood (4-opt, 5-opt) the computational effort becomes prohibitive
- It is not necessarily obvious how to define more complicated neighbourhoods
- How to extend a simple neighbourhood to make more significant changes to the solution
- Without increasing the complexity too much?

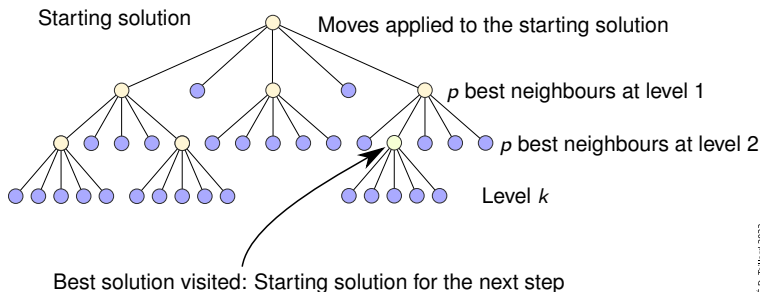
Neighbourhood Extension

- A simple neighbourhood, such as 2-opt for the TSP, has many local optima from which it is difficult to escape
- The changes made to the solution only marginally alter its structure
- Using a larger neighbourhood (4-opt, 5-opt) the computational effort becomes prohibitive
- It is not necessarily obvious how to define more complicated neighbourhoods
- How to extend a simple neighbourhood to make more significant changes to the solution
- Without increasing the complexity too much?

Neighbourhood Extension: Filter and Fan

Idea: Beam search adapted to the case of neighbourhood evaluation

- Evaluate the solution set of a simple neighbourhood
- Examine neighbours of neighbours, etc. up to a certain depth
- Avoid exponential growth in the number of solutions by filtering at each stage

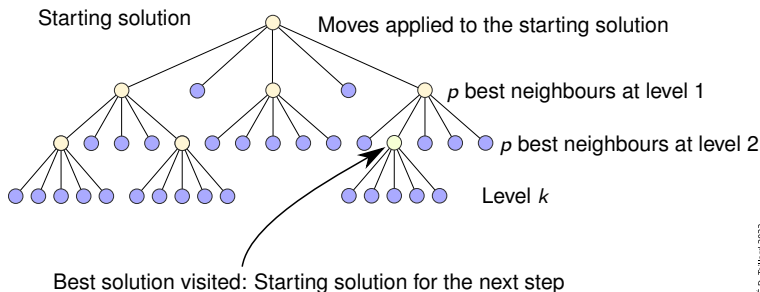


© E.D. Taillard 2023

Neighbourhood Extension: Filter and Fan

Idea: Beam search adapted to the case of neighbourhood evaluation

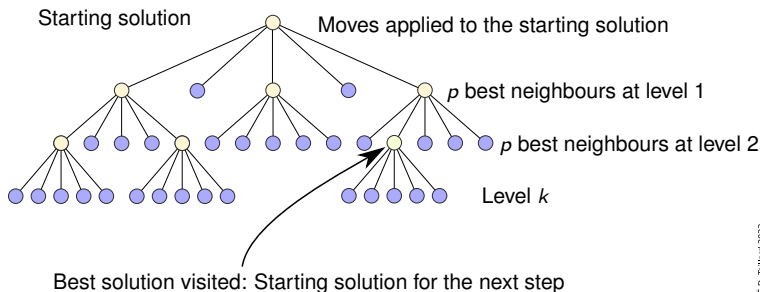
- Evaluate the solution set of a simple neighbourhood
- Examine neighbours of neighbours, etc. up to a certain depth
- Avoid exponential growth in the number of solutions by filtering at each stage



Neighbourhood Extension: Filter and Fan

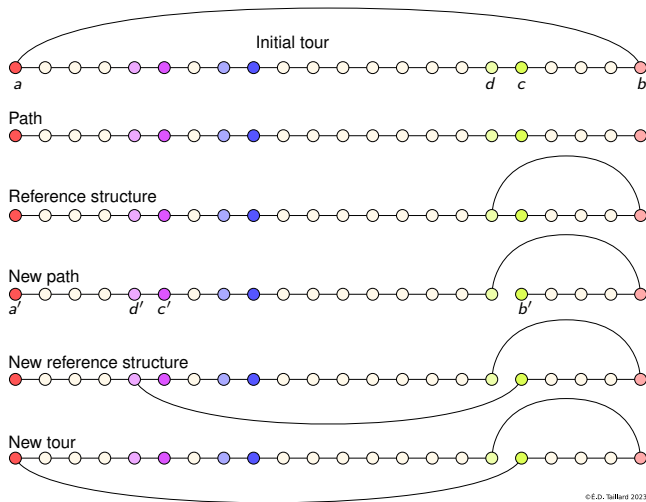
Idea: Beam search adapted to the case of neighbourhood evaluation

- Evaluate the solution set of a simple neighbourhood
- Examine neighbours of neighbours, etc. up to a certain depth
- Avoid exponential growth in the number of solutions by filtering at each stage



© E.D. Taillard 2023

Neighbourhood Extension: Ejection Chains



©É.D. Taillard 2023

Ejection Chains

- **Chain Initialization:** Eject an edge $[a, b]$ to get a path
- **Chain Propagation:** Insert an edge $[b, d]$ to obtain a *reference structure*
 - The edge $[b, d]$ must not have been previously deleted during the ejection chain
 - The weight of the reference structure must be smaller than that of the starting tour
 - Ejecting the edge $[c, d]$ (not previously added during the ejection chain) results in the same situation as before
- **Stop of the Propagation**
 - There is no longer a suitable $[b, d]$ or $[c, d]$ edges
 - The tour obtained by adding the edge $[a, c]$ is shorter than the starting tour

Ejection Chains

- **Chain Initialization:** Eject an edge $[a, b]$ to get a path
- **Chain Propagation:** Insert an edge $[b, d]$ to obtain a *reference structure*
 - The edge $[b, d]$ must not have been previously deleted during the ejection chain
 - The weight of the reference structure must be smaller than that of the starting tour
 - Ejecting the edge $[c, d]$ (not previously added during the ejection chain) results in the same situation as before
- **Stop of the Propagation**
 - There is no longer a suitable $[b, d]$ or $[c, d]$ edges
 - The tour obtained by adding the edge $[a, c]$ is shorter than the starting tour

Ejection Chains

- **Chain Initialization:** Eject an edge $[a, b]$ to get a path
- **Chain Propagation:** Insert an edge $[b, d]$ to obtain a *reference structure*
 - The edge $[b, d]$ must not have been previously deleted during the ejection chain
 - The weight of the reference structure must be smaller than that of the starting tour
 - Ejecting the edge $[c, d]$ (not previously added during the ejection chain) results in the same situation as before
- **Stop of the Propagation**
 - There is no longer a suitable $[b, d]$ or $[c, d]$ edges
 - The tour obtained by adding the edge $[a, c]$ is shorter than the starting tour

Ejection Chains

- **Chain Initialization:** Eject an edge $[a, b]$ to get a path
- **Chain Propagation:** Insert an edge $[b, d]$ to obtain a *reference structure*
 - The edge $[b, d]$ must not have been previously deleted during the ejection chain
 - The weight of the reference structure must be smaller than that of the starting tour
 - Ejecting the edge $[c, d]$ (not previously added during the ejection chain) results in the same situation as before
- **Stop of the Propagation**
 - There is no longer a suitable $[b, d]$ or $[c, d]$ edges
 - The tour obtained by adding the edge $[a, c]$ is shorter than the starting tour

Ejection Chains

- **Chain Initialization:** Eject an edge $[a, b]$ to get a path
- **Chain Propagation:** Insert an edge $[b, d]$ to obtain a *reference structure*
 - The edge $[b, d]$ must not have been previously deleted during the ejection chain
 - The weight of the reference structure must be smaller than that of the starting tour
 - Ejecting the edge $[c, d]$ (not previously added during the ejection chain) results in the same situation as before
- **Stop of the Propagation**
 - There is no longer a suitable $[b, d]$ or $[c, d]$ edges
 - The tour obtained by adding the edge $[a, c]$ is shorter than the starting tour

Ejection Chains

- **Chain Initialization:** Eject an edge $[a, b]$ to get a path
- **Chain Propagation:** Insert an edge $[b, d]$ to obtain a *reference structure*
 - The edge $[b, d]$ must not have been previously deleted during the ejection chain
 - The weight of the reference structure must be smaller than that of the starting tour
 - Ejecting the edge $[c, d]$ (not previously added during the ejection chain) results in the same situation as before
- **Stop of the Propagation**
 - There is no longer a suitable $[b, d]$ or $[c, d]$ edges
 - The tour obtained by adding the edge $[a, c]$ is shorter than the starting tour

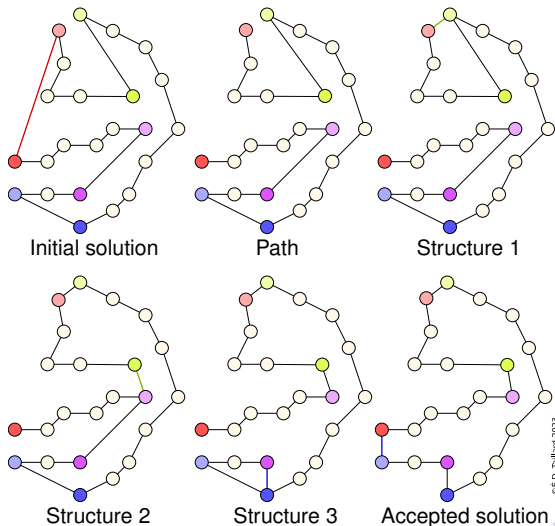
Ejection Chains

- **Chain Initialization:** Eject an edge $[a, b]$ to get a path
- **Chain Propagation:** Insert an edge $[b, d]$ to obtain a *reference structure*
 - The edge $[b, d]$ must not have been previously deleted during the ejection chain
 - The weight of the reference structure must be smaller than that of the starting tour
 - Ejecting the edge $[c, d]$ (not previously added during the ejection chain) results in the same situation as before
- **Stop of the Propagation**
 - There is no longer a suitable $[b, d]$ or $[c, d]$ edges
 - The tour obtained by adding the edge $[a, c]$ is shorter than the starting tour

Ejection Chains

- **Chain Initialization:** Eject an edge $[a, b]$ to get a path
- **Chain Propagation:** Insert an edge $[b, d]$ to obtain a *reference structure*
 - The edge $[b, d]$ must not have been previously deleted during the ejection chain
 - The weight of the reference structure must be smaller than that of the starting tour
 - Ejecting the edge $[c, d]$ (not previously added during the ejection chain) results in the same situation as before
- **Stop of the Propagation**
 - There is no longer a suitable $[b, d]$ or $[c, d]$ edges
 - The tour obtained by adding the edge $[a, c]$ is shorter than the starting tour

Lin-Kernighan Neighbourhood for the TSP



Ejection Chain for the TSP

Input: TSP Solution s

Result: Improved solution s

```

1  repeat
2      Eject edge  $[a, b]$  to initiate the chain
3      repeat
4          Find the edge  $[b, d]$  to add, minimizing the reference structure weight, with  $[c, d]$  not has been removed in the
            current ejection chain
5          if Edge  $[b, d]$  found and reference structure weight smaller than tour  $s$  then
6              if Adding  $[a, c]$  and removing  $[c, d]$  improve the tour then
7                  Success: Replace solution  $s$  by the new discovered tour
8              else
9                  Add edge  $[b, d]$ 
10                 Remove edge  $[c, d]$ 
11                  $b \leftarrow c$ 
12          else
13              Ejection failure: come back to  $s$  and try another ejection
14      until  $s$  improved or no edge  $[b, d]$  exists
15  until all edges of  $s$  have vainly initiated a chain
  
```


5.4 Multi-Objective Local Search

Multi-Objective Local Search: Iterative Scalarizations

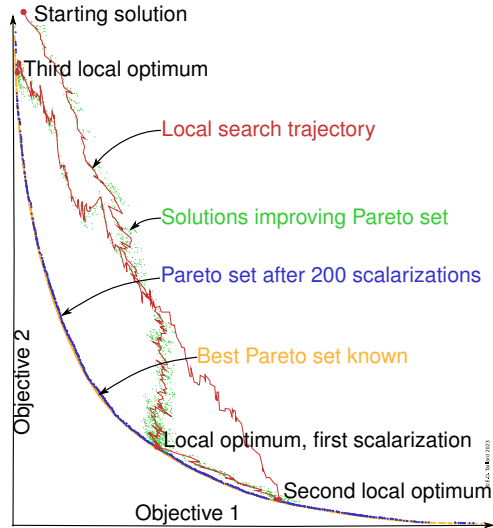
Input: Solution s , neighbourhood $N(\cdot)$, objective functions $\vec{f}(\cdot)$ to minimize; parameter I_{max}

Result: Approximation P of the Pareto set

```

1   $P = s$ 
2  for  $I_{max}$  iterations do
3      Randomly draw a weight vector  $\vec{w}$ 
4      repeat
5           $end \leftarrow \text{true}$ 
6           $best\_neighbour\_value \leftarrow \infty$ 
7          forall  $s' \in N(s)$  do
8              if  $s'$  is not dominated by solutions of  $P$  then
9                  Insert  $s'$  in  $P$  and remove the solutions of  $P$  dominated by  $s'$ 
10             if  $\vec{w} \cdot \vec{f}(s') < best\_neighbour\_value$  then
11                  $best\_neighbour\_value \leftarrow \vec{w} \cdot \vec{f}(s')$ 
12                  $best\_neighbour \leftarrow s'$ 
13             if  $best\_neighbour\_value < \vec{w} \cdot \vec{f}(s)$  then
14                  $s \leftarrow best\_neighbour$ 
15                  $end \leftarrow \text{false}$ 
16  until  $end$ 
  
```

Three steps of
iterative
scalarizations



Pareto Local Search

Neighbourhood_evaluation

Input: Solution s ; neighbourhood $N(\cdot)$ objective functions $\vec{f}(\cdot)$

Result: Approximation of Pareto set P completed with neighbours of s

1 **forall** $s' \in N(s)$ **do**

2 Update_Pareto(s' , $\vec{f}(s')$)

3 **Update_Pareto**

Input: Solution s , objective values \vec{v}

Result: Updated Pareto set P

4 **if** (s, \vec{v}) *either dominates a solution of P or $P = \emptyset$* **then**

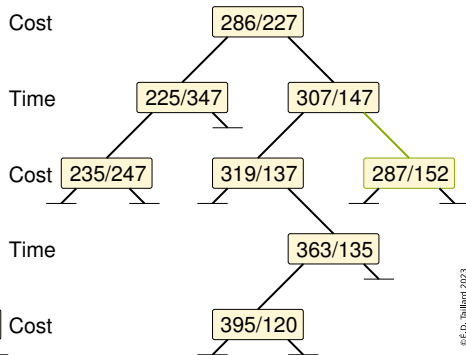
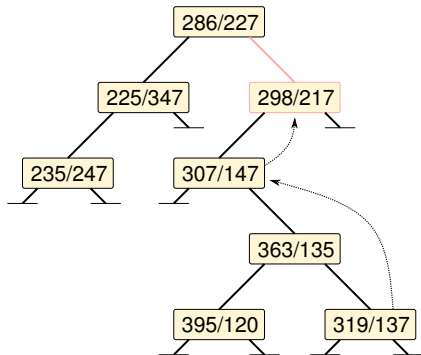
5 From P , remove all the solutions dominated by (s, \vec{v})

6 $P \leftarrow P \cup (s, \vec{v})$

7 Neighbourhood_evaluation(s)

Removal of a Dominated Element and Replacing it in a KD-Tree

New element of cost 287 and time 152, dominating element 298/217



©É.D. Taillard 2023

Chapter 6

Decomposition Methods

Chapter Content

| | | |
|----------|---------------------------------------|------------|
| 6 | Decomposition Methods | 142 |
| • | Consideration on the Problem Size | 144 |
| • | Recursive Algorithms | 145 |
| • | Master Theorem for Divide-and-Conquer | |
| • | Low Complexity Constructive Methods | 147 |
| • | Proximity Graph Construction | |
| • | Linearithmic Heuristics for the TSP | |
| • | Local Search for Large Instances | 152 |
| • | Large Neighbourhood Search | |
| • | POPMUSIC | |

6.1 Consideration on the Problem Size

| Class | Typical Approach | Size |
|----------|------------------------|---------------------------------|
| Toy | Exhaustive Enumeration | $n \approx 10$ |
| Small | Implicit Enumeration | $10 \lesssim n \lesssim 10^2$ |
| Standard | Metaheuristics | $10^2 \lesssim n \lesssim 10^4$ |
| Large | Decomposition Methods | $10^3 \lesssim n \lesssim 10^7$ |
| Huge | Big data | $n > 10^7$ elements |

6.2 Recursive Algorithms

- Solving the problem directly requires a more than linear computational effort, otherwise a decomposition only makes sense for a parallel calculation
- The pieces must be able to be processed independently of each other
- The assembly of the pieces must be of a lower complexity than solving the problem directly

Investigate the relevance of a recursive approach with the general recurrence theorem

6.2 Recursive Algorithms

- Solving the problem directly requires a more than linear computational effort, otherwise a decomposition only makes sense for a parallel calculation
- The pieces must be able to be processed independently of each other
- The assembly of the pieces must be of a lower complexity than solving the problem directly

Investigate the relevance of a recursive approach with the general recurrence theorem

6.2 Recursive Algorithms

- Solving the problem directly requires a more than linear computational effort, otherwise a decomposition only makes sense for a parallel calculation
- The pieces must be able to be processed independently of each other
- The assembly of the pieces must be of a lower complexity than solving the problem directly

Investigate the relevance of a recursive approach with the general recurrence theorem

Master Theorem for Divide-and-Conquer

Problem of size n split into b parts; among them, a are recursively solved
Find the solution of the functional equation $T(n) = a \cdot T(n/b) + f(n)$

- If $f(n) = O(n^{\log_b(a)-\epsilon})$, then $T(n) = \Theta(n^{\log_b(a)})$
- If $f(n) = \Theta(n^{\log_b(a)})$, then $T(n) = \Theta(n^{\log_b(a)} \cdot \log n)$
- If $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ and if $a \cdot f(n/b) < c \cdot f(n)$ with $c < 1$, constant, then $T(n) = \Theta(f(n))$

Examples

Dichotomous search $a = 1, b = 2, f(n) = \Theta(n^{\log_2(1)}) = \Theta(1) \implies T(n) = \Theta(\log n)$

Merge Sort $a = 2, b = 2, f(n) = \Theta(n) \implies T(n) = \Theta(n \log n)$

Query in a 2D-Tree

$$a = 2, b = 4, f(n) = O(1) = O(n^{\log_b(a)-\epsilon}) = O(n^{0.5-0.5}) \implies T(n) = \Theta(\sqrt{n})$$

Master Theorem for Divide-and-Conquer

Problem of size n split into b parts; among them, a are recursively solved
Find the solution of the functional equation $T(n) = a \cdot T(n/b) + f(n)$

- If $f(n) = O(n^{\log_b(a)-\epsilon})$, then $T(n) = \Theta(n^{\log_b(a)})$
- If $f(n) = \Theta(n^{\log_b(a)})$, then $T(n) = \Theta(n^{\log_b(a)} \cdot \log n)$
- If $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ and if $a \cdot f(n/b) < c \cdot f(n)$ with $c < 1$, constant, then $T(n) = \Theta(f(n))$

Examples

Dichotomous search $a = 1, b = 2, f(n) = \Theta(n^{\log_2(1)}) = \Theta(1) \implies T(n) = \Theta(\log n)$

Merge Sort $a = 2, b = 2, f(n) = \Theta(n) \implies T(n) = \Theta(n \log n)$

Query in a 2D-Tree

$$a = 2, b = 4, f(n) = O(1) = O(n^{\log_b(a)-\epsilon}) = O(n^{0.5-0.5}) \implies T(n) = \Theta(\sqrt{n})$$

Master Theorem for Divide-and-Conquer

Problem of size n split into b parts; among them, a are recursively solved
Find the solution of the functional equation $T(n) = a \cdot T(n/b) + f(n)$

- If $f(n) = O(n^{\log_b(a)-\epsilon})$, then $T(n) = \Theta(n^{\log_b(a)})$
- If $f(n) = \Theta(n^{\log_b(a)})$, then $T(n) = \Theta(n^{\log_b(a)} \cdot \log n)$
- If $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ and if $a \cdot f(n/b) < c \cdot f(n)$ with $c < 1$, constant, then $T(n) = \Theta(f(n))$

Examples

Dichotomous search $a = 1, b = 2, f(n) = \Theta(n^{\log_2(1)}) = \Theta(1) \implies T(n) = \Theta(\log n)$

Merge Sort $a = 2, b = 2, f(n) = \Theta(n) \implies T(n) = \Theta(n \log n)$

Query in a 2D-Tree

$$a = 2, b = 4, f(n) = O(1) = O(n^{\log_b(a)-\epsilon}) = O(n^{0.5-0.5}) \implies T(n) = \Theta(\sqrt{n})$$

6.3 Low Complexity Constructive Methods

First Decomposition Level

Assumption: not all elements of a problem are related to all others

- Idea: create a subset of the possible relationships between elements
- Limit the number of relationships for each element



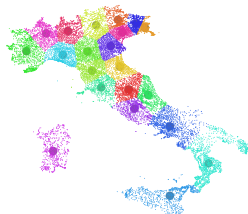
(a) Complete set



(b) Sample



(c) Centers



(d) Clusters

First Decomposition Level

Assumption: not all elements of a problem are related to all others

- Idea: create a subset of the possible relationships between elements
- Limit the number of relationships for each element



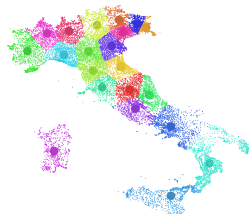
(a) Complete set



(b) Sample

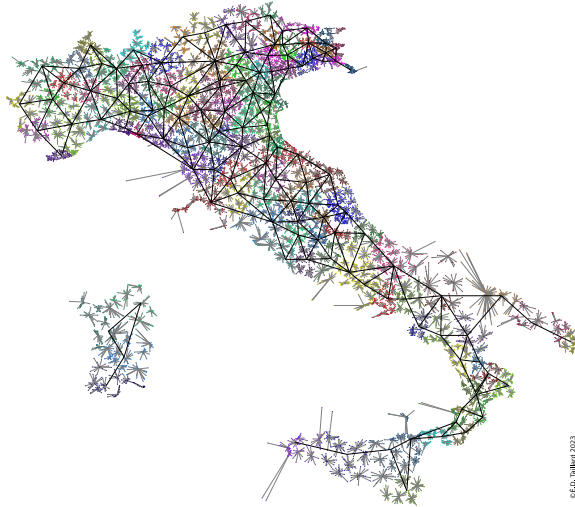


(c) Centers



(d) Clusters

Second Decomposition Level



©É.D. Taillard 2023

Clustering for Creating Relationships

- Create $\Theta(\sqrt{n})$ groups of approximately the same size (e.g. with a modified and accelerated PAM algorithm; this can be done with empirical complexity $\hat{O}(n^{3/2})$)
- Two centres c_i and c_j are related if they are the nearest and second nearest centres of an element
- Assuming that a centre is only related to $O(1)$ other centres and that the groups have only $O(n)$ elements, we can construct in $O(n^{3/2})$ proximity relations between the n entities
- We can exploit these relations in a greedy constructive method to obtain a solution in $O(n^{3/2})$

Clustering for Creating Relationships

- Create $\Theta(\sqrt{n})$ groups of approximately the same size (e.g. with a modified and accelerated PAM algorithm; this can be done with empirical complexity $\hat{O}(n^{3/2})$)
- Two centres c_i and c_j are related if they are the nearest and second nearest centres of an element
- Assuming that a centre is only related to $O(1)$ other centres and that the groups have only $O(n)$ elements, we can construct in $O(n^{3/2})$ proximity relations between the n entities
- We can exploit these relations in a greedy constructive method to obtain a solution in $O(n^{3/2})$

Clustering for Creating Relationships

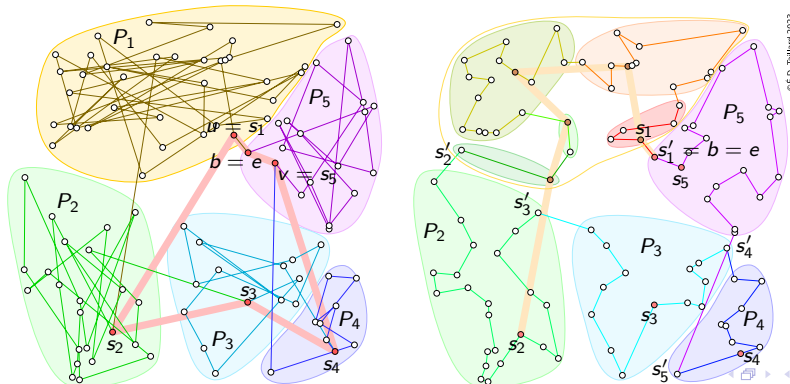
- Create $\Theta(\sqrt{n})$ groups of approximately the same size (e.g. with a modified and accelerated PAM algorithm; this can be done with empirical complexity $\hat{O}(n^{3/2})$)
- Two centres c_i and c_j are related if they are the nearest and second nearest centres of an element
- Assuming that a centre is only related to $O(1)$ other centres and that the groups have only $O(n)$ elements, we can construct in $O(n^{3/2})$ proximity relations between the n entities
- We can exploit these relations in a greedy constructive method to obtain a solution in $O(n^{3/2})$

Clustering for Creating Relationships

- Create $\Theta(\sqrt{n})$ groups of approximately the same size (e.g. with a modified and accelerated PAM algorithm; this can be done with empirical complexity $\hat{O}(n^{3/2})$)
- Two centres c_i and c_j are related if they are the nearest and second nearest centres of an element
- Assuming that a centre is only related to $O(1)$ other centres and that the groups have only $O(n)$ elements, we can construct in $O(n^{3/2})$ proximity relations between the n entities
- We can exploit these relations in a greedy constructive method to obtain a solution in $O(n^{3/2})$

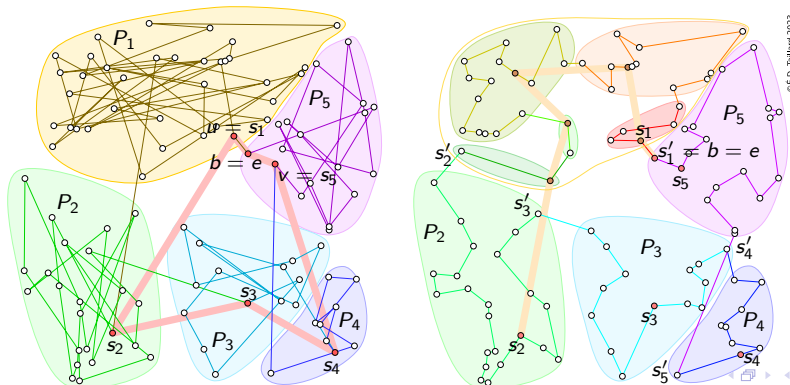
Recursive Construction for the TSP

- Create a tour on a sample of r cities
- Insert the remaining cities in any order after the nearest sample city
- If the path between 2 cities in the sample contains too many cities, decompose it recursively
- Otherwise, optimize it with a good quality method



Recursive Construction for the TSP

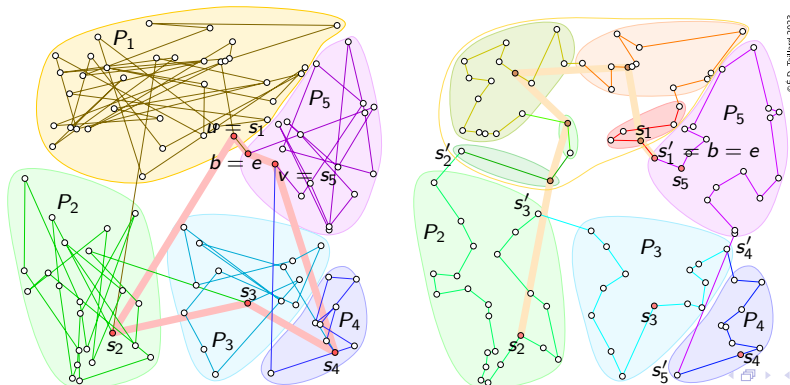
- Create a tour on a sample of r cities
- Insert the remaining cities in any order after the nearest sample city
- If the path between 2 cities in the sample contains too many cities, decompose it recursively
- Otherwise, optimize it with a good quality method



©É.D. Taillard 2023

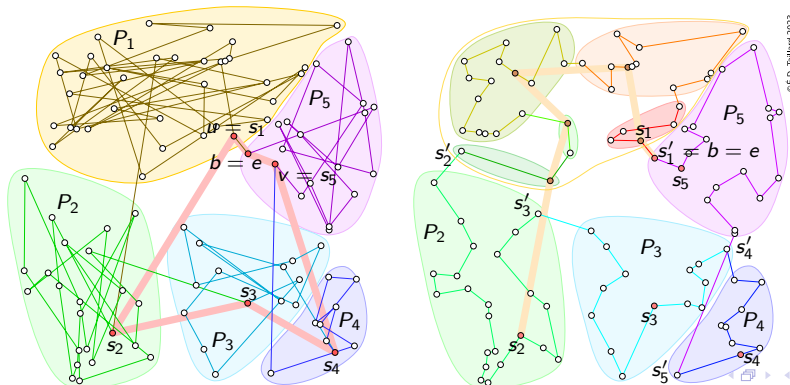
Recursive Construction for the TSP

- Create a tour on a sample of r cities
- Insert the remaining cities in any order after the nearest sample city
- If the path between 2 cities in the sample contains too many cities, decompose it recursively
- Otherwise, optimize it with a good quality method



Recursive Construction for the TSP

- Create a tour on a sample of r cities
- Insert the remaining cities in any order after the nearest sample city
- If the path between 2 cities in the sample contains too many cities, decompose it recursively
- Otherwise, optimize it with a good quality method



©É.D. Taillard 2023

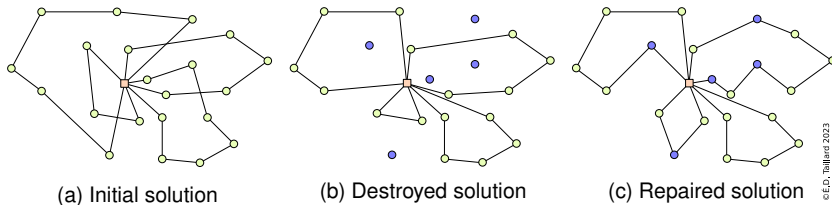
6.4 Local Search for Large Instances

Large Neighbourhood Search(LNS)

General Idea

- Destroy part of the solution
- Rebuild it by trying to do better

Illustration for a vehicle routing problem

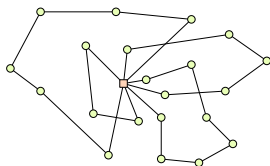


Large Neighbourhood Search(LNS)

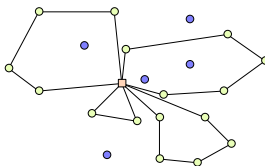
General Idea

- Destroy part of the solution
- Rebuild it by trying to do better

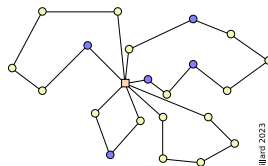
Illustration for a vehicle routing problem



(a) Initial solution



(b) Destroyed solution



(c) Repaired solution

© É.D. Taillard 2023

Large Neighbourhood SearchFrame

Input: Solution s , destroy method $d(\cdot)$, repair method $r(\cdot)$, acceptance criterion $a(\cdot, \cdot)$

Result: Improved solution s^*

```
1  $s^* \leftarrow s$ 
2 repeat
3    $s' \leftarrow r(d(s))$ 
4   if  $a(s, s')$  then
5      $s \leftarrow s'$ 
6   if  $s'$  better than  $s^*$  then
7      $s^* \leftarrow s'$ 
8 until a stopping criterion is satisfied
```

- Example of adaptation: $d(\cdot)$ consists in choosing a small subset of variables
- $r(\cdot)$ finds a local (or global) optimum by changing only the variables chosen by $d(\cdot)$
- $a(s, s') : f(s') < f(s)$

Large Neighbourhood SearchFrame

Input: Solution s , destroy method $d(\cdot)$, repair method $r(\cdot)$, acceptance criterion $a(\cdot, \cdot)$

Result: Improved solution s^*

```

1  $s^* \leftarrow s$ 
2 repeat
3    $s' \leftarrow r(d(s))$ 
4   if  $a(s, s')$  then
5      $s \leftarrow s'$ 
6   if  $s'$  better than  $s^*$  then
7      $s^* \leftarrow s'$ 
8 until a stopping criterion is satisfied

```

- Example of adaptation: $d(\cdot)$ consists in choosing a small subset of variables
- $r(\cdot)$ finds a local (or global) optimum by changing only the variables chosen by $d(\cdot)$
- $a(s, s') : f(s') < f(s)$

Large Neighbourhood SearchFrame

Input: Solution s , destroy method $d(\cdot)$, repair method $r(\cdot)$, acceptance criterion $a(\cdot, \cdot)$

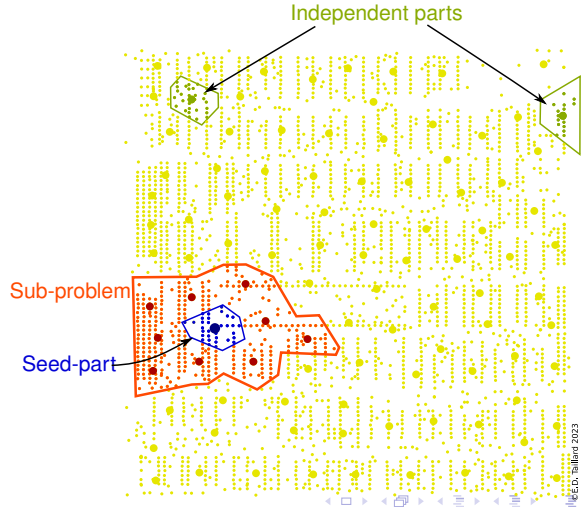
Result: Improved solution s^*

```
1  $s^* \leftarrow s$ 
2 repeat
3    $s' \leftarrow r(d(s))$ 
4   if  $a(s, s')$  then
5      $s \leftarrow s'$ 
6   if  $s'$  better than  $s^*$  then
7      $s^* \leftarrow s'$ 
8 until a stopping criterion is satisfied
```

- Example of adaptation: $d(\cdot)$ consists in choosing a small subset of variables
- $r(\cdot)$ finds a local (or global) optimum by changing only the variables chosen by $d(\cdot)$
- $a(s, s') : f(s') < f(s)$

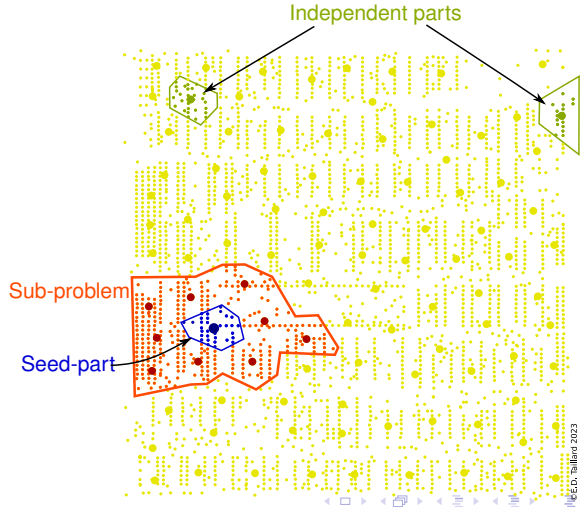
Local Improvements for a Centroid Clustering Problem

- Optimizing 2 very distant groups independently cannot bring benefits
- Choose a seed centre and a number of centres that are closest to it
- Optimize the position of these centres by considering the entities attached to them as an independent sub-problem
- Repeat with other seed-centres



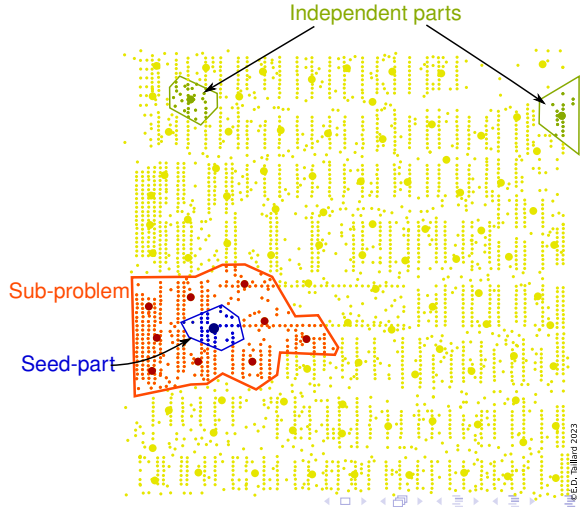
Local Improvements for a Centroid Clustering Problem

- Optimizing 2 very distant groups independently cannot bring benefits
- Choose a seed centre and a number of centres that are closest to it
- Optimize the position of these centres by considering the entities attached to them as an independent sub-problem
- Repeat with other seed-centres



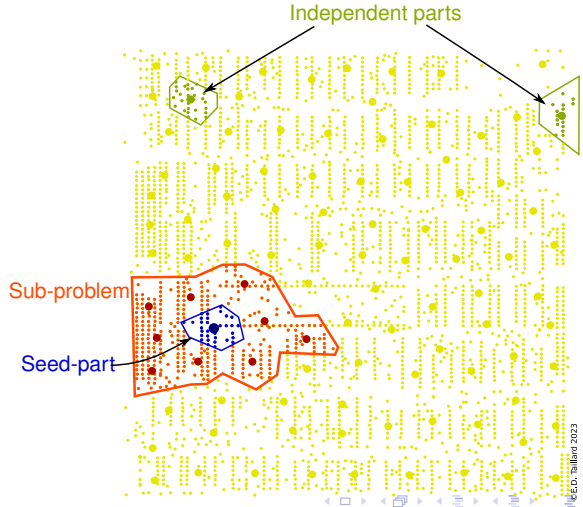
Local Improvements for a Centroid Clustering Problem

- Optimizing 2 very distant groups independently cannot bring benefits
- Choose a seed centre and a number of centres that are closest to it
- Optimize the position of these centres by considering the entities attached to them as an independent sub-problem
- Repeat with other seed-centres



Local Improvements for a Centroid Clustering Problem

- Optimizing 2 very distant groups independently cannot bring benefits
- Choose a seed centre and a number of centres that are closest to it
- Optimize the position of these centres by considering the entities attached to them as an independent sub-problem
- Repeat with other seed-centres



POPMUSIC Frame

Input: Initial solution s composed of q disjoint parts s_1, \dots, s_q ; sub-problem improvement method

Result: Improved solution s

```
1  $U = \{s_1, \dots, s_q\}$ 
2 while  $U \neq \emptyset$  do
3   Select  $s_g \in U$  //  $s_g$ : Seed part
4   Build a sub-problem  $R$  composed of the  $r$  parts of  $s$  the closest to  $s_g$ 
5   Tentatively optimize  $R$ 
6   if  $R$  is improved then
7     Update  $s$ 
8     From  $U$ , remove the part no longer belonging to  $s$ 
9     In  $U$ , insert the parts composing  $R$ 
10  else  $R$  not improved
11    Remove  $s_g$  from  $U$ 
```

Illustration of POPMUSIC for the VRP

- The clients of a tour define a part
- The proximity of the parts is measured by the distance of their centre of gravity

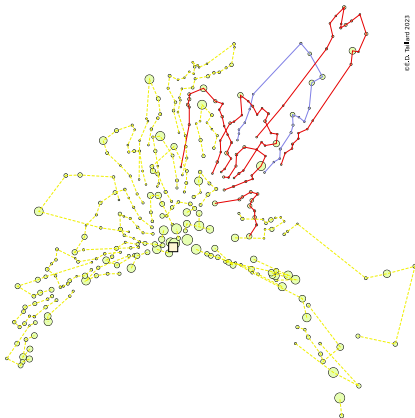
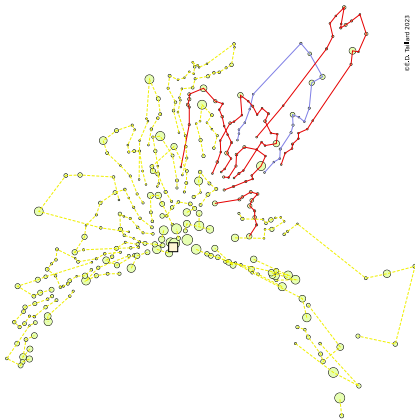


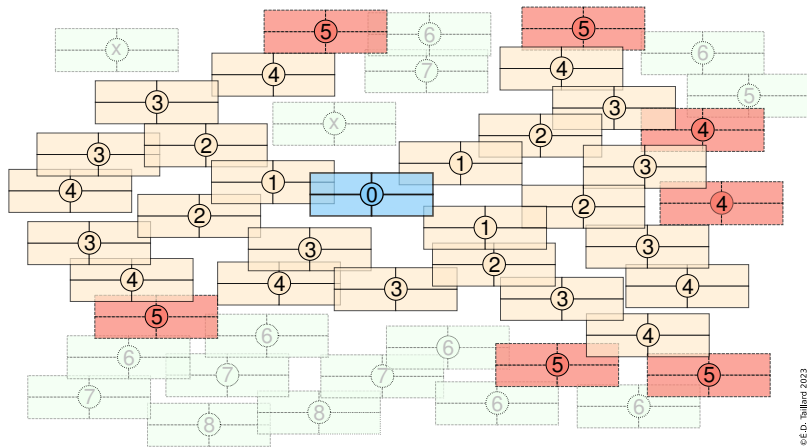
Illustration of POPMUSIC for the VRP

- The clients of a tour define a part
- The proximity of the parts is measured by the distance of their centre of gravity



POPMUSIC for Map Labelling

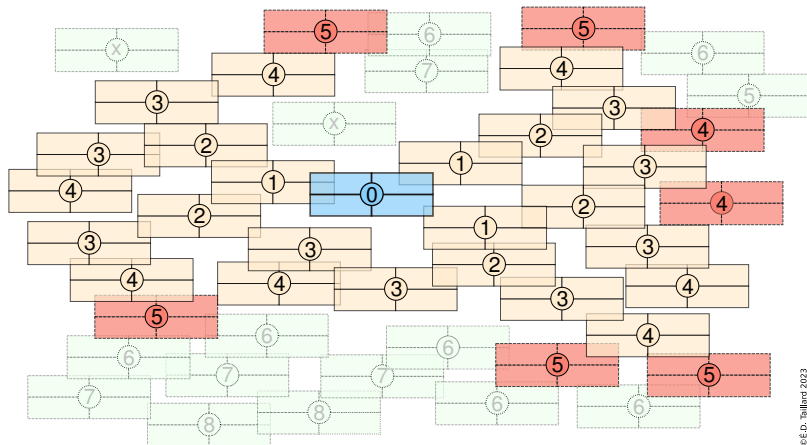
- A part is an object to be labelled (here: 4 possible positions for the label of each object)
- Two objects are at a distance of 1 if their labels can overlap



©É.D. Taillard 2023

POPMUSIC for Map Labelling

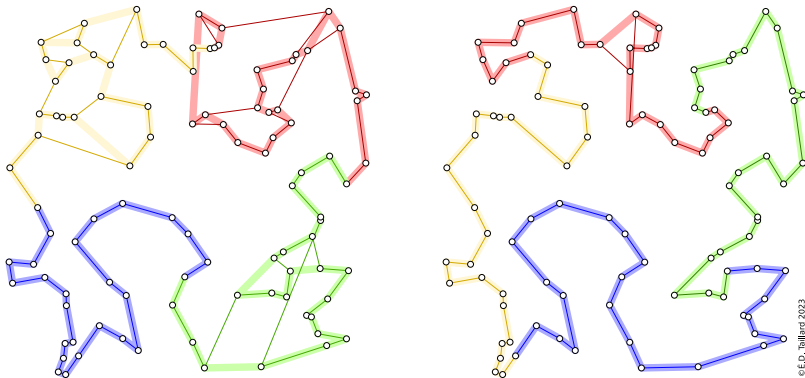
- A part is an object to be labelled (here: 4 possible positions for the label of each object)
- Two objects are at a distance of 1 if their labels can overlap



©É.D. Taillard 2023

POPMUSIC for the TSP

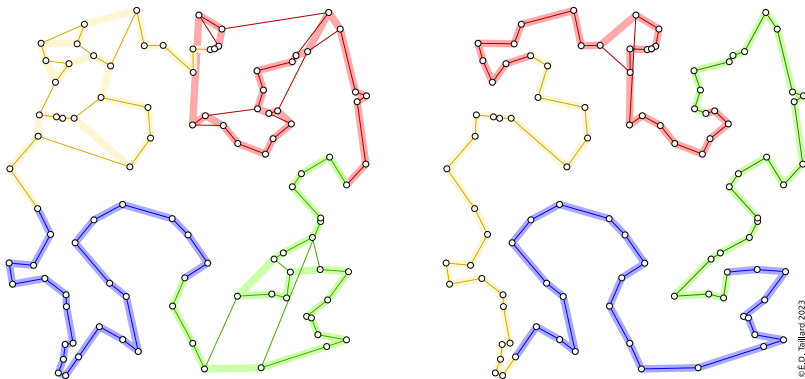
- Decompose the tour into sub-paths with approximately r cities
- Optimize each sub-path with a good method
- Repeat, shifting the sub-paths by $r/2$ cities



©É.D. Taillard 2023

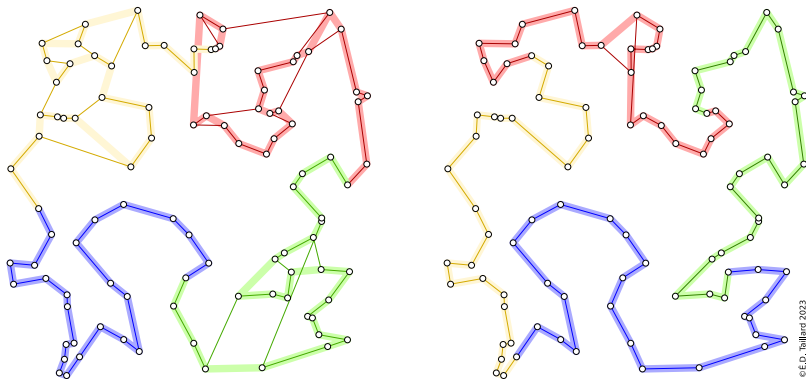
POPMUSIC for the TSP

- Decompose the tour into sub-paths with approximately r cities
- Optimize each sub-path with a good method
- Repeat, shifting the sub-paths by $r/2$ cities

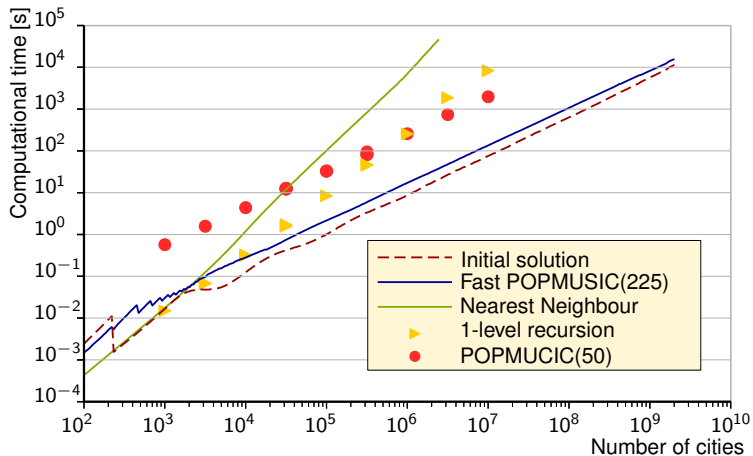


POPMUSIC for the TSP

- Decompose the tour into sub-paths with approximately r cities
- Optimize each sub-path with a good method
- Repeat, shifting the sub-paths by $r/2$ cities



Empirical Complexity of POPMUSIC

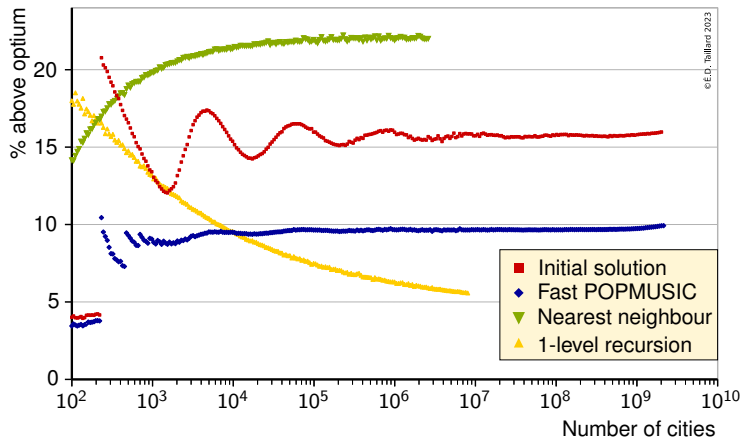


©É.D. Taillard 2023

Solution Quality for the TSP with Toroidal Distances

Fluctuations are due to the number of recursion levels

The sub-path optimization method provides solutions about 4% above the optimum



Chapter 7

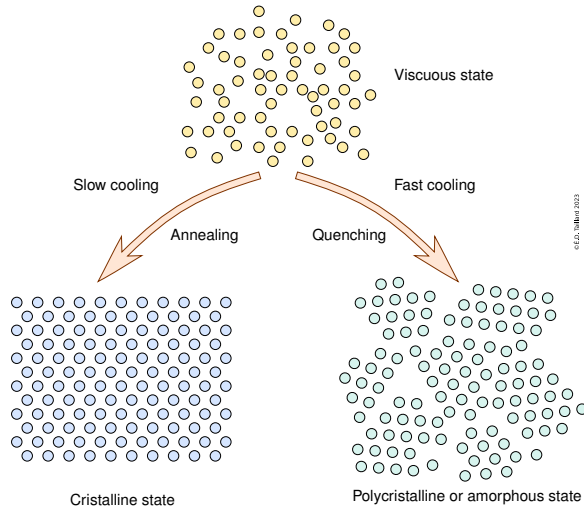
Randomized Methods

Chapter Content

| | | |
|---|-------------------------------------|-----|
| 7 | Randomized Methods | 162 |
| • | Simulated Annealing | 164 |
| • | Threshold Accepting | 169 |
| • | Great Deluge | 172 |
| • | Demon Algorithm | 174 |
| • | Noising Methods | 176 |
| • | Late Acceptance Hill Climbing | 179 |
| • | Variable Neighbourhood Search | 180 |
| • | GRASP | 182 |

7.1 Simulated Annealing

Annealing and Quenching Physical Processes



Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

Analogy with the metallurgical annealing process

- By rapidly cooling a molten metal, an amorphous structure is obtained
- By slowly cooling the metal, a crystalline structure is obtained
- The crystal structure is that which minimizes the internal energy of the of the metal
- The amorphous structure corresponds to a local minimum of energy
- The particles are in a stable state
- They do not have enough energy to change to another stable state with less energy

Idea of Simulated Annealing

- Make the analogy between the internal energy of a metal and the fitness function of a problem to be minimized
- Particles move randomly
- Allow degrading moves with a certain probability

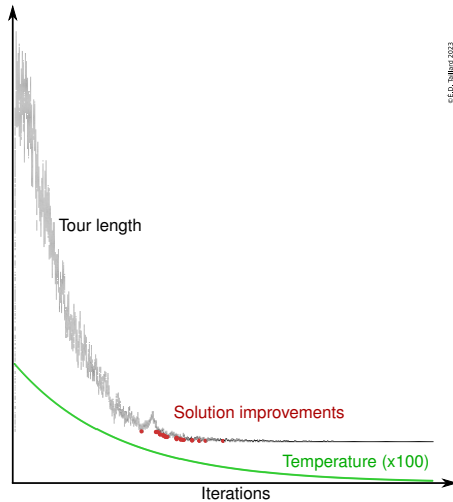
Simulated Annealing Frame

Input: Initial solution s ; fitness function f to minimize; neighbourhood structure M , parameters T_{init} , $T_{end} < T_{init}$ and $0 < \alpha < 1$

Result: Modified solution s

```
1  $T \leftarrow T_{init}$ 
2 while  $T > T_{end}$  do
3   Randomly generate  $m \in M$ 
4    $\Delta = f(s \oplus m) - f(s)$ 
5   Randomly generate  $0 < u < 1$ 
6   if  $\Delta < 0$  or  $e^{-\Delta/T} > u$  then  $m$  is accepted
7      $s \leftarrow s \oplus m$ 
8    $T \leftarrow \alpha \cdot T$ 
```

Evolution of a TSP Tour Length and Temperature



7.2 Threshold Accepting

Idea

- Observation: the computation of exponentials can take most of the computational effort of simulated annealing
- Draw a random move
- Accept this move if it degrades the solution by less than a certain threshold
- Start again, gradually lowering the threshold
- Threshold values:
 - Start by making random moves
 - Estimate the distribution function of the differences in values Δ between neighbour solutions

7.2 Threshold Accepting

Idea

- Observation: the computation of exponentials can take most of the computational effort of simulated annealing
- Draw a random move
- Accept this move if it degrades the solution by less than a certain threshold
- Start again, gradually lowering the threshold
- Threshold values:
 - Start by making random moves
 - Estimate the distribution function of the differences in values Δ between neighbour solutions

7.2 Threshold Accepting

Idea

- Observation: the computation of exponentials can take most of the computational effort of simulated annealing
- Draw a random move
- Accept this move if it degrades the solution by less than a certain threshold
- Start again, gradually lowering the threshold
- Threshold values:
 - Start by making random moves
 - Estimate the distribution function of the differences in values Δ between neighbour solutions

7.2 Threshold Accepting

Idea

- Observation: the computation of exponentials can take most of the computational effort of simulated annealing
- Draw a random move
- Accept this move if it degrades the solution by less than a certain threshold
- Start again, gradually lowering the threshold
- Threshold values:
 - Start by making random moves
 - Estimate the distribution function of the differences in values Δ between neighbour solutions

7.2 Threshold Accepting

Idea

- Observation: the computation of exponentials can take most of the computational effort of simulated annealing
- Draw a random move
- Accept this move if it degrades the solution by less than a certain threshold
- Start again, gradually lowering the threshold
- Threshold values:
 - Start by making random moves
 - Estimate the distribution function of the differences in values Δ between neighbour solutions

7.2 Threshold Accepting

Idea

- Observation: the computation of exponentials can take most of the computational effort of simulated annealing
- Draw a random move
- Accept this move if it degrades the solution by less than a certain threshold
- Start again, gradually lowering the threshold
- Threshold values:
 - Start by making random moves
 - Estimate the distribution function of the differences in values Δ between neighbour solutions

7.2 Threshold Accepting

Idea

- Observation: the computation of exponentials can take most of the computational effort of simulated annealing
- Draw a random move
- Accept this move if it degrades the solution by less than a certain threshold
- Start again, gradually lowering the threshold
- Threshold values:
 - Start by making random moves
 - Estimate the distribution function of the differences in values Δ between neighbour solutions

Threshold Accepting Frame

Input: Initial solution s ; fitness function f to minimize; neighbourhood structure M , parameters $T, R, \tau_1, \dots, \tau_T$

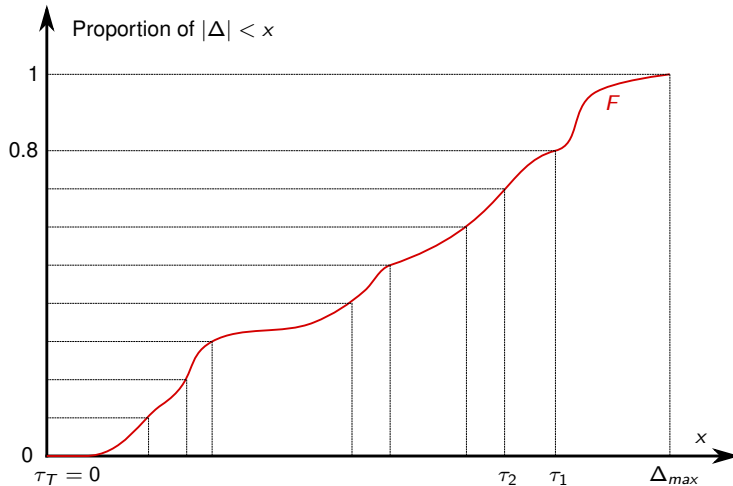
Result: Solution s^*

```

1  $s^* \leftarrow s$ 
2 for  $t$  from 1 to  $T$  do
3   for  $R$  iterations do
4     Randomly generate  $m \in M$ 
5     if  $f(s \oplus m) - f(s) < \tau_t$  then the move  $m$  is accepted
6        $s \leftarrow s \oplus m$ 
7       if  $f(s) < f(s^*)$  then
8          $s^* \leftarrow s$ 

```

Technique for determining thresholds τ_1, \dots, τ_T

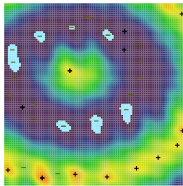


©É.D. Taillard 2023

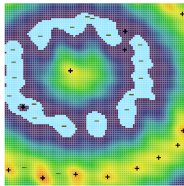
7.3 Great Deluge

Idea

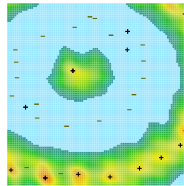
- Draw a random move
- Accept this move if it leads to a solution better than a certain level
- Start again by gradually increasing the level (for a function to be maximized)



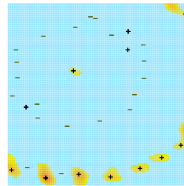
(a)



(b)



(c)

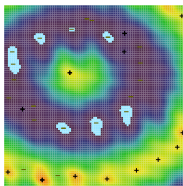


(d)

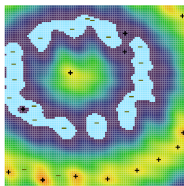
7.3 Great Deluge

Idea

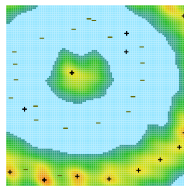
- Draw a random move
- Accept this move if it leads to a solution better than a certain level
- Start again by gradually increasing the level (for a function to be maximized)



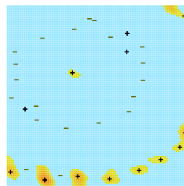
(a)



(b)



(c)

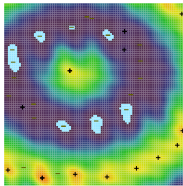


(d)

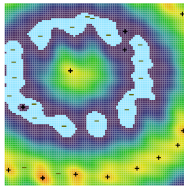
7.3 Great Deluge

Idea

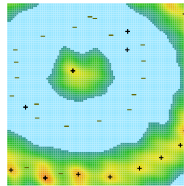
- Draw a random move
- Accept this move if it leads to a solution better than a certain level
- Start again by gradually increasing the level (for a function to be maximized)



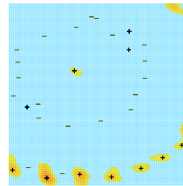
(a)



(b)



(c)



(d)

Great Deluge Frame

Input: Solution s , fitness function f to maximize, neighbourhood structure M , parameters L and P

Result: s^*

```
1  $s^* \leftarrow s$ 
2 while  $f(s) > L$  do
3   Generate a random move  $m \in M$ 
4   if  $f(s \oplus m) > L$  then
5      $s \leftarrow s \oplus m$ 
6     if  $f(s) < f(s^*)$  then
7        $s^* \leftarrow s$ 
8      $L \leftarrow L + P$ 
```

7.4 Demon Algorithm

Idea

- Simulate a player caught by the game demon
- The player enters the casino with a certain amount
- As long as he has credit, the player bets
- There has a gain if a random move improves the current solution (which becomes the new current solution)
- There is a loss if it degrades the current solution
- He cannot bet if the degradation is too great (the current solution is not modified)
- If the player's credit exceeds a certain threshold, he blows the amount that exceeds this threshold

In terms of optimization process: navigate randomly through the solution space, but never visiting a solution worse than the best one found, increased by a certain threshold

7.4 Demon Algorithm

Idea

- Simulate a player caught by the game demon
- The player enters the casino with a certain amount
- As long as he has credit, the player bets
- There has a gain if a random move improves the current solution (which becomes the new current solution)
- There is a loss if it degrades the current solution
- He cannot bet if the degradation is too great (the current solution is not modified)
- If the player's credit exceeds a certain threshold, he blows the amount that exceeds this threshold

In terms of optimization process: navigate randomly through the solution space, but never visiting a solution worse than the best one found, increased by a certain threshold

7.4 Demon Algorithm

Idea

- Simulate a player caught by the game demon
- The player enters the casino with a certain amount
- As long as he has credit, the player bets
- There has a gain if a random move improves the current solution (which becomes the new current solution)
- There is a loss if it degrades the current solution
- He cannot bet if the degradation is too great (the current solution is not modified)
- If the player's credit exceeds a certain threshold, he blows the amount that exceeds this threshold

In terms of optimization process: navigate randomly through the solution space, but never visiting a solution worse than the best one found, increased by a certain threshold

7.4 Demon Algorithm

Idea

- Simulate a player caught by the game demon
- The player enters the casino with a certain amount
- As long as he has credit, the player bets
- There has a gain if a random move improves the current solution (which becomes the new current solution)
- There is a loss if it degrades the current solution
- He cannot bet if the degradation is too great (the current solution is not modified)
- If the player's credit exceeds a certain threshold, he blows the amount that exceeds this threshold

In terms of optimization process: navigate randomly through the solution space, but never visiting a solution worse than the best one found, increased by a certain threshold

7.4 Demon Algorithm

Idea

- Simulate a player caught by the game demon
- The player enters the casino with a certain amount
- As long as he has credit, the player bets
- There has a gain if a random move improves the current solution (which becomes the new current solution)
- There is a loss if it degrades the current solution
- He cannot bet if the degradation is too great (the current solution is not modified)
- If the player's credit exceeds a certain threshold, he blows the amount that exceeds this threshold

In terms of optimization process: navigate randomly through the solution space, but never visiting a solution worse than the best one found, increased by a certain threshold

7.4 Demon Algorithm

Idea

- Simulate a player caught by the game demon
- The player enters the casino with a certain amount
- As long as he has credit, the player bets
- There has a gain if a random move improves the current solution (which becomes the new current solution)
- There is a loss if it degrades the current solution
- He cannot bet if the degradation is too great (the current solution is not modified)
- If the player's credit exceeds a certain threshold, he blows the amount that exceeds this threshold

In terms of optimization process: navigate randomly through the solution space, but never visiting a solution worse than the best one found, increased by a certain threshold

7.4 Demon Algorithm

Idea

- Simulate a player caught by the game demon
- The player enters the casino with a certain amount
- As long as he has credit, the player bets
- There has a gain if a random move improves the current solution (which becomes the new current solution)
- There is a loss if it degrades the current solution
- He cannot bet if the degradation is too great (the current solution is not modified)
- If the player's credit exceeds a certain threshold, he blows the amount that exceeds this threshold

In terms of optimization process: navigate randomly through the solution space, but never visiting a solution worse than the best one found, increased by a certain threshold

Demon Algorithm Frame

Input: Solution s , fitness function f to minimize, neighbourhood structure M , parameters

I_{max}, D, D_{max}

Result: s^*

```

1  $s^* \leftarrow s$ 
2 for  $I_{max}$  iterations do
3   Randomly generate a move  $m \in M$ 
4    $\Delta = f(s \oplus m) - f(s)$ 
5   if  $\Delta \leq D$  then
6      $D \leftarrow D - \Delta$ 
7     if  $D > D_{max}$  then
8        $D \leftarrow D_{max}$ 
9      $s \leftarrow s \oplus m$ 
10    if  $f(s^*) > f(s)$  then
11       $s^* \leftarrow s$ 
  
```

7.5 Noising Methods

Conceptually, the above methods are very similar

The noising methods group them into a more general frame

- Choose a sequence of distribution functions $Noise_k$ with (usually) standard deviations such that : $\sigma(Noise_{k+1}) < \sigma(Noise_k)$
- At iteration k , add a random noise distributed according to the function $Noise_k$:
 - Either to the problem data
 - Or to the move evaluation
- Frame : Similar to an improvement method
- The noise functions can be chosen dynamically during the iterations
- Countless variants can be implemented depending on how the noise functions are chosen
- With appropriate choices one can implement simulated annealing, threshold accepting or a great deluge algorithms

7.5 Noising Methods

Conceptually, the above methods are very similar

The noising methods group them into a more general frame

- Choose a sequence of distribution functions $Noise_k$ with (usually) standard deviations such that : $\sigma(Noise_{k+1}) < \sigma(Noise_k)$
- At iteration k , add a random noise distributed according to the function $Noise_k$:
 - Either to the problem data
 - Or to the move evaluation
- Frame : Similar to an improvement method
- The noise functions can be chosen dynamically during the iterations
- Countless variants can be implemented depending on how the noise functions are chosen
- With appropriate choices one can implement simulated annealing, threshold accepting or a great deluge algorithms

7.5 Noising Methods

Conceptually, the above methods are very similar

The noising methods group them into a more general frame

- Choose a sequence of distribution functions $Noise_k$ with (usually) standard deviations such that : $\sigma(Noise_{k+1}) < \sigma(Noise_k)$
- At iteration k , add a random noise distributed according to the function $Noise_k$:
 - Either to the problem data
 - Or to the move evaluation
- Frame : Similar to an improvement method
- The noise functions can be chosen dynamically during the iterations
- Countless variants can be implemented depending on how the noise functions are chosen
- With appropriate choices one can implement simulated annealing, threshold accepting or a great deluge algorithms

7.5 Noising Methods

Conceptually, the above methods are very similar

The noising methods group them into a more general frame

- Choose a sequence of distribution functions $Noise_k$ with (usually) standard deviations such that : $\sigma(Noise_{k+1}) < \sigma(Noise_k)$
- At iteration k , add a random noise distributed according to the function $Noise_k$:
 - Either to the problem data
 - Or to the move evaluation
- Frame : Similar to an improvement method
- The noise functions can be chosen dynamically during the iterations
- Countless variants can be implemented depending on how the noise functions are chosen
- With appropriate choices one can implement simulated annealing, threshold accepting or a great deluge algorithms

7.5 Noising Methods

Conceptually, the above methods are very similar

The noising methods group them into a more general frame

- Choose a sequence of distribution functions $Noise_k$ with (usually) standard deviations such that : $\sigma(Noise_{k+1}) < \sigma(Noise_k)$
- At iteration k , add a random noise distributed according to the function $Noise_k$:
 - Either to the problem data
 - Or to the move evaluation
- Frame : Similar to an improvement method
- The noise functions can be chosen dynamically during the iterations
- Countless variants can be implemented depending on how the noise functions are chosen
- With appropriate choices one can implement simulated annealing, threshold accepting or a great deluge algorithms

7.5 Noising Methods

Conceptually, the above methods are very similar

The noising methods group them into a more general frame

- Choose a sequence of distribution functions $Noise_k$ with (usually) standard deviations such that : $\sigma(Noise_{k+1}) < \sigma(Noise_k)$
- At iteration k , add a random noise distributed according to the function $Noise_k$:
 - Either to the problem data
 - Or to the move evaluation
- Frame : Similar to an improvement method
- The noise functions can be chosen dynamically during the iterations
- Countless variants can be implemented depending on how the noise functions are chosen
- With appropriate choices one can implement simulated annealing, threshold accepting or a great deluge algorithms

7.5 Noising Methods

Conceptually, the above methods are very similar

The noising methods group them into a more general frame

- Choose a sequence of distribution functions $Noise_k$ with (usually) standard deviations such that : $\sigma(Noise_{k+1}) < \sigma(Noise_k)$
- At iteration k , add a random noise distributed according to the function $Noise_k$:
 - Either to the problem data
 - Or to the move evaluation
- Frame : Similar to an improvement method
- The noise functions can be chosen dynamically during the iterations
- Countless variants can be implemented depending on how the noise functions are chosen
- With appropriate choices one can implement simulated annealing, threshold accepting or a great deluge algorithms

7.5 Noising Methods

Conceptually, the above methods are very similar

The noising methods group them into a more general frame

- Choose a sequence of distribution functions $Noise_k$ with (usually) standard deviations such that : $\sigma(Noise_{k+1}) < \sigma(Noise_k)$
- At iteration k , add a random noise distributed according to the function $Noise_k$:
 - Either to the problem data
 - Or to the move evaluation
- Frame : Similar to an improvement method
- The noise functions can be chosen dynamically during the iterations
- Countless variants can be implemented depending on how the noise functions are chosen
- With appropriate choices one can implement simulated annealing, threshold accepting or a great deluge algorithms

Noising Methods Frame

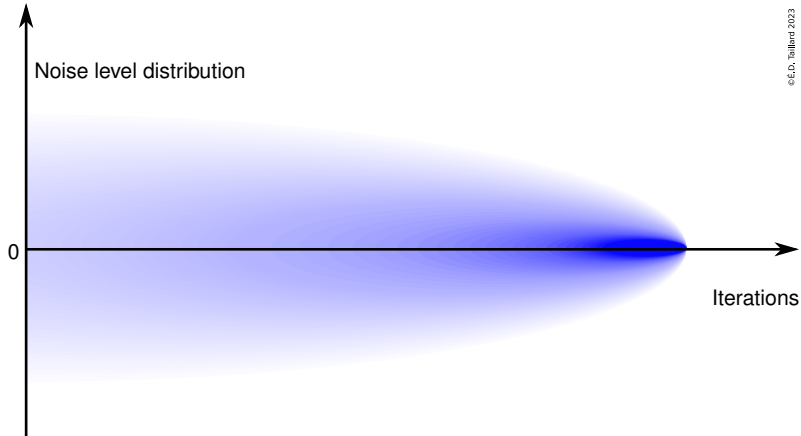
Input: Solution s , fitness function f to minimize, neighbourhood structure M , parameters $l_{max}, noise(i)$

Result: s^*

```

1  $s^* \leftarrow s$ 
2 forall  $i \in 1 \dots l_{max}$  do
3   Randomly generate a move  $m \in M$ 
4   if  $f(s \oplus m) + noise(i) < f(s)$  then
5      $s \leftarrow s \oplus m$ 
6     if  $f(s) < f(s^*)$  then
7        $s^* \leftarrow s$ 
  
```

Evolution of Noise Distribution as a Function of Iteration Number



7.6 Late Acceptance Hill Climbing

Input: Solution s , fitness function f to minimize, neighbourhood structure M , parameters h ,
stopping criterion

Result: Improved solution s

```
1 for  $k \in 0 \dots h - 1$  do
2    $L_k \leftarrow f(s)$ 
3  $i \leftarrow 0$ 
4 repeat
5   Randomly select a move  $m \in M$ 
6   if  $f(s \oplus m) < L_i$  or  $f(s \oplus m) \leq f(s)$  then
7      $s \leftarrow s \oplus m$ 
8   if  $f(s) < L_i$  then
9      $L_i \leftarrow f(s)$ 
10   $i \leftarrow (i + 1) \bmod h$ 
11 until The stopping criterion is satisfied
```

7.7 Variable Neighbourhood Search(VNS)

Idea

- Use several neighbourhoods
- Perform a random modification of the current solution with a neighbourhood
- Find the associated local optimum for a given neighbourhood
- If this optimum is better, it replaces the current solution
- Otherwise, start again using a different neighbourhood for the random jump
- In general, neighbourhoods are increasingly disrupting the current solution

7.7 Variable Neighbourhood Search(VNS)

Idea

- Use several neighbourhoods
- Perform a random modification of the current solution with a neighbourhood
- Find the associated local optimum for a given neighbourhood
- If this optimum is better, it replaces the current solution
- Otherwise, start again using a different neighbourhood for the random jump
- In general, neighbourhoods are increasingly disrupting the current solution

7.7 Variable Neighbourhood Search(VNS)

Idea

- Use several neighbourhoods
- Perform a random modification of the current solution with a neighbourhood
- Find the associated local optimum for a given neighbourhood
- If this optimum is better, it replaces the current solution
- Otherwise, start again using a different neighbourhood for the random jump
- In general, neighbourhoods are increasingly disrupting the current solution

7.7 Variable Neighbourhood Search(VNS)

Idea

- Use several neighbourhoods
- Perform a random modification of the current solution with a neighbourhood
- Find the associated local optimum for a given neighbourhood
- If this optimum is better, it replaces the current solution
- Otherwise, start again using a different neighbourhood for the random jump
- In general, neighbourhoods are increasingly disrupting the current solution

7.7 Variable Neighbourhood Search(VNS)

Idea

- Use several neighbourhoods
- Perform a random modification of the current solution with a neighbourhood
- Find the associated local optimum for a given neighbourhood
- If this optimum is better, it replaces the current solution
- Otherwise, start again using a different neighbourhood for the random jump
- In general, neighbourhoods are increasingly disrupting the current solution

7.7 Variable Neighbourhood Search(VNS)

Idea

- Use several neighbourhoods
- Perform a random modification of the current solution with a neighbourhood
- Find the associated local optimum for a given neighbourhood
- If this optimum is better, it replaces the current solution
- Otherwise, start again using a different neighbourhood for the random jump
- In general, neighbourhoods are increasingly disrupting the current solution

Variable Neighbourhood SearchFrame

Input: Solution s , fitness function f to minimize, neighbourhood structures $M_1 \dots M_p$

Result: s^*

```
1  $s^* \leftarrow s$ 
2  $k \leftarrow 1$ 
3 while  $k \leq p$  do
4   Randomly generate a move  $m \in M_k$ 
5    $s \leftarrow s \oplus m$ 
6   Find the local optimum  $s'$  associated with  $s$  in neighbourhood  $M_1$ 
7   if  $f(s') < f(s^*)$  then
8      $s^* \leftarrow s'$ 
9      $k \leftarrow 1$ 
10  else
11     $s \leftarrow s^*$ 
12     $k \leftarrow k + 1$ 
```

7.8 GRASP(Greedy Randomized Adaptive Search Procedure)

Idea

- Build a solution in a greedy but non-deterministic way
- Randomly select the new element completing the partial solution from a fraction of those that cost the least
- Apply a local search on the constructed solution
- Repeat this I_{max} times

7.8 GRASP(Greedy Randomized Adaptive Search Procedure)

Idea

- Build a solution in a greedy but non-deterministic way
- Randomly select the new element completing the partial solution from a fraction of those that cost the least
- Apply a local search on the constructed solution
- Repeat this I_{max} times

7.8 GRASP(Greedy Randomized Adaptive Search Procedure)

Idea

- Build a solution in a greedy but non-deterministic way
- Randomly select the new element completing the partial solution from a fraction of those that cost the least
- Apply a local search on the constructed solution
- Repeat this I_{max} times

7.8 GRASP(Greedy Randomized Adaptive Search Procedure)

Idea

- Build a solution in a greedy but non-deterministic way
- Randomly select the new element completing the partial solution from a fraction of those that cost the least
- Apply a local search on the constructed solution
- Repeat this I_{max} times

GRASP Frame

Input: Set E of elements constituting a solution; incremental cost function $c(s, e)$; fitness function f to minimize, parameters I_{max} and $0 \leq \alpha \leq 1$, improvement method *local_search*

Result: Complete solution s^*

```

1  $f^* \leftarrow \infty$ 
2 for  $I_{max}$  iterations do
3   Initialize  $s$  to a trivial partial solution
4    $R \leftarrow E$  // Elements that can be added to  $s$ 
5   while  $R \neq \emptyset$  do
6     Find  $c_{min} = \min_{e \in R} c(s, e)$  and  $c_{max} = \max_{e \in R} c(s, e)$ 
7     Choose randomly, uniformly  $e' \in R$  such that  $c_{min} \leq c(s, e') \leq c_{min} + \alpha(c_{max} - c_{min})$ 
8      $s \leftarrow s \cup e'$  // Include  $e'$  in the partial solution  $s$ 
9     Remove from  $R$  the elements that cannot be added any more to  $s$ 
10     $s' \leftarrow \text{local\_search}(s)$  // Find the local optimum associated with  $s$ 
11    if  $f^* > f(s')$  then
12       $f^* \leftarrow f(s')$ 
13       $s^* \leftarrow s'$ 

```

Chapter 8

Construction Learning

Chapter Content

| | | |
|---|--|-----|
| 8 | Construction Learning..... | 184 |
| • | Artificial Ants..... | 187 |
| • | Behaviour of real ants | |
| • | Transcription into an optimization process | |
| • | Max-Min Ant system | |
| • | Fast Ant System (FANT) | |
| • | Vocabulary Building..... | 194 |

Learning: Avoid relying solely on chance

Adding knowledge

- Repeat experiments and analyze successes (or failures, as one learns more by making mistakes!)
- Remember what has been done
- Forgetting certain details, which gives the ability to generalize when in a similar but different situation

Learning: Avoid relying solely on chance

Adding knowledge

- Repeat experiments and analyze successes (or failures, as one learns more by making mistakes!)
- Remember what has been done
- Forgetting certain details, which gives the ability to generalize when in a similar but different situation

Learning: Avoid relying solely on chance

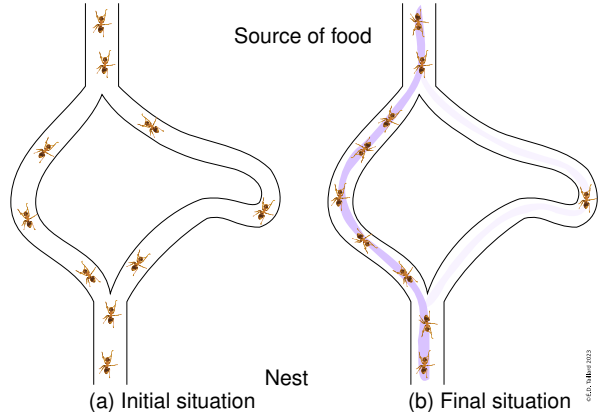
Adding knowledge

- Repeat experiments and analyze successes (or failures, as one learns more by making mistakes!)
- Remember what has been done
- Forgetting certain details, which gives the ability to generalize when in a similar but different situation

8.1 Artificial Ants

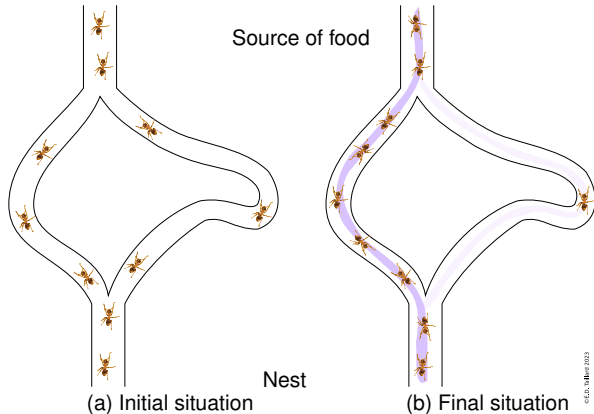
Observations with real ants

- The colony is separated from a food source by a path that forks
- Initially the ants are equally distributed in the two branches
- Those that took the shortest route deposit pheromones more quickly on the way back
- After a while, all ants use the shortest path



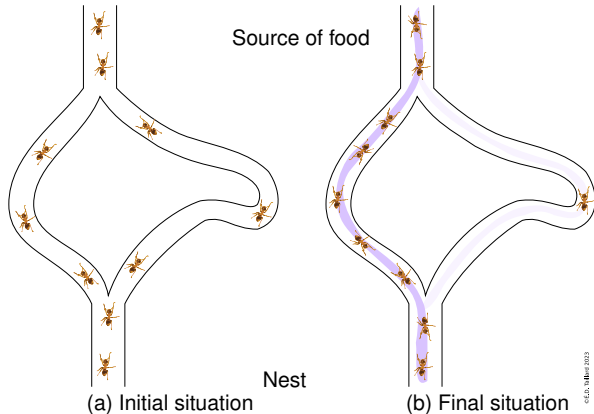
Observations with real ants

- The colony is separated from a food source by a path that forks
- Initially the ants are equally distributed in the two branches
- Those that took the shortest route deposit pheromones more quickly on the way back
- After a while, all ants use the shortest path



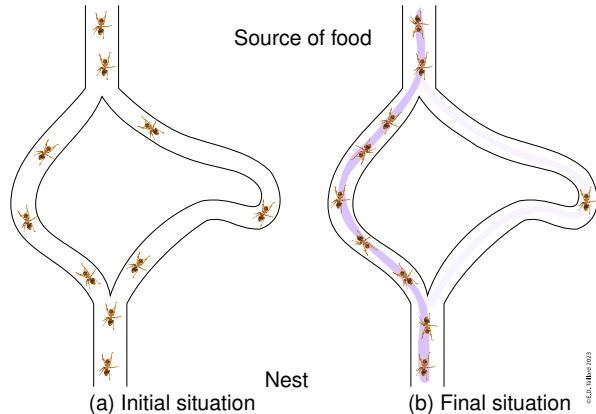
Observations with real ants

- The colony is separated from a food source by a path that forks
- Initially the ants are equally distributed in the two branches
- Those that took the shortest route deposit pheromones more quickly on the way back
- After a while, all ants use the shortest path



Observations with real ants

- The colony is separated from a food source by a path that forks
- Initially the ants are equally distributed in the two branches
- Those that took the shortest route deposit pheromones more quickly on the way back
- After a while, all ants use the shortest path



Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Transcription into an optimization process

- An ant = a process executing a solution building procedure with a random component
- Several processes can run in parallel
- Pheromone traces are τ_e values associated with each element e that can constitute a solution
- Traces are a collective memory
- After the construction of a solution, the values of its elements are increased in proportion to its quality
- The phenomenon of forgetting is simulated by the evaporation of traces over time

Idea: Adding learning to a GRASP process

- Consider the value τ_e in addition to the incremental cost function $c(s, e)$
- Choose the next element e to be included with probability proportional to $\tau_e^\alpha \cdot c(s, e)^\beta$, where $\alpha > 0$ and $\beta < 0$ are two parameters
- At each iteration, τ_e is multiplied by $1 - \rho$ ($0 \leq \rho \leq 1$) and is reinforced by a quantity $1/f(s)$

Various Ant Systems

Ant System (AS) Difficult to parameterize α , β , ρ to avoid a too fast convergence or no convergence

Ant Colony Optimization (ACO) Add a local search (*demon action*)

Max-Min Ant system Keeping the value of the traces between τ_{min} and τ_{max} facilitates the adjustment of the parameters and makes the process more robust

Fast Ant (FANT) Limit the number of parameters
Self-learning of trace reinforcement level

Various Ant Systems

Ant System (AS) Difficult to parameterize α , β , ρ to avoid a too fast convergence or no convergence

Ant Colony Optimization (ACO) Add a local search (*demon action*)

Max-Min Ant system Keeping the value of the traces between τ_{min} and τ_{max} facilitates the adjustment of the parameters and makes the process more robust

Fast Ant (FANT) Limit the number of parameters
Self-learning of trace reinforcement level

Various Ant Systems

Ant System (AS) Difficult to parameterize α , β , ρ to avoid a too fast convergence or no convergence

Ant Colony Optimization (ACO) Add a local search (*demon action*)

Max-Min Ant system Keeping the value of the traces between τ_{min} and τ_{max} facilitates the adjustment of the parameters and makes the process more robust

Fast Ant (FANT) Limit the number of parameters
Self-learning of trace reinforcement level

Various Ant Systems

Ant System (AS) Difficult to parameterize α , β , ρ to avoid a too fast convergence or no convergence

Ant Colony Optimization (ACO) Add a local search (*demon action*)

Max-Min Ant system Keeping the value of the traces between τ_{min} and τ_{max} facilitates the adjustment of the parameters and makes the process more robust

Fast Ant (FANT) Limit the number of parameters
Self-learning of trace reinforcement level

Max-Min Ant system Frame

Input: Set E of elements constituting a solution; incremental cost function $c(s, e) > 0$; fitness function f to minimize, parameters $l_{max}, m, \alpha, \beta, \tau_{min}, \tau_{max}, \rho$ and improvement method $a(\cdot)$

Result: Solution s^*

```

1   $f^* \leftarrow \infty$ 
2  for  $\forall e \in E$  do
3       $\tau_e \leftarrow \tau_{max}$ 
4  for  $l_{max}$  iterations do
5      for  $k = 1 \dots m$  do
6          Initialize  $s$  as a trivial, partial solution
7           $R \leftarrow E$                                      // Elements that can be added to  $s$ 
8          while  $R \neq \emptyset$  do Build a new solution
9              Randomly choose  $e \in R$  with a probability proportional to  $\tau_e^\alpha \cdot c(s, e)^\beta$            // Ant colony formula
10              $s \leftarrow s \cup e$ 
11             From  $R$ , remove the elements that cannot be added any more to  $s$ 
12
13              $s_k \leftarrow a(s)$                                      // Find the local optimum  $s_k$  associated with  $s$ 
14             if  $f^* > f(s_k)$  then Update the best solution found
15                  $f^* \leftarrow f(s_k)$ 
16                  $s^* \leftarrow s_k$ 
16  for  $\forall e \in E$  do Pheromone trail evaporation
17       $\tau_e \leftarrow \max(\tau_{min}, (1 - \rho) \cdot \tau_e)$ 
18   $s_b \leftarrow$  best solution from  $\{s_1, \dots, s_m\}$ 
19  for  $\forall e \in s_b$  do Update trail, maintaining it between the bounds
20       $\tau_e \leftarrow \min(\tau_{max}, \tau_e + 1/f(s_b))$ 

```

FANT Frame

Input: Set E of elements constituting a solution; fitness function f to minimize, parameters l_{max} , τ_b and improvement method $a(\cdot)$

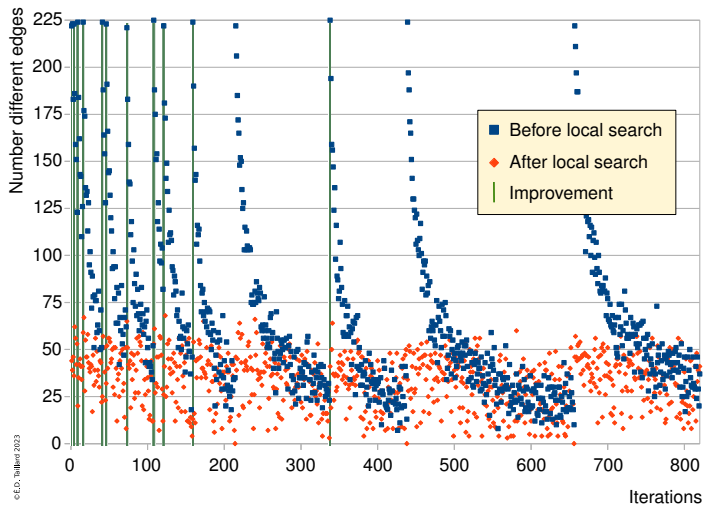
Result: Solution s^*

```

1  $f^* \leftarrow \infty$ 
2  $\tau_c \leftarrow 1$ 
3 for  $\forall e \in E$  do
4    $\tau_e \leftarrow \tau_c$ 
5 for  $l_{max}$  iterations do
6   Initialize  $s$  to a partial, trivial solution
7    $R \leftarrow E$  // Elements that can be added to  $s$ 
8   while  $R \neq \emptyset$  do
9     Randomly choose  $e \in R$  with a probability proportionnal to  $\tau_e$ 
10     $s \leftarrow s \cup e$ 
11    From  $R$ , remove the elements that cannot be added any more to  $s$ 
12   $s' \leftarrow a(s)$  // Find the local optimum  $s'$  associated with  $s$ 
13  if  $s' = s^*$  then manage over-learning // More weight to the newly constructed solutions
14     $\tau_c \leftarrow \tau_c + 1$ 
15    for  $\forall e \in E$  do Erase all trails
16       $\tau_e \leftarrow \tau_c$ 
17  if  $f^* > f(s')$  then manage best solution improvement // Update best solution
18     $f^* \leftarrow f(s')$  // Give minimum weight to the newly constructed solutions
19     $s^* \leftarrow s'$ 
20     $\tau_c \leftarrow 1$ 
21    for  $\forall e \in E$  do Erase all trails
22       $\tau_e \leftarrow \tau_c$ 
23  for  $\forall e \in s'$  do reinforce the trails associated with the current solution
24     $\tau_e \leftarrow \tau_e + \tau_c$ 
25  for  $\forall e \in s^*$  do reinforce the trails associated with the best solution
26     $\tau_e \leftarrow \tau_e + \tau_b$ 

```

Behaviour of FANT for a TSP Instance with 225 Cities



8.2 Vocabulary Building

8.2 Vocabulary Building

Idea: memorize parts (**words**) of solutions (**sentences**)

- Build new solutions from these parts
- A *dictionary* of parts is developed
- A randomized attempt of sentence is constructed from the words in the dictionary

8.2 Vocabulary Building

Idea: memorize parts (**words**) of solutions (**sentences**)

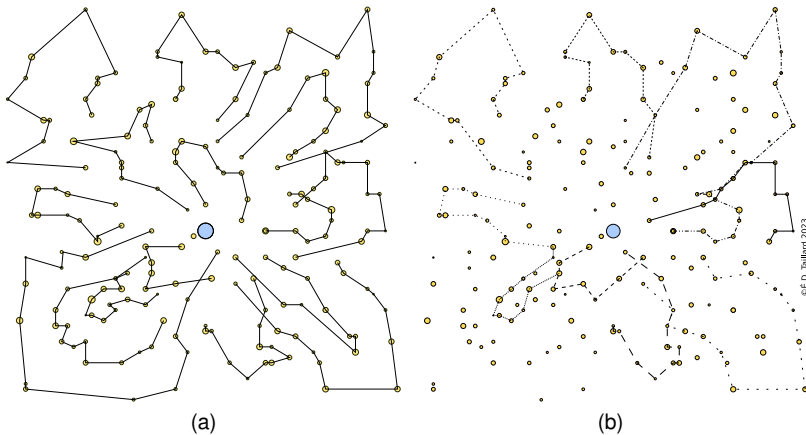
- Build new solutions from these parts
- A *dictionary* of parts is developed
- A randomized attempt of sentence is constructed from the words in the dictionary

8.2 Vocabulary Building

Idea: memorize parts (**words**) of solutions (**sentences**)

- Build new solutions from these parts
- A *dictionary* of parts is developed
- A randomized attempt of sentence is constructed from the words in the dictionary

Vocabulary Building for the VRP

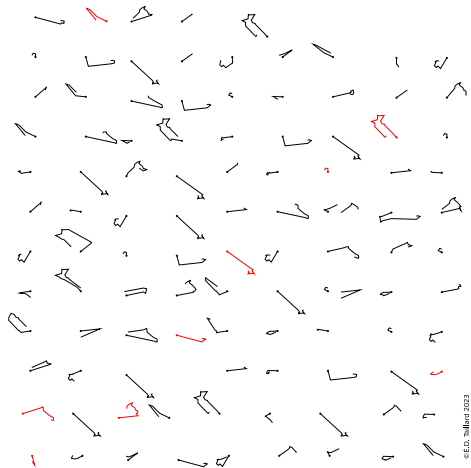


(a) The optimal solution to a VRP instance

(b) A few tours quickly obtained with a taboo search

Solution Fragments (tours) Constituting the Dictionary

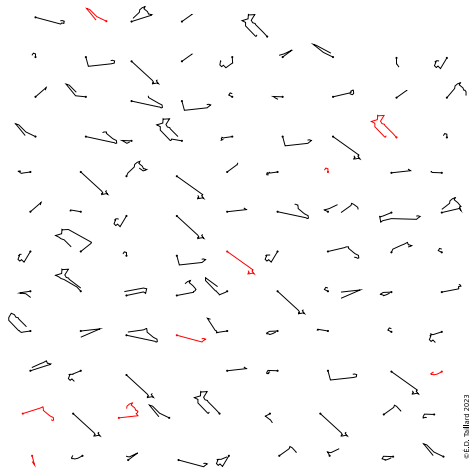
- A tour is selected from the dictionary
- Tours with customers from the last selected tour are set aside
- Repeat as long as possible



©É.D. Taillard 2023

Solution Fragments (tours) Constituting the Dictionary

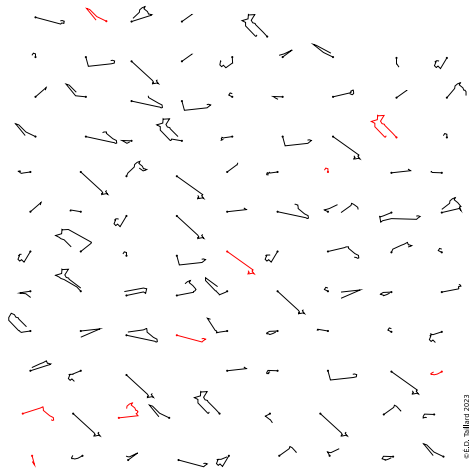
- A tour is selected from the dictionary
- Tours with customers from the last selected tour are set aside
- Repeat as long as possible



©É.D. Taillard 2023

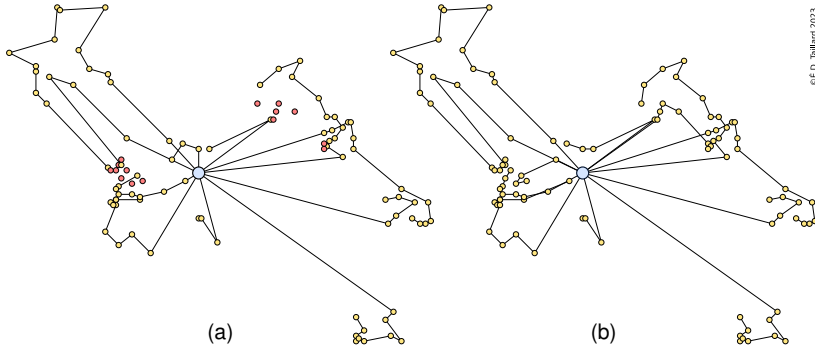
Solution Fragments (tours) Constituting the Dictionary

- A tour is selected from the dictionary
- Tours with customers from the last selected tour are set aside
- Repeat as long as possible



© E.D. Taillard 2023

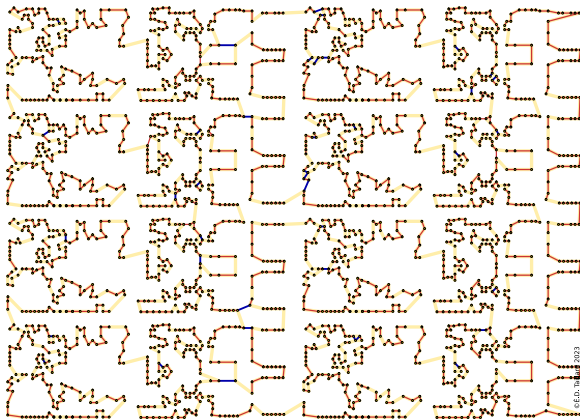
Sentence Attempt, Completion and Improvement



©É.D. Taillard 2023

Fragments of solutions (chains) for a TSP

Edges appearing in more than $2/3$ of 100 local optima (dark blue: not in global optimum)



Chapter 9

Local Search Learning

Chapter Content

| | | |
|---|------------------------------|-----|
| 9 | Local Search Learning | 200 |
| • | Taboo Search | 202 |
| • | • Taboo Duration | |
| • | Strategic Oscillations | 209 |
| • | • Long Term Memory | |

9.1 Taboo Search

Basic Ideas

- Local search with best move policy
- Allow degradation of the solution from one iteration to the next
- Using a memory to avoid cycling

9.1 Taboo Search

Basic Ideas

- Local search with best move policy
- Allow degradation of the solution from one iteration to the next
- Using a memory to avoid cycling

9.1 Taboo Search

Basic Ideas

- Local search with best move policy
- Allow degradation of the solution from one iteration to the next
- Using a memory to avoid cycling

Learning Process

- **Memory**

- Visited solutions
- Moves performed

- **Memory use**

- Prohibit returning to a previously visited solution (the solution is *taboo*)
- Prohibit performing the reverse of a move
- Penalize frequently used moves
- Force the use of never used moves

- **Oblivion**

- Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Learning Process

- Memory
 - Visited solutions
 - Moves performed
- Memory use
 - Prohibit returning to a previously visited solution (the solution is *taboo*)
 - Prohibit performing the reverse of a move
 - Penalize frequently used moves
 - Force the use of never used moves
- Oblivion
 - Remove a prohibition after a certain number of iterations

Elementary Taboo Search Frame

Input: Solution s , set M of moves, fitness function $f(\cdot)$ to minimize, parameters l_{max}, d

Result: Improved solution s^*

```

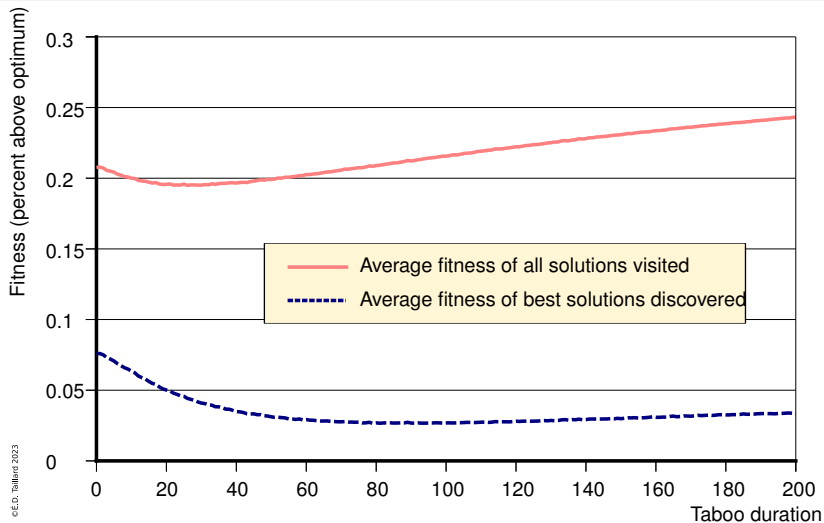
1  $s^* \leftarrow s$ 
2 for  $l_{max}$  iterations do
3    $best\_neighbour\_value \leftarrow \infty$ 
4   forall  $m \in M$  (such that  $m$  (or  $s \oplus m$ ) is not marked as taboo) do
5     if  $f(s \oplus m) < best\_neighbour\_value$  then
6        $best\_neighbour\_value \leftarrow f(s \oplus m)$ 
7        $m^* \leftarrow m$ 
8   if  $best\_neighbour\_value < \infty$  then
9     Mark  $(m^*)^{-1}$  (or  $s$ ) as taboo for the next  $d$  iterations
10     $s \leftarrow s \oplus m^*$ 
11    if  $f(s) < f(s^*)$  then
12       $s^* \leftarrow s$ 
13  else
14    Error message:  $d$  too large: no move allowed!
  
```

Elementary Taboo Search for the Knapsack Problem

$$\begin{aligned} \max r = & 12s_1 + 10s_2 + 9s_3 + 7s_4 + 4s_5 + 8s_6 + 11s_7 + 6s_8 + 13s_9 \\ \text{S.t. } & 10s_1 + 12s_2 + 8s_3 + 7s_4 + 5s_5 + 13s_6 + 9s_7 + 6s_8 + 14s_9 \leq 45 \\ & s_i \in \{0, 1\} \quad (i = 1, \dots, 9) \end{aligned}$$

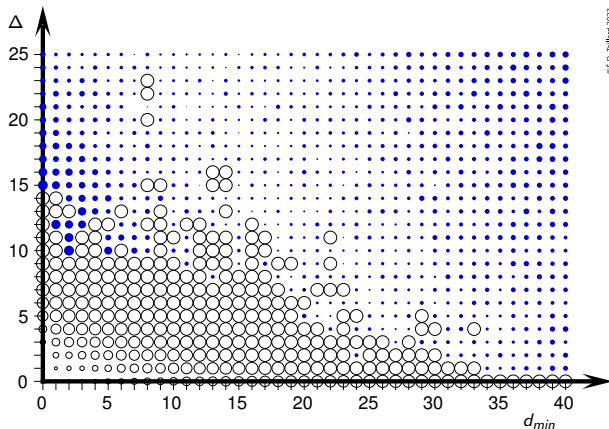
| Iter. | Var. Modif. | Solution | r | v | Taboo status (duration $d = 3$) |
|-------|-------------|-----------------------------|-----|-----|----------------------------------|
| 0 | — | (0, 0, 0, 0, 0, 0, 0, 0, 0) | 0 | 0 | (0, 0, 0, 0, 0, 0, 0, 0, 0) |
| 1 | s_9 | (0, 0, 0, 0, 0, 0, 0, 0, 1) | 13 | 14 | (0, 0, 0, 0, 0, 0, 0, 0, 4) |
| 2 | s_1 | (1, 0, 0, 0, 0, 0, 0, 0, 1) | 25 | 24 | (5, 0, 0, 0, 0, 0, 0, 0, 4) |
| 3 | s_7 | (1, 0, 0, 0, 0, 0, 1, 0, 1) | 36 | 33 | (5, 0, 0, 0, 0, 0, 6, 0, 4) |
| 4 | s_2 | (1, 1, 0, 0, 0, 0, 1, 0, 1) | 46 | 45 | (5, 7, 0, 0, 0, 0, 6, 0, 4) |
| 5 | s_9 | (1, 1, 0, 0, 0, 0, 1, 0, 0) | 33 | 31 | (5, 7, 0, 0, 0, 0, 6, 0, 8) |
| 6 | s_3 | (1, 1, 1, 0, 0, 0, 1, 0, 0) | 42 | 39 | (5, 7, 9, 0, 0, 0, 6, 0, 8) |
| 7 | s_8 | (1, 1, 1, 0, 0, 0, 1, 1, 0) | 48 | 45 | (5, 7, 9, 0, 0, 0, 6, 10, 8) |
| 8 | s_2 | (1, 0, 1, 0, 0, 0, 1, 1, 0) | 38 | 33 | (5, 11, 9, 0, 0, 0, 6, 10, 8) |
| 9 | s_4 | (1, 0, 1, 1, 0, 0, 1, 1, 0) | 45 | 40 | (5, 11, 9, 12, 0, 0, 6, 10, 8) |
| 10 | s_5 | (1, 0, 1, 1, 1, 0, 1, 1, 0) | 49 | 45 | (5, 11, 9, 12, 13, 0, 6, 10, 8) |

Influence of Taboo Duration



Random Taboo Duration for the QAP

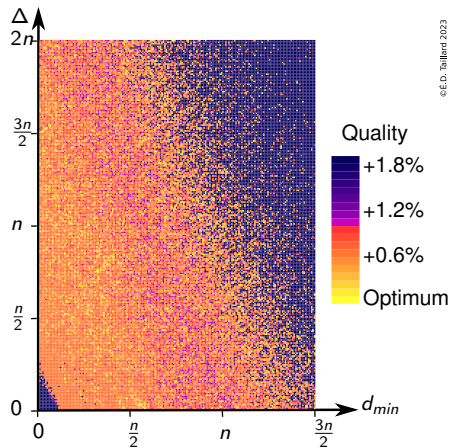
Number of iterations to reach the optimum (disc diameter) or number of times the optimum is found (diameter of empty circles)



© É.D. Taillard / 2023

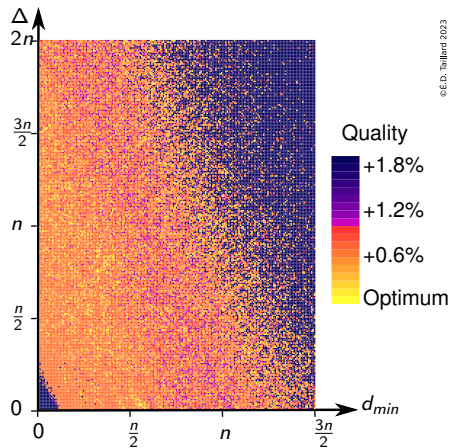
Taboo Duration Choice

- Taboo duration too short: cycling
- Taboo duration too long: good solutions are not allowed to be visited
- Remedy: Randomly draw taboo duration between d_{min} and $d_{min} + \Delta$
- Example for a TSP with $n = 52$ cities performing $10n$ iterations



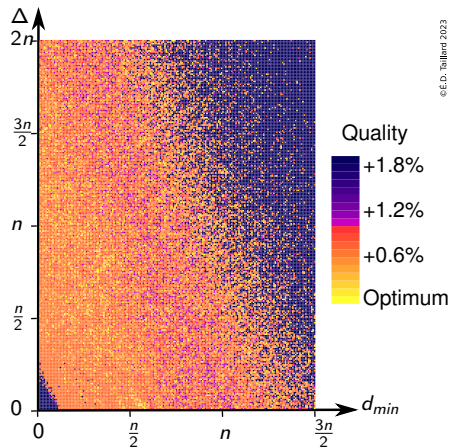
Taboo Duration Choice

- Taboo duration too short: cycling
- Taboo duration too long: good solutions are not allowed to be visited
- Remedy: Randomly draw taboo duration between d_{min} and $d_{min} + \Delta$
- Example for a TSP with $n = 52$ cities performing $10n$ iterations



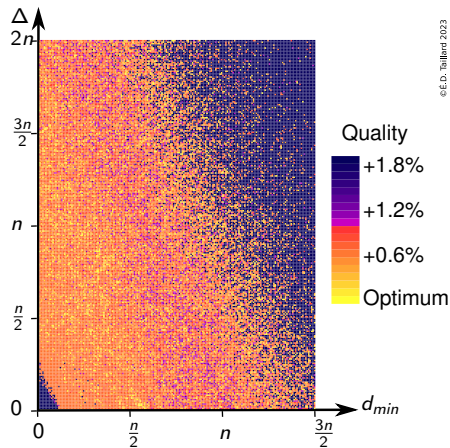
Taboo Duration Choice

- Taboo duration too short: cycling
- Taboo duration too long: good solutions are not allowed to be visited
- Remedy: Randomly draw taboo duration between d_{min} and $d_{min} + \Delta$
- Example for a TSP with $n = 52$ cities performing $10n$ iterations



Taboo Duration Choice

- Taboo duration too short: cycling
- Taboo duration too long: good solutions are not allowed to be visited
- Remedy: Randomly draw taboo duration between d_{min} and $d_{min} + \Delta$
- Example for a TSP with $n = 52$ cities performing $10n$ iterations



9.2 Strategic Oscillations

Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- **Reactive Taboo Search (RTS)**
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- **Reactive Taboo Search (RTS)**
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- **Reactive Taboo Search (RTS)**
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- **Reactive Taboo Search (RTS)**
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- **Reactive Taboo Search (RTS)**
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- Reactive Taboo Search (RTS)
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- Reactive Taboo Search (RTS)
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- Reactive Taboo Search (RTS)
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

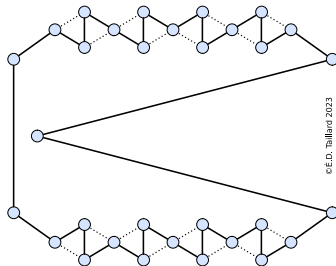
Strategic Oscillations

Alternate phases of intensification and diversification

- **Intensification:** Forbidding a move or solution for d iterations implements a **short term** memory
 - Lower the value of d
 - Start again from best solution found
- **Diversification:** Try to go elsewhere
 - By penalizing frequently performed moves
 - By forcing to do a very rarely performed move, regardless of its evaluation
 - Starting with a new solution
- Reactive Taboo Search (RTS)
 - Increase taboo duration if visiting a solution already encountered
 - Diminish taboo duration if no solution is revisited for many iterations

Pathological Situation for the TSP

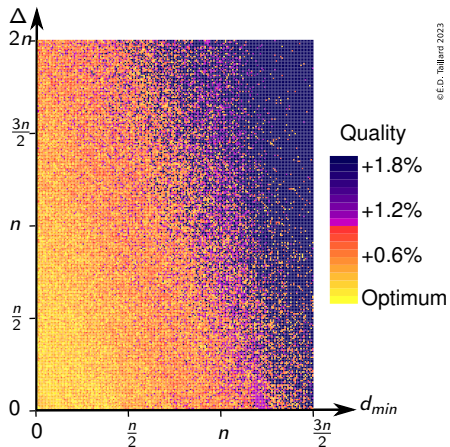
Penalizing frequently performed moves allows escape from plateaus



For this TSP tour: many 2-opt moves with 0 cost

Move Penalty for the TSP

Penalizing moves according to the frequency of their use can even eliminate the need for a taboo list!



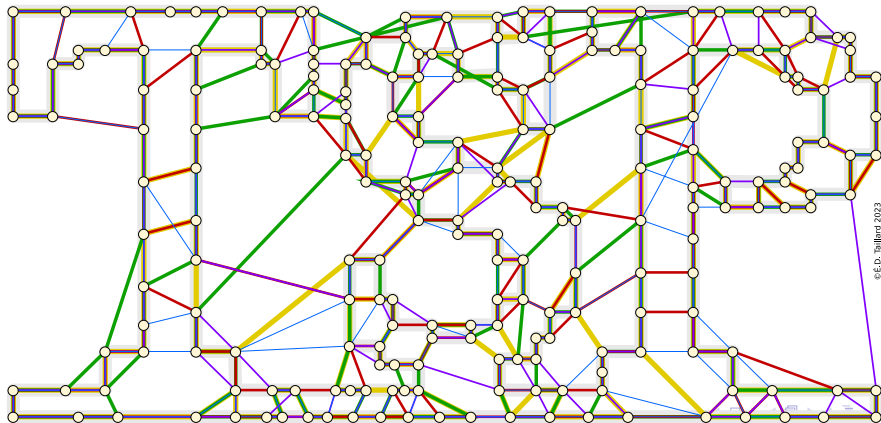
Chapter 10

Population Management

Chapter Content

| | | |
|----|--|-----|
| 10 | Population Management..... | 213 |
| • | Evolutionary Algorithms Framework..... | 217 |
| • | Genetic Algorithms..... | 221 |
| • | Selection Operators for Reproduction | |
| • | Mutation Operators | |
| • | Selection operator for survival | |
| • | Memetic Algorithms..... | 232 |
| • | Scatter Search | 233 |
| • | Example of Scatter Search for the Knapsack | |
| • | Biased Random Key Genetic Algorithm (BRKGA)..... | 237 |
| • | Path Relinking | 239 |
| • | GRASP with Path Relinking | |
| • | Fixed Set Search..... | 242 |
| • | Particle Swarm..... | 242 |

Learning with a population of solutions



Idea of Evolutionary Algorithms

- It is easy to generate many different solutions for a problem
- Build new solutions by combining previously obtained solutions
- Scatter the generated solutions throughout the solution set
- The combination of solutions can select the right features from each



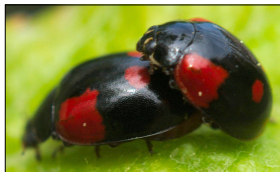
Idea of Evolutionary Algorithms

- It is easy to generate many different solutions for a problem
- Build new solutions by combining previously obtained solutions
- Scatter the generated solutions throughout the solution set
- The combination of solutions can select the right features from each



Idea of Evolutionary Algorithms

- It is easy to generate many different solutions for a problem
- Build new solutions by combining previously obtained solutions
- Scatter the generated solutions throughout the solution set
- The combination of solutions can select the right features from each



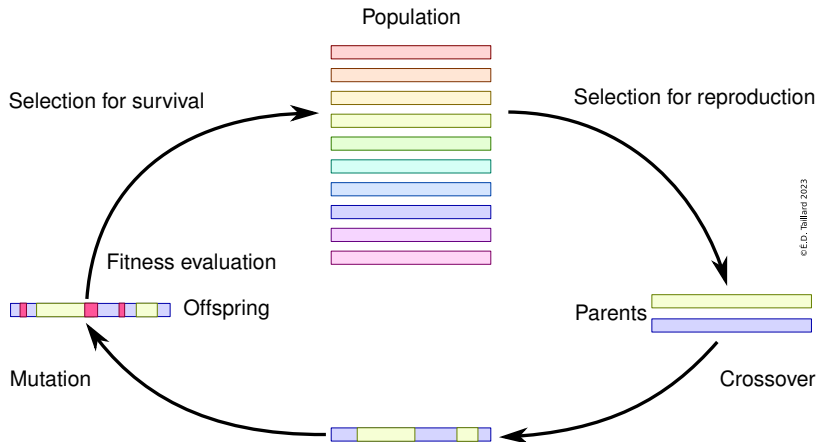
Idea of Evolutionary Algorithms

- It is easy to generate many different solutions for a problem
- Build new solutions by combining previously obtained solutions
- Scatter the generated solutions throughout the solution set
- The combination of solutions can select the right features from each



10.1 Evolutionary Algorithms Framework

Generational loop in an evolutionary algorithm



Evolutionary Algorithms Frame

Input: Parameters μ and λ , selection for reproduction, crossover, mutation and selection for survival operators

Result: Population of solutions P

- 1 Generate a population P of μ solutions
- 2 **repeat**
 - 3 Select individuals from P with the selection for reproduction operator
 - 4 Combine the selected individuals with the crossover operator and apply the mutation operator to get λ new solutions
 - 5 Among the $\mu + \lambda$ solutions, select μ individuals with the selection for survival operator; these μ individuals constitute the population P for the next generation
- 6 **until** *a stopping criterion is satisfied*

Evolutionary Algorithms Options

- Selection operators for reproduction and survival
- Solution combination operator
- Mutation operator
- Population Management
- Stopping criterion

Evolutionary Algorithms Options

- Selection operators for reproduction and survival
- Solution combination operator
- Mutation operator
- Population Management
- Stopping criterion

Evolutionary Algorithms Options

- Selection operators for reproduction and survival
- Solution combination operator
- Mutation operator
- Population Management
- Stopping criterion

Evolutionary Algorithms Options

- Selection operators for reproduction and survival
- Solution combination operator
- Mutation operator
- Population Management
- Stopping criterion

Evolutionary Algorithms Options

- Selection operators for reproduction and survival
- Solution combination operator
- Mutation operator
- Population Management
- Stopping criterion

10.2 Genetic Algorithms

Selection Operators for Reproduction

Selection by rank Assigning a quality measure $f_i = (1 - \frac{r_i}{\mu})^p$, where r_i is the rank of the i th individual (the best individual has a rank of 1 and the worst of μ) and $p \geq 0$ is a parameter to modulate the *selection pressure* ($p = 0$: uniform (no selection pressure), $p = 2$ is a relatively high pressure)

Proportional selection Individual i of quality f_i has a probability $f_i / \sum f_i$ of being selected (*roulette wheel*)

Natural selection Each individual has the same probability of being selected

Complete selection All individuals are selected for reproduction

Selection Operators for Reproduction

Selection by rank Assigning a quality measure $f_i = (1 - \frac{r_i}{\mu})^p$, where r_i is the rank of the i th individual (the best individual has a rank of 1 and the worst of μ) and $p \geq 0$ is a parameter to modulate the *selection pressure* ($p = 0$: uniform (no selection pressure), $p = 2$ is a relatively high pressure)

Proportional selection Individual i of quality f_i has a probability $f_i / \sum f_i$ of being selected (*roulette wheel*)

Natural selection Each individual has the same probability of being selected

Complete selection All individuals are selected for reproduction

Selection Operators for Reproduction

Selection by rank Assigning a quality measure $f_i = (1 - \frac{r_i}{\mu})^p$, where r_i is the rank of the i th individual (the best individual has a rank of 1 and the worst of μ) and $p \geq 0$ is a parameter to modulate the *selection pressure* ($p = 0$: uniform (no selection pressure), $p = 2$ is a relatively high pressure)

Proportional selection Individual i of quality f_i has a probability $f_i / \sum f_i$ of being selected (*roulette wheel*)

Natural selection Each individual has the same probability of being selected

Complete selection All individuals are selected for reproduction

Selection Operators for Reproduction

Selection by rank Assigning a quality measure $f_i = (1 - \frac{r_i}{\mu})^p$, where r_i is the rank of the i th individual (the best individual has a rank of 1 and the worst of μ) and $p \geq 0$ is a parameter to modulate the *selection pressure* ($p = 0$: uniform (no selection pressure), $p = 2$ is a relatively high pressure)

Proportional selection Individual i of quality f_i has a probability $f_i / \sum f_i$ of being selected (*roulette wheel*)

Natural selection Each individual has the same probability of being selected

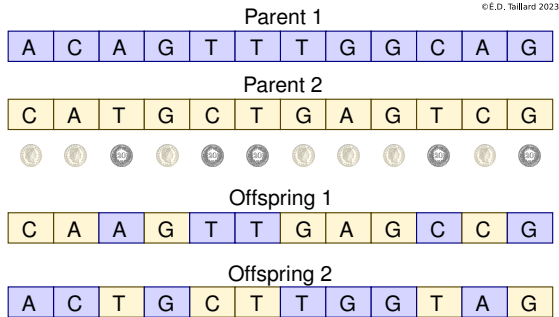
Complete selection All individuals are selected for reproduction

Selection probability for different operators for reproduction

| Fitness-function | Rank | Rank-b. ($p = 2$) | Rank-b. ($p = 1$) | Natural | Proportional |
|------------------|------|---------------------|---------------------|---------|--------------|
| 220 | 1 | 0.260 | 0.182 | 0.1 | 0.220 |
| 162 | 2 | 0.210 | 0.164 | 0.1 | 0.162 |
| 157 | 3 | 0.166 | 0.146 | 0.1 | 0.157 |
| 93 | 4 | 0.127 | 0.127 | 0.1 | 0.093 |
| 85 | 5 | 0.094 | 0.109 | 0.1 | 0.085 |
| 74 | 6 | 0.065 | 0.091 | 0.1 | 0.074 |
| 61 | 7 | 0.042 | 0.073 | 0.1 | 0.061 |
| 55 | 8 | 0.023 | 0.054 | 0.1 | 0.055 |
| 49 | 9 | 0.010 | 0.036 | 0.1 | 0.049 |
| 44 | 10 | 0.003 | 0.018 | 0.1 | 0.044 |

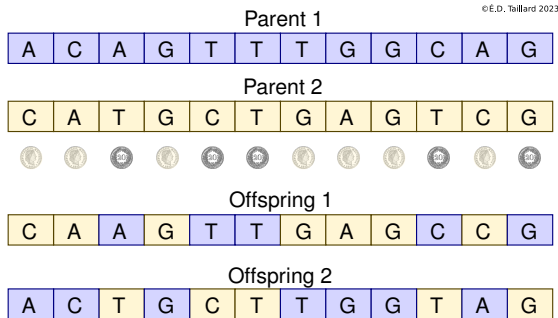
Uniform Crossover: Production of 2 children from 2 parents

- Going through the genes one by one
- Each element of the first child is drawn randomly from the first or second parent
- The other child receives the complementary elements



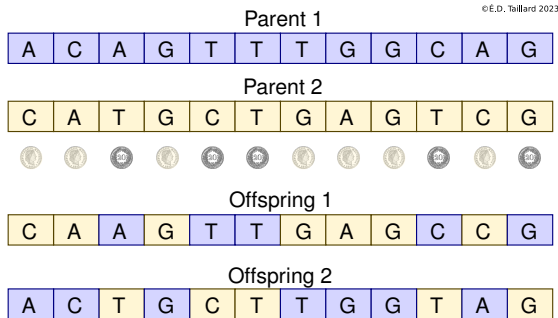
Uniform Crossover: Production of 2 children from 2 parents

- Going through the genes one by one
- Each element of the first child is drawn randomly from the first or second parent
- The other child receives the complementary elements



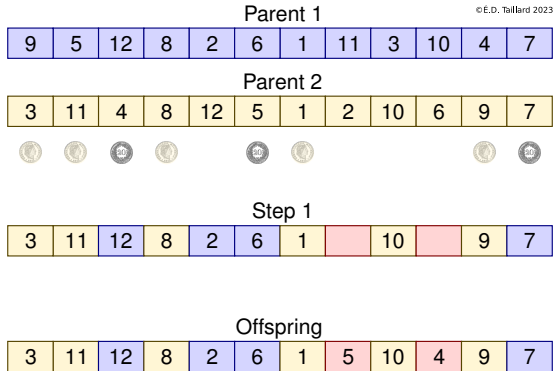
Uniform Crossover: Production of 2 children from 2 parents

- Going through the genes one by one
- Each element of the first child is drawn randomly from the first or second parent
- The other child receives the complementary elements



Uniform Crossover for Permutation

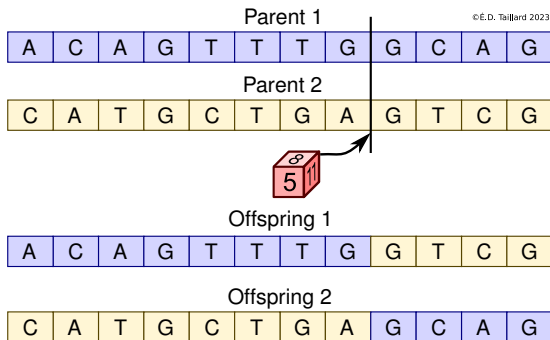
Proceed in two steps, leaving empty the cells for which no choice is possible
Randomly fill in the blanks with the missing elements



Single-Point Crossover: Production of 2 children from 2 parents

Randomly picks a point within the solution vector

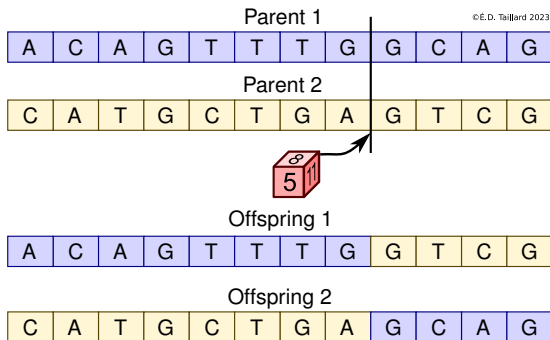
- Copy the items from the first parent up to the point, and those of the second from there for the first child
- Copy the items from the second parent up to the point, and those of the first from there for the second child



Single-Point Crossover: Production of 2 children from 2 parents

Randomly picks a point within the solution vector

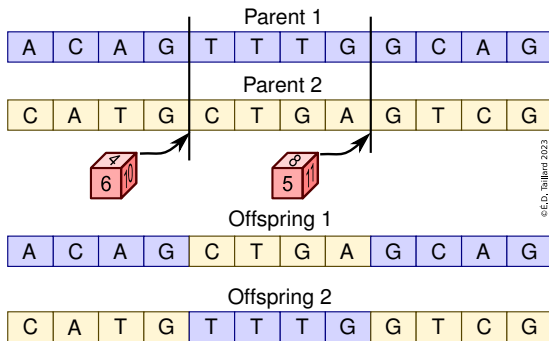
- Copy the items from the first parent up to the point, and those of the second from there for the first child
- Copy the items from the second parent up to the point, and those of the first from there for the second child



2-Points Crossover : Production of 2 children from 2 parents

Randomly picks 2 different points within the solution vector

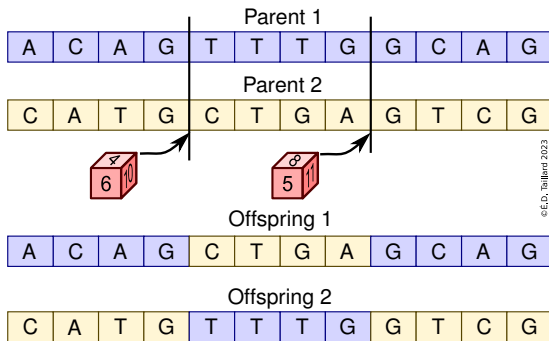
- For the first child: Copy the elements from the first parent up to the first point and from the second point up to the end, copy the intermediate part from the second parent
- For the second child: Copy the elements from the second parent up to the first point and from the second point up to the end, copy the intermediate part from the first parent



2-Points Crossover : Production of 2 children from 2 parents

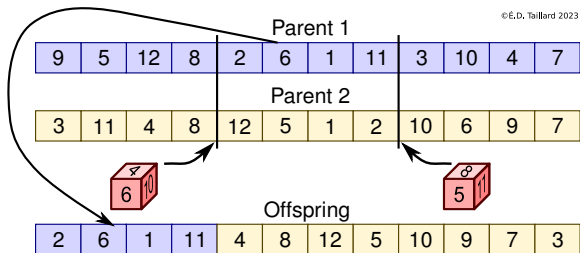
Randomly picks 2 different points within the solution vector

- For the first child: Copy the elements from the first parent up to the first point and from the second point up to the end, copy the intermediate part from the second parent
- For the second child: Copy the elements from the second parent up to the first point and from the second point up to the end, copy the intermediate part from the first parent



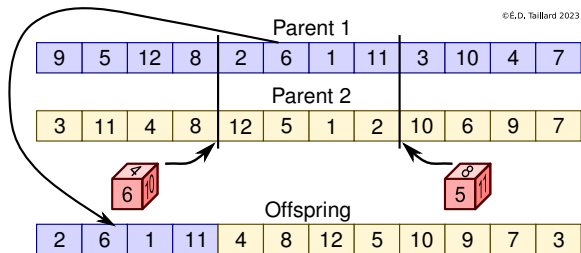
OX Crossover for the TSP : Respect the sequence of successors

- Choose 2 crossover points
- The intermediate portion is copied from the first parent
- Locate the last item copied into the child in the second parent
- Complete the child in the order in which the elements appear in the second parent (skip elements already introduced; cyclic path)



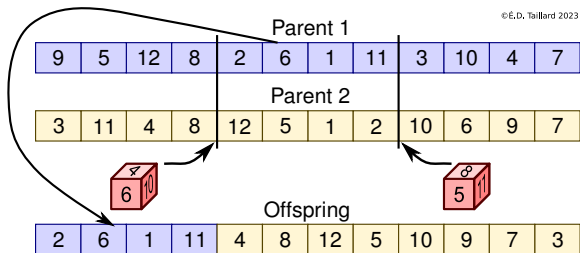
OX Crossover for the TSP : Respect the sequence of successors

- Choose 2 crossover points
- The intermediate portion is copied from the first parent
- Locate the last item copied into the child in the second parent
- Complete the child in the order in which the elements appear in the second parent (skip elements already introduced; cyclic path)



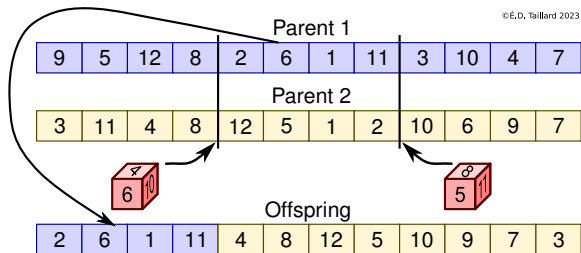
OX Crossover for the TSP : Respect the sequence of successors

- Choose 2 crossover points
- The intermediate portion is copied from the first parent
- Locate the last item copied into the child in the second parent
- Complete the child in the order in which the elements appear in the second parent (skip elements already introduced; cyclic path)



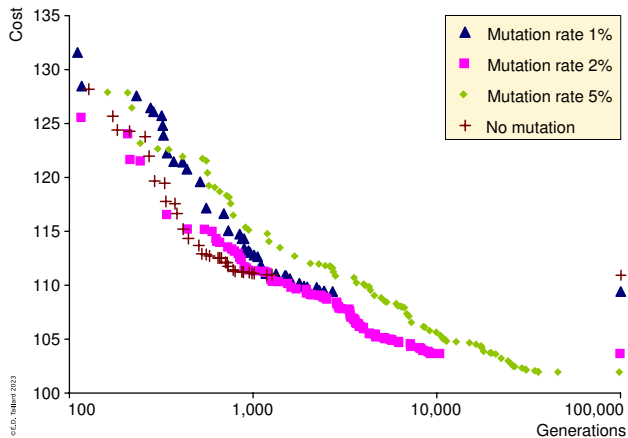
OX Crossover for the TSP : Respect the sequence of successors

- Choose 2 crossover points
- The intermediate portion is copied from the first parent
- Locate the last item copied into the child in the second parent
- Complete the child in the order in which the elements appear in the second parent (skip elements already introduced; cyclic path)



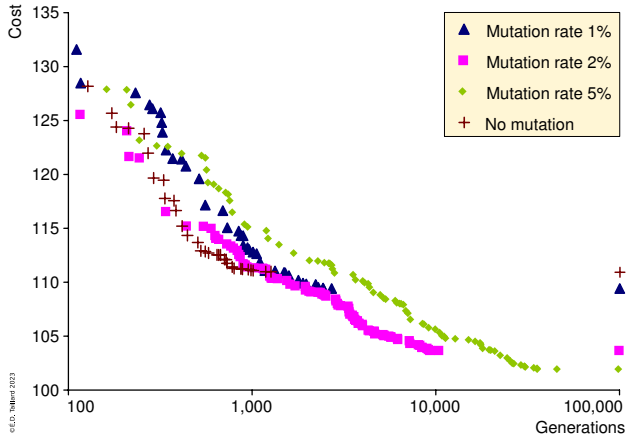
Mutation Operators: Randomly disturb the children

- Bringing diversity (new blood) into the population
- Avoid to clone individuals



Mutation Operators: Randomly disturb the children

- Bringing diversity (new blood) into the population
- Avoid to clone individuals



©É.D. Taillard 2023

Selection operator for survival

Generational replacement $\lambda = \mu$, children replace parents

Evolutionary strategy $\lambda > \mu$, the best children replace the parents

Stationary replacement $\lambda = 2$, children replace parents

Elitist replacement The μ best elements among $\mu + \lambda$ survive

Selection operator for survival

Generational replacement $\lambda = \mu$, children replace parents

Evolutionary strategy $\lambda > \mu$, the best children replace the parents

Stationary replacement $\lambda = 2$, children replace parents

Elitist replacement The μ best elements among $\mu + \lambda$ survive

Selection operator for survival

Generational replacement $\lambda = \mu$, children replace parents

Evolutionary strategy $\lambda > \mu$, the best children replace the parents

Stationary replacement $\lambda = 2$, children replace parents

Elitist replacement The μ best elements among $\mu + \lambda$ survive

Selection operator for survival

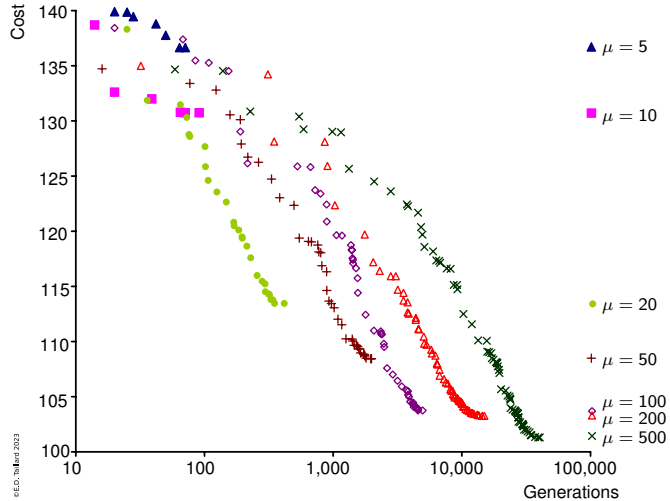
Generational replacement $\lambda = \mu$, children replace parents

Evolutionary strategy $\lambda > \mu$, the best children replace the parents

Stationary replacement $\lambda = 2$, children replace parents

Elitist replacement The μ best elements among $\mu + \lambda$ survive

Influence of population size, elitist replacement



10.3 Memetic Algorithms

Idea: Learn better by managing the population intelligently, improve solutions faster by combining a genetic algorithm with other ingredients of meta-heuristics

- Consider several small populations (islands)
 - Accelerated convergence of each population
- Define a diversity measure between solutions
 - Reject the solutions that are too similar to others in the population
 - Survival of the best solution of a population
- Create a communication network between islands (ring, grid, ...)
 - Send solutions from one population to another to diversify them
- Replace the mutation operator with a local search

10.3 Memetic Algorithms

Idea: Learn better by managing the population intelligently, improve solutions faster by combining a genetic algorithm with other ingredients of meta-heuristics

- Consider several small populations (islands)
 - Accelerated convergence of each population
- Define a diversity measure between solutions
 - Reject the solutions that are too similar to others in the population
 - Survival of the best solution of a population
- Create a communication network between islands (ring, grid, ...)
 - Send solutions from one population to another to diversify them
- Replace the mutation operator with a local search

10.3 Memetic Algorithms

Idea: Learn better by managing the population intelligently, improve solutions faster by combining a genetic algorithm with other ingredients of meta-heuristics

- Consider several small populations (islands)
 - Accelerated convergence of each population
- Define a diversity measure between solutions
 - Reject the solutions that are too similar to others in the population
 - Survival of the best solution of a population
- Create a communication network between islands (ring, grid, ...)
 - Send solutions from one population to another to diversify them
- Replace the mutation operator with a local search

10.3 Memetic Algorithms

Idea: Learn better by managing the population intelligently, improve solutions faster by combining a genetic algorithm with other ingredients of meta-heuristics

- Consider several small populations (islands)
 - Accelerated convergence of each population
- Define a diversity measure between solutions
 - Reject the solutions that are too similar to others in the population
 - Survival of the best solution of a population
- Create a communication network between islands (ring, grid, ...)
 - Send solutions from one population to another to diversify them
- Replace the mutation operator with a local search

10.3 Memetic Algorithms

Idea: Learn better by managing the population intelligently, improve solutions faster by combining a genetic algorithm with other ingredients of meta-heuristics

- Consider several small populations (islands)
 - Accelerated convergence of each population
- Define a diversity measure between solutions
 - Reject the solutions that are too similar to others in the population
 - Survival of the best solution of a population
- Create a communication network between islands (ring, grid, ...)
 - Send solutions from one population to another to diversify them
- Replace the mutation operator with a local search

10.3 Memetic Algorithms

Idea: Learn better by managing the population intelligently, improve solutions faster by combining a genetic algorithm with other ingredients of meta-heuristics

- Consider several small populations (islands)
 - Accelerated convergence of each population
- Define a diversity measure between solutions
 - Reject the solutions that are too similar to others in the population
 - Survival of the best solution of a population
- Create a communication network between islands (ring, grid, ...)
 - Send solutions from one population to another to diversify them
- Replace the mutation operator with a local search

10.3 Memetic Algorithms

Idea: Learn better by managing the population intelligently, improve solutions faster by combining a genetic algorithm with other ingredients of meta-heuristics

- Consider several small populations (islands)
 - Accelerated convergence of each population
- Define a diversity measure between solutions
 - Reject the solutions that are too similar to others in the population
 - Survival of the best solution of a population
- Create a communication network between islands (ring, grid, ...)
 - Send solutions from one population to another to diversify them
- Replace the mutation operator with a local search

10.3 Memetic Algorithms

Idea: Learn better by managing the population intelligently, improve solutions faster by combining a genetic algorithm with other ingredients of meta-heuristics

- Consider several small populations (islands)
 - Accelerated convergence of each population
- Define a diversity measure between solutions
 - Reject the solutions that are too similar to others in the population
 - Survival of the best solution of a population
- Create a communication network between islands (ring, grid, ...)
 - Send solutions from one population to another to diversify them
- Replace the mutation operator with a local search

10.4 Scatter Search

Break the taboo of a reproduction limited to the crossover of two solutions

- Initial population: as scattered as possible
- Combine several solutions with each other
- Repair/improvement operator
- Population management in elite and diverse solutions



10.4 Scatter Search

Break the taboo of a reproduction limited to the crossover of two solutions

- Initial population: as scattered as possible
- Combine several solutions with each other
- Repair/improvement operator
- Population management in elite and diverse solutions



10.4 Scatter Search

Break the taboo of a reproduction limited to the crossover of two solutions

- Initial population: as scattered as possible
- Combine several solutions with each other
- Repair/improvement operator
- Population management in elite and diverse solutions



10.4 Scatter Search

Break the taboo of a reproduction limited to the crossover of two solutions

- Initial population: as scattered as possible
- Combine several solutions with each other
- Repair/improvement operator
- Population management in elite and diverse solutions



Scatter Search Frame

Input: Size μ of the complete population, E size of the subset of elite solutions

Result: Population of solutions

- 1 Systematically generate a (large) population P of potential solutions as dispersed as possible
- 2 **repeat**
 - 3 Repair and improve the solutions from P to make them feasible using the repair/improvement operator
 - 4 Eliminate identical solutions from P
 - 5 Identify the E best solutions from the population; they are retained in the reference set as elites
 - 6 Identify from P the $\mu - E$ solutions which are the most different from the elite solutions, they are kept and complete the reference set
 - 7 Combine in all possible ways the μ solutions of the reference set to obtain $2^\mu - \mu - 1$ new potential solutions
 - 8 Join the potential solutions to the reference set to obtain the new population P of the next iteration
- 9 **until** *the population remains stable*

Example of Scatter Search for the Knapsack

$$\max r = 11s_1 + 10s_2 + 9s_3 + 12s_4 + 10s_5 + 6s_6 + 7s_7 + 5s_8 + 3s_9 + 8s_{10}$$

$$\text{Subject } 33s_1 + 27s_2 + 16s_3 + 14s_4 + 29s_5 + 30s_6 + 31s_7 + 33s_8 + 14s_9 + 18s_{10} \leq 100$$

$$\text{to: } s_i \in \{0, 1\} (i = 1, \dots, 10)$$

(1)

| | Potential solution | Value | Repaired/improved solution | Value | |
|----|-----------------------|-------|----------------------------|-------|--------------|
| 1 | (1,1,1,1,1,1,1,1,1,1) | 81 | (0,1,1,1,0,0,0,0,1,1) | 42 | |
| 2 | (1,0,1,0,1,0,1,0,1,0) | 40 | (1,0,1,1,1,0,0,0,0,0) | 42 | |
| 3 | (1,0,0,1,0,0,1,0,0,1) | 38 | (1,0,0,1,0,0,1,0,0,1) | 38 | |
| 4 | (1,0,0,0,1,0,0,0,1,0) | 24 | (1,0,0,1,1,0,0,0,1,0) | 36 | |
| 5 | (1,0,0,0,0,1,0,0,0,0) | 17 | (1,0,1,1,0,1,0,0,0,0) | 38 | |
| 6 | (0,0,0,0,0,0,0,0,0,0) | 0 | (0,1,1,1,0,0,0,0,0,1) | 39 | |
| 7 | (0,1,0,1,0,1,0,1,0,1) | 41 | (0,1,0,1,0,1,0,0,0,1) | 36 | |
| 8 | (0,1,1,0,1,1,0,1,1,0) | 43 | (0,1,1,1,1,0,0,0,1,0) | 44 | |
| 9 | (0,1,1,1,0,1,1,1,0,1) | 57 | (0,1,1,1,0,0,0,0,1,1) | 42 | = solution 1 |
| 10 | (0,1,1,1,1,0,1,1,1,1) | 64 | (0,1,1,1,0,0,0,0,1,1) | 42 | = solution 1 |

Determining the reference set and combining 3 solutions

| | Candidate Solution | Hamming Distance | | | Minimal Distance |
|---|-----------------------|------------------|---------|---------|------------------|
| | | Elite 1 | Elite 2 | Elite 8 | |
| 3 | (1,0,0,1,0,0,1,0,0,1) | 5 | 4 | 7 | 4 |
| 4 | (1,0,0,1,1,0,0,0,1,0) | 5 | 2 | 3 | 2 |
| 5 | (1,0,1,1,0,1,0,0,0,0) | 5 | 2 | 5 | 2 |
| 6 | (0,1,1,1,0,0,0,0,0,1) | 1 | 4 | 3 | 1 |
| 7 | (0,1,0,1,0,1,0,0,0,1) | 3 | 6 | 5 | 3 |

$$\frac{38}{38 + 36 + 44} \cdot (1, 0, 0, 1, 0, 0, 1, 0, 0, 1) +$$

$$\frac{36}{38 + 36 + 44} \cdot (0, 1, 0, 1, 0, 1, 0, 0, 0, 1) +$$

$$\frac{44}{38 + 36 + 44} \cdot (0, 1, 1, 1, 1, 0, 0, 0, 1, 0)$$

$$=(0.322, 0.678, 0.373, 1.000, 0.373, 0.305, 0.322, 0.000, 0.373, 0.627)$$

$$\text{Round}(0, 1, 0, 1, 0, 0, 0, 0, 0, 1)$$

10.5 Biased Random Key Genetic Algorithm (BRKGA)

Working Principles

- A solution (permutation) is coded as an array of real numbers
 - Sorting the array allows the permutation to be reconstructed
- The E best solutions are kept for the next iteration
- The selection for reproduction always chooses a solution from the elite set E
- The child is generated with a uniform crossover, but with a probability $\neq 0.5$
 - The selected elite solution is favoured
- $\lambda < \mu - E$ children are generated
- $\mu - E - \lambda$ new arrays are generated randomly to maintain diversity

10.5 Biased Random Key Genetic Algorithm (BRKGA)

Working Principles

- A solution (permutation) is coded as an array of real numbers
 - Sorting the array allows the permutation to be reconstructed
- The E best solutions are kept for the next iteration
- The selection for reproduction always chooses a solution from the elite set E
- The child is generated with a uniform crossover, but with a probability $\neq 0.5$
 - The selected elite solution is favoured
- $\lambda < \mu - E$ children are generated
- $\mu - E - \lambda$ new arrays are generated randomly to maintain diversity

10.5 Biased Random Key Genetic Algorithm (BRKGA)

Working Principles

- A solution (permutation) is coded as an array of real numbers
 - Sorting the array allows the permutation to be reconstructed
- The E best solutions are kept for the next iteration
- The selection for reproduction always chooses a solution from the elite set E
- The child is generated with a uniform crossover, but with a probability $\neq 0.5$
 - The selected elite solution is favoured
- $\lambda < \mu - E$ children are generated
- $\mu - E - \lambda$ new arrays are generated randomly to maintain diversity

10.5 Biased Random Key Genetic Algorithm (BRKGA)

Working Principles

- A solution (permutation) is coded as an array of real numbers
 - Sorting the array allows the permutation to be reconstructed
- The E best solutions are kept for the next iteration
- The selection for reproduction always chooses a solution from the elite set E
- The child is generated with a uniform crossover, but with a probability $\neq 0.5$
 - The selected elite solution is favoured
- $\lambda < \mu - E$ children are generated
- $\mu - E - \lambda$ new arrays are generated randomly to maintain diversity

10.5 Biased Random Key Genetic Algorithm (BRKGA)

Working Principles

- A solution (permutation) is coded as an array of real numbers
 - Sorting the array allows the permutation to be reconstructed
- The E best solutions are kept for the next iteration
- The selection for reproduction always chooses a solution from the elite set E
- The child is generated with a uniform crossover, but with a probability $\neq 0.5$
 - The selected elite solution is favoured
- $\lambda < \mu - E$ children are generated
- $\mu - E - \lambda$ new arrays are generated randomly to maintain diversity

10.5 Biased Random Key Genetic Algorithm (BRKGA)

Working Principles

- A solution (permutation) is coded as an array of real numbers
 - Sorting the array allows the permutation to be reconstructed
- The E best solutions are kept for the next iteration
- The selection for reproduction always chooses a solution from the elite set E
- The child is generated with a uniform crossover, but with a probability $\neq 0.5$
 - The selected elite solution is favoured
- $\lambda < \mu - E$ children are generated
- $\mu - E - \lambda$ new arrays are generated randomly to maintain diversity

10.5 Biased Random Key Genetic Algorithm (BRKGA)

Working Principles

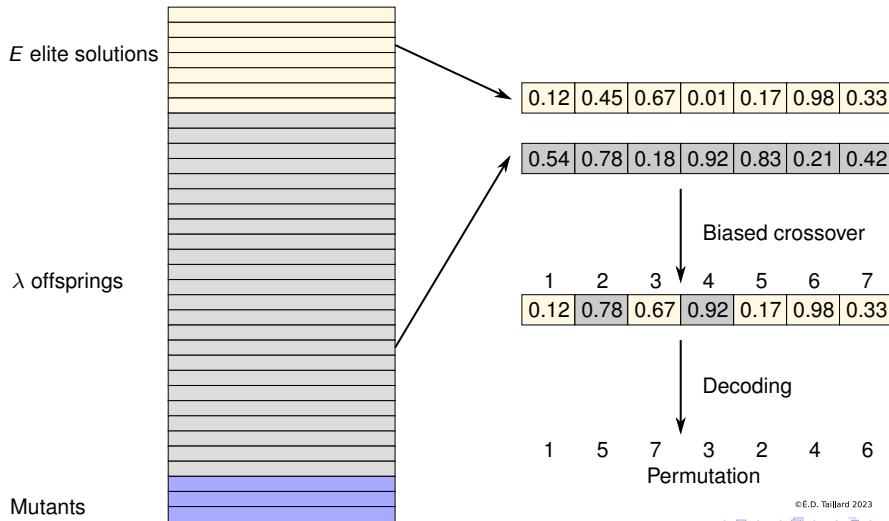
- A solution (permutation) is coded as an array of real numbers
 - Sorting the array allows the permutation to be reconstructed
- The E best solutions are kept for the next iteration
- The selection for reproduction always chooses a solution from the elite set E
- The child is generated with a uniform crossover, but with a probability $\neq 0.5$
 - The selected elite solution is favoured
- $\lambda < \mu - E$ children are generated
- $\mu - E - \lambda$ new arrays are generated randomly to maintain diversity

10.5 Biased Random Key Genetic Algorithm (BRKGA)

Working Principles

- A solution (permutation) is coded as an array of real numbers
 - Sorting the array allows the permutation to be reconstructed
- The E best solutions are kept for the next iteration
- The selection for reproduction always chooses a solution from the elite set E
- The child is generated with a uniform crossover, but with a probability $\neq 0.5$
 - The selected elite solution is favoured
- $\lambda < \mu - E$ children are generated
- $\mu - E - \lambda$ new arrays are generated randomly to maintain diversity

Biased Random Key Genetic Algorithm (BRKGA): Illustration



©É.D. Taillard 2023

10.6 Path Relinking

General Idea

- Gradually transform a starting solution into a given target solution
- Evaluate at each stage the neighbour solutions that allow to get closer to the target one
- Choose the neighbour with the best objective function
- The path can also be tried in the opposite direction
- Or modify alternately the 2 solutions and stop when they are identical

10.6 Path Relinking

General Idea

- Gradually transform a starting solution into a given target solution
- Evaluate at each stage the neighbour solutions that allow to get closer to the target one
- Choose the neighbour with the best objective function
- The path can also be tried in the opposite direction
- Or modify alternately the 2 solutions and stop when they are identical

10.6 Path Relinking

General Idea

- Gradually transform a starting solution into a given target solution
- Evaluate at each stage the neighbour solutions that allow to get closer to the target one
- Choose the neighbour with the best objective function
- The path can also be tried in the opposite direction
- Or modify alternately the 2 solutions and stop when they are identical

10.6 Path Relinking

General Idea

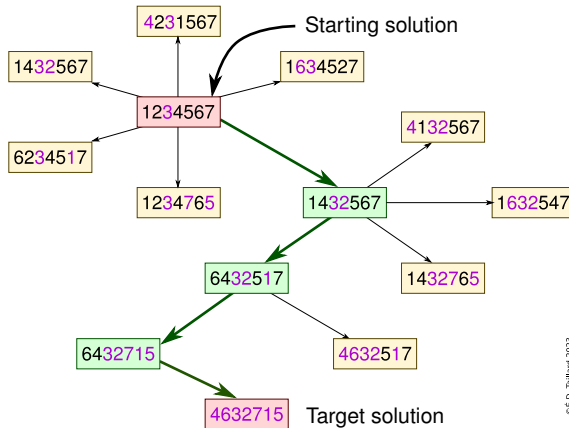
- Gradually transform a starting solution into a given target solution
- Evaluate at each stage the neighbour solutions that allow to get closer to the target one
- Choose the neighbour with the best objective function
- The path can also be tried in the opposite direction
- Or modify alternately the 2 solutions and stop when they are identical

10.6 Path Relinking

General Idea

- Gradually transform a starting solution into a given target solution
- Evaluate at each stage the neighbour solutions that allow to get closer to the target one
- Choose the neighbour with the best objective function
- The path can also be tried in the opposite direction
- Or modify alternately the 2 solutions and stop when they are identical

Illustration of a Path Relinking for Permutations



GRASP with Path Relinking

Input: GRASP procedure (with local search LS and parameter $0 \leq \alpha \leq 1$), parameters I_{max} and μ

Result: Population P of solutions

```

1  $P \leftarrow \emptyset$ 
2 while  $|P| < \mu$  do
3    $s \leftarrow \text{GRASP}(\alpha, \text{LS})$ 
4   if  $s \notin P$  then
5      $P \leftarrow P \cup s$ 
6 for  $I_{max}$  iterations do
7    $s \leftarrow \text{GRASP}(\alpha, \text{LS})$ 
8   Randomly draw  $s' \in P$  Apply a path relinking method between  $s$  and  $s'$ ; identifying the
     best solution  $s''$  of the path
9   if  $s'' \notin P$  and  $s''$  is strictly better than a solution of  $P$  then
10     $s''$  replaces the most different solution of  $P$  which is worse than  $s''$ 

```

10.8 Particle Swarm

General idea: Solution = particle

- The solutions are represented by real number vectors
- Each particle has initially a random position and speed
- Each particle is influenced by the best position it has found and the position of the best particle in the swarm

10.8 Particle Swarm

General idea: Solution = particle

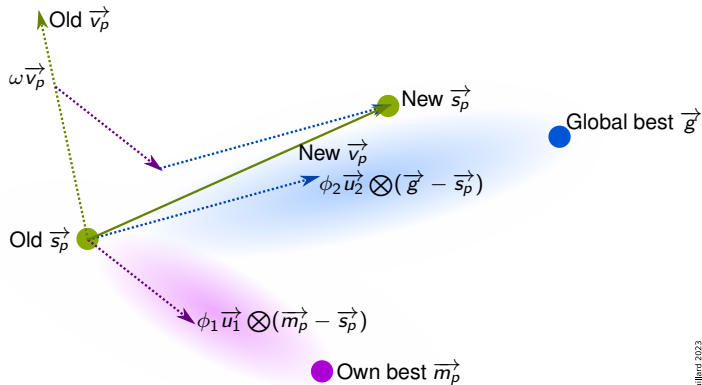
- The solutions are represented by real number vectors
- Each particle has initially a random position and speed
- Each particle is influenced by the best position it has found and the position of the best particle in the swarm

10.8 Particle Swarm

General idea: Solution = particle

- The solutions are represented by real number vectors
- Each particle has initially a random position and speed
- Each particle is influenced by the best position it has found and the position of the best particle in the swarm

Particle Swarm: Velocity and Position Update



©É.D. Taillard 2023

Particle Swarm Frame

Input: Function $f : [\vec{x}_{min}, \vec{x}_{max}] \in \mathbb{R}^n \rightarrow \mathbb{R}$ to minimize, parameters $\mu, \omega, \phi_1, \phi_2, l_{max}$

Result: \vec{g}

```

1   $f^* = \infty$ 
2  for  $p = 1 \dots \mu$  do
3       $\vec{v}_p \leftarrow \text{unif}(\vec{x}_{min} - \vec{x}_{max}, \vec{x}_{max} - \vec{x}_{min})$  // Initial particle velocity
4       $\vec{s}_p \leftarrow \text{unif}(\vec{x}_{min}, \vec{x}_{max})$  // Initial particle position (solution)
5       $\vec{m}_p \leftarrow \vec{s}_p$  // Best own position
6      if  $f^* > f(\vec{s}_p)$  then Update gloal best
7           $f^* \leftarrow f(\vec{s}_p)$ 
8           $\vec{g} \leftarrow \vec{s}_p$ 
9  for  $l_{max}$  iterations do
10     for  $p = 1 \dots \mu$  do
11          $\vec{u}_1 \leftarrow \text{unif}(\vec{0}, \vec{1})$ 
12          $\vec{u}_2 \leftarrow \text{unif}(\vec{0}, \vec{1})$ 
13          $\vec{v}_p \leftarrow \omega \vec{v}_p + \phi_1 \vec{u}_1 \otimes (\vec{m}_p - \vec{s}_p) + \phi_2 \vec{u}_2 \otimes (\vec{g} - \vec{s}_p)$  // Update velocity
14          $\vec{s}_p \leftarrow \vec{max}(\vec{min}(\vec{s}_p + \vec{v}_p, \vec{x}_{max}), \vec{x}_{min})$  // Update position
15         if  $f(\vec{m}_p) > f(\vec{s}_p)$  then Update own best
16              $\vec{m}_p \leftarrow \vec{s}_p$ 
17             if  $f^* > f(\vec{s}_p)$  then Update gloal best
18                  $f^* \leftarrow f(\vec{s}_p)$ 
19                  $\vec{g} \leftarrow \vec{s}_p$ 

```

Chapter 11

Heuristics Design

Chapter Content

| | | |
|----|---|-----|
| 11 | Heuristics Design | 245 |
| • | Problem Modelling | 247 |
| • | Model Choice | |
| • | Algorithmic Construction | 250 |
| • | Decomposition into a Series of Sub-problems | |
| • | Heuristics Tuning | 252 |

11.1 Problem Modelling

Choose the Right Model

Modeling has a preponderant influence on the difficulty of resolution

In some cases, one can go from a difficult problem to a simple one

Example: **de Bruijn** Graph for the reconstitution of a genetic sequence

- An edge link the detected k -nucleotides that can appear successively in the sequence
 - Hamiltonian path, NP-complete
- An edge represents a detected k -nucleotides linking two $k - 1$ -nucleotides
 - Eulerian path, can be solved in linear time

Choose the Right Model

Modeling has a preponderant influence on the difficulty of resolution

In some cases, one can go from a difficult problem to a simple one

Example: **de Bruijn** Graph for the reconstitution of a genetic sequence

- An edge link the detected k -nucleotides that can appear successively in the sequence
 - Hamiltonian path, NP-complete
- An edge represents a detected k -nucleotides linking two $k - 1$ -nucleotides
 - Eulerian path, can be solved in linear time

Choose the Right Model

Modeling has a preponderant influence on the difficulty of resolution

In some cases, one can go from a difficult problem to a simple one

Example: **de Bruijn** Graph for the reconstitution of a genetic sequence

- An edge link the detected k -nucleotides that can appear successively in the sequence
 - Hamiltonian path, NP-complete
- An edge represents a detected k -nucleotides linking two $k - 1$ -nucleotides
 - Eulerian path, can be solved in linear time

Choose the Right Model

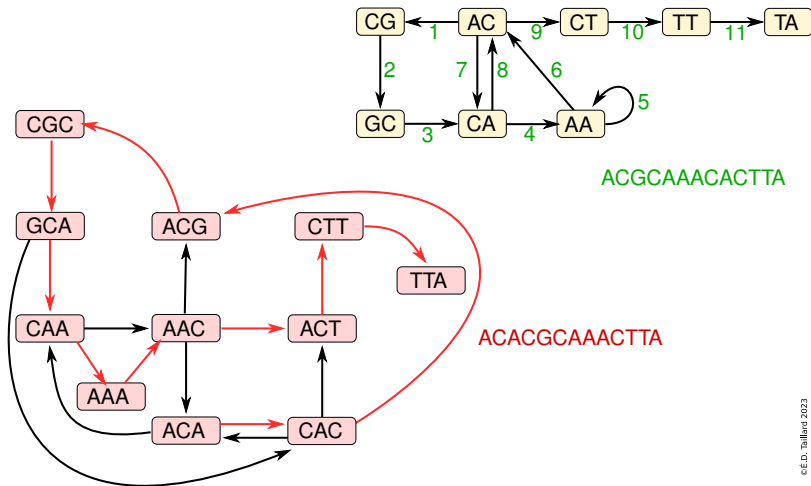
Modeling has a preponderant influence on the difficulty of resolution

In some cases, one can go from a difficult problem to a simple one

Example: **de Bruijn** Graph for the reconstitution of a genetic sequence

- An edge link the detected k -nucleotides that can appear successively in the sequence
 - Hamiltonian path, NP-complete
- An edge represents a detected k -nucleotides linking two $k - 1$ -nucleotides
 - Eulerian path, can be solved in linear time

Reconstitution of a Genetic Sequence



11.2 Algorithmic Construction

Using the divide-and-regard paradigm when dealing with large data

- Data partitioning
- Designing a specific local search
 - The choice of a good neighbourhood is crucial
 - The no free lunch theorem says that there is no universally better heuristic

11.2 Algorithmic Construction

Using the divide-and-regard paradigm when dealing with large data

- Data partitioning
- Designing a specific local search
 - The choice of a good neighbourhood is crucial
 - The **no free lunch** theorem says that there is no universally better heuristic
 - The heuristics must be adapted to the numerical characteristics of the problem

11.2 Algorithmic Construction

Using the divide-and-regard paradigm when dealing with large data

- Data partitioning
- Designing a specific local search
 - The choice of a good neighbourhood is crucial
 - The **no free lunch** theorem says that there is no universally better heuristic
 - The heuristics must be adapted to the numerical characteristics of the problem

11.2 Algorithmic Construction

Using the divide-and-regard paradigm when dealing with large data

- Data partitioning
- Designing a specific local search
 - The choice of a good neighbourhood is crucial
 - The **no free lunch** theorem says that there is no universally better heuristic
 - The heuristics must be adapted to the numerical characteristics of the problem

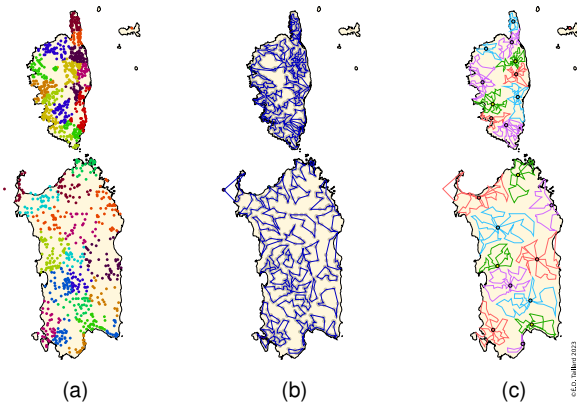
11.2 Algorithmic Construction

Using the divide-and-regard paradigm when dealing with large data

- Data partitioning
- Designing a specific local search
 - The choice of a good neighbourhood is crucial
 - The **no free lunch** theorem says that there is no universally better heuristic
 - The heuristics must be adapted to the numerical characteristics of the problem

Decomposition into a Series of Sub-problems: Location Routing

- (a) Create groups of cities with an unsupervised clustering method
- (b) Create a tour on each group
- (c) Position the depots with a clustering method (grouping of tours)



11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

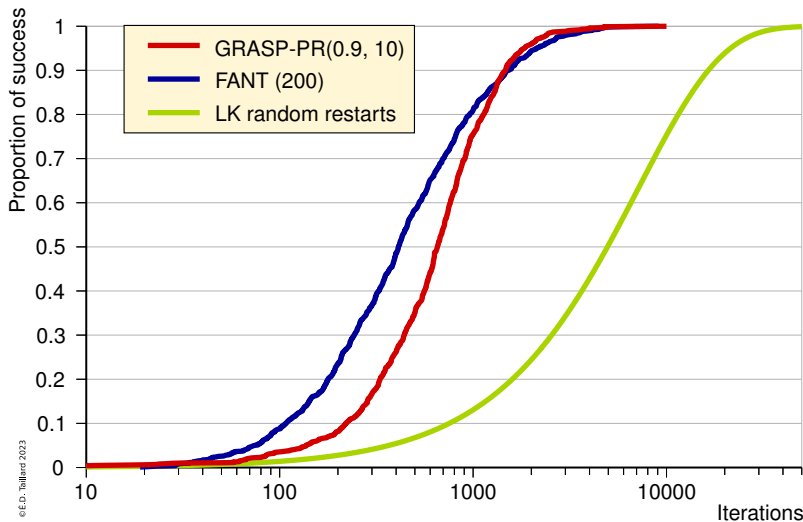
11.3 Heuristics Tuning

- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

11.3 Heuristics Tuning

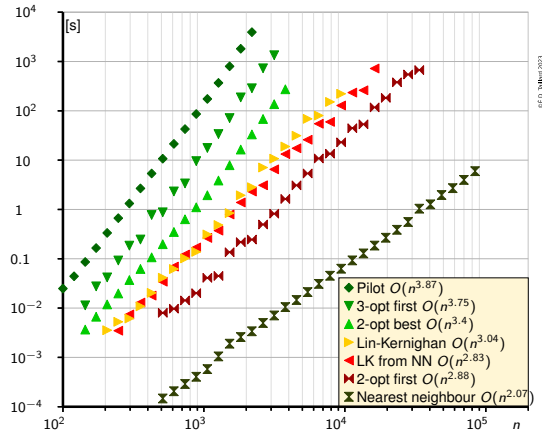
- Problem instance choice
 - Do not bias the results
- Graphical representation of the solutions produced
 - Allows a better understanding of the (non-)functioning of a heuristic
- Adjustment of settings and options
 - By far the most time consuming part
 - Consider using automated processes to avoid bias
- Measurement criteria
 - Success rate
 - Computational time
 - Quality of the solutions produced

Time to Target Plot, Optimal Resolution of a TSP



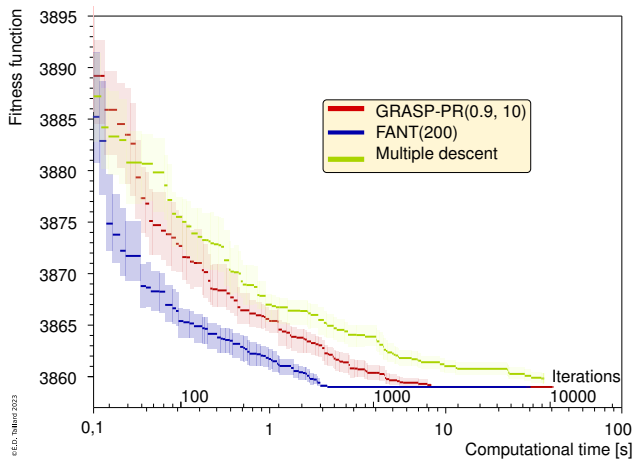
Empirical complexity analysis

Allows to see the evolution of the resources needed according to the size of the problem
Highlights algorithmic optimizations to be undertaken


















STAMP Graph








Estimates by bootstrapping the confidence interval of the observed means or medians according to the calculation effort
Double scale of effort measurement: absolute (number of GRASP iterations) or relative (calculation time)










©É.D. Taillard 2023

-  Adams J., Balas E., Zawack D.: The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science* **34**(4), 391–401 (1998). <https://doi.org/10.1287/mnsc.34.3.391>
-  Alvim A.C.F., Taillard É.D.: POPMUSIC for the World Location Routing Problem. *EURO Journal on Transportation and Logistics* **2**(3), 231–254 (2013). <https://doi.org/10.1007/s13676-013-0024-2>
-  Applegate D.L., Bixby R.E., Chvátal V., Cook W.J.: Concorde: A code for solving Traveling Salesman Problems. <https://github.com/matthelb/concorde> (1999). Accessed 16 June 2022
-  Battiti R., Tecchiolli G.: The Reactive Tabu Search, *ORSA Journal on Computing* **6**, 126–140 (1994). <https://doi.org/10.1287/ijoc.6.2.126>
-  Bradley E., Tibshirani R.J.: *An Introduction to the Bootstrap*, Chapman and Hall (1994)
-  Brélaz D.: New Methods to Color the Vertices of a Graph. *Communications of the ACM* **22**(4), 251–256 (1979). <https://doi.org/10.1145/359094.359101>
-  Černý V.: Thermodynamical Approach to the Traveling Salesman Problem: An efficient Simulation Algorithm. *Journal of Optimization Theory and Applications* **45**(1), 41–51 (1985). <https://doi.org/10.1007/BF00940812>
-  Charon I., Hudry O.: The Noising Method: a New Method for Combinatorial Optimization. *Operations Research Letters* **14**(3), 133–137 (1993). [https://doi.org/10.1016/0167-6377\(93\)90023-A](https://doi.org/10.1016/0167-6377(93)90023-A)

-  Colorni A., Dorigo M., Maniezzo V.: Distributed Optimization by Ant Colonies. In: Actes de la première conférence européenne sur la vie artificielle, pp. 134–142, Paris. Elsevier, (1991)
-  Cook S.A.: The Complexity of Theorem-Proving Procedures. In: Proceedings of the third annual ACM symposium on Theory of computing 151–158. ACM (1971). <https://doi.org/10.1145/800157.805047>
-  Croes G.A.: A Method for Solving Traveling Salesman Problems. Operations Research **6**, 791–812 (1958). <https://doi.org/10.1287/opre.6.6.791>
-  Dakin R.J.: A Tree Search Algorithm for Mixed Integer Programming Problems. The Computer Journal **8**(3), 250–255 (1965). <https://doi.org/10.1093/comjnl/8.3.250>
-  Deneubourg J., Goss S., Pasteels J., Fresneau D., Lachaud J.: Self-Organization Mechanisms in Ant Societies (II) : Learning in Foraging and Division of Labor. In: Pasteels J., et al. (eds.) From Individual to Collective Behavior in Social Insects, Experientia supplementum **54**, 177–196. Birkhäuser, Basel (1987)
-  Dorigo M., Gambardella L.M.: Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation **1**(1), 53–66 (1997). <https://doi.org/10.1109/4235.585892>
-  Dueck G.: New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel. Journal of Computational Physics **104**(1), 86–92 (1993). <https://doi.org/10.1006/jcph.1993.1010>

-  Dueck G., Scheuer T.: Threshold accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *Journal of Computational Physics* **90**(1), 161–175(1990).
[https://doi.org/10.1016/0021-9991\(90\)90201-B](https://doi.org/10.1016/0021-9991(90)90201-B)
-  Duin C., Voß S.: The Pilot Method: A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs. *Networks* **34**, 181-191 (1999).
[https://doi.org/10.1002/\(SICI\)1097-0037\(199910\)34:3%3C181::AID-NET2%3E3.0.CO;2-Y](https://doi.org/10.1002/(SICI)1097-0037(199910)34:3%3C181::AID-NET2%3E3.0.CO;2-Y)
-  Feo T.A., Resende M.G.C.: Greedy Randomized Adaptive Search Procedure. *Journal of Global Optimization* **6**, 109–133 (1995). <https://doi.org/10.1007/BF01096763>
-  Furcy D., Koenig S.: Limited Discrepancy Beam Search. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 125–131 (2005)
-  Garey M.R., Johnson D.S.: *Computers and Intractability — A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co. (1979)
-  Gilli M., Küllezi E., Hysi H.: A Data-Driven Optimization Heuristic for Downside Risk Minimization. *Journal of Risk* **8**(3), 1–19 (2006)
-  Glover F.: Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences* **8**(1), 156–166 (1977). <https://doi.org/10.1111/j.1540-5915.1977.tb01074.x>

-  Glover F.: Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research* **13**(5), 533–549 (1986). [https://doi.org/10.1016/0305-0548\(86\)90048-1](https://doi.org/10.1016/0305-0548(86)90048-1)
-  Glover F.: Tabu Search — part I. *ORSA Journal on Computing* **1**(3), 190–206 (1989). <https://doi.org/10.1287/ijoc.1.3.190>
-  Glover F.: Tabu Search — part II. *ORSA Journal on Computing* **2**(1), 4–32 (1990). <https://doi.org/10.1287/ijoc.2.1.4>
-  Glover F.: Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems. *Discrete Applied Mathematics* **65**, 223–253 (1996). [https://doi.org/10.1016/0166-218X\(94\)00037-E](https://doi.org/10.1016/0166-218X(94)00037-E)
-  Glover F.: Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges. In: Barr R.S., Helgason R.V., Kennington J.L. (eds.) *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*. pp. 1–75. Springer, Boston (1997). https://doi.org/10.1007/978-1-4615-4102-8_1
-  Glover F.: A Template for Scatter Search and Path Relinking. In: Hao J.K., Lutton E., Ronald E., Schoenauer M., Snyers D. (eds.) *Artificial Evolution, Lecture Notes in Computer Science* **1363**, 13–54 (1998). <https://doi.org/10.1007/BFb0026589>
-  Glover F., Laguna M.: *Tabu Search*. Kluwer, Dordrecht (1997)



Greistorfer P., Staněk R., Maniezzo V.: The Magnifying Glass Heuristic for the Generalized Quadratic Assignment Problem. Metaheuristic International Conference (MIC'19) proceedings, Cartagena, Columbia (2019)



Helsgaun K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal of Operational Research **126**(1), 106–130 (2000).
[https://doi.org/10.1016/S0377-2217\(99\)00284-2](https://doi.org/10.1016/S0377-2217(99)00284-2)



Helsgaun K.: Using POPMUSIC for Candidate Set Generation in the Lin-Kernighan-Helsgaun TSP Solver. Department of Computer Science, Roskilde University, Denmark (2018)



Holland J.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Harbor (1975)



Jarník V.: O jistém problému minimálním. (Z dopisu panu O. Borůvkovi [On a certain problem of minimization (from a letter to O. Borůvka)], in Czech. Práce moravské přírodovědecké společnosti **6**(4), 57–63 (1930). <http://dml.cz/dmlcz/500726>



Jarník V.: O minimálních grafech, obsahujících n daných bodů [On minimal graphs containing n given points], in Czech. Časopis pro Pěstování Matematiky a Fysiky **63**(8) 223–235 (1934).
<https://doi.org/10.21136/CPMF.1934.122548>



Jovanovic R., Tuba M., Voß S.: Fixed Set Search Applied to the Traveling Salesman Problem. In: Blum C., Gambini Santos H., Pinacho-Davidson P., Godoy del Campo J. (eds.) Hybrid Metaheuristics. Lecture Notes in Computer Science, vol. 11299, pp. 63–77 Springer, Cham (2019).
https://doi.org/10.1007/978-3-030-05983-5_5



Kaufman L., Rousseeuw P.J.: Clustering by means of Medoids. In: Dodge Y. (ed.) Statistical Data Analysis Based on the L_1 -Norm and Related Methods, pp. 405–416 North-Holland, Amsterdam (1987)



Kennedy J., Eberhart R.C.: Particle Swarm Optimization. IEEE International Conference on Neural Networks. vol. IV, pp. 1942–1948 Piscataway, NJ. IEEE Service Center, Perth (1995).
<https://doi.org/10.1109/ICNN.1995.488968>



Kirkpatrick S., Gelatt C.D., Vecchi M.P.: Optimization by Simulated Annealing. Science **220** (4598), 671–680 (1983). <https://doi.org/10.1126/science.220.4598.671>



Laguna M., Martí R.: GRASP and Path Relinking for 2-layer Straight Line Crossing Minimization. INFORMS Journal on Computing **11**(1), 44–52 (1999). <https://doi.org/10.1287/ijoc.11.1.44>



L'Ecuyer P.: Combined Multiple Recursive Random Number Generators. Operations Research **44**(5), 816–822 (1996). <http://www.jstor.org/stable/171570>



Lin S., Kernighan B. W.: An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research **21**(2), 498–516 (1973). <https://www.jstor.org/stable/169020>

-  Lones M.: Mitigating Metaphors: A Comprehensible Guide to Recent Nature-Inspired Algorithms. SN Computer Science **1**(49), (2020). <https://doi.org/10.1007/s42979-019-0050-8>
-  Lowerre B.: The Harpy Speech Recognition System. Ph. D. Thesis, Carnegie Mellon University (1976)
-  López-Ibáñez M., Dubois-Lacoste J., Pérez Cáceres L., Birattari M., Stützle T.: The Irace Package: Iterated Racing for Automatic Algorithm Configuration. Operations Research Perspectives **3**, 43–58 (2016). <https://doi.org/10.1016/j.orp.2016.09.002>
-  Martello S., Toth P.: Knapsack Problems — Algorithms and Computer Implementations. Wiley, Chichester (1990)
-  Hansen P., Mladenović N.: An Introduction to Variable Neighborhood Search. In: Voß S. , Martello S., Osman I.H., Roucairol C. (eds.) Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization, pp. 422–458, Kluwer, Dordrecht (1999). https://doi.org/10.1007/978-1-4615-5775-3_30
-  Moscato P: Memetic Algorithms: A Short Introduction. In: Corne D., Glover F., Dorigo M. (eds.) New Ideas in Optimisation, pp. 219–235, McGraw-Hill, London (1999)
-  Or I.: Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking. Ph. D. Thesis, Northwestern University (1976)



Osterman C., Rego C.: A k-level Data Structure for Large-scale Traveling Salesman Problems. *Annals of Operations Research* **244**(2), 1–19 (2016). <https://doi.org/10.1007/s10479-016-2159-7>



Paquete L., Chiarandini M., Stützle T.: Pareto Local Optimum Sets in the Biobjective Traveling Salesman Problem: An Experimental Study. In: Gandibleux, X., Sevaux, M., Sörensen, K., T'kindt, V. (eds.) *Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems*, vol. 535, pp. 177–200 Springer (2004). https://doi.org/10.1007/978-3-642-17144-4_7



Rechenberg I.: *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart (1973). <https://doi.org/10.1002/fedr.19750860506>










Reinelt G.: TSPLIB — A T.S.P. Library. (1990). <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95>. Accessed 16 June 2022











Resende M.G.C., Riberio C.C.: *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer, New-York, (2016). <https://doi.org/10.1007/978-1-4939-6530-4>



Sahling F., Buschkühl L., Tempelmeier H., Helber S.: Solving a Multi-level Capacitated Lot Sizing Problem with Multi-period Setup Carry-over via a Fix-and-Optimize Heuristic. *Computers and Operations Research* **36**(9), 2546–2553 (2009). <https://doi.org/10.1016/j.cor.2008.10.009>

-  Shaw P.: Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. 4. International Conference of Principles and Practice of Constraint Programming, pp. 417–431. Springer-Verlag (1998). https://doi.org/10.1007/3-540-49481-2_30
-  Sniedovich M., Voß S.: The Corridor Method: a Dynamic Programming Inspired Metaheuristic. Control and Cybernetics **35**(3), 551–578 (2006). <http://eudml.org/doc/209435>
-  Sörensen K., Seveaux M.: MA—PM: Memetic Algorithms with Population Management. Computers & Operations Research **33**, 1214–1225 (2006). <https://doi.org/10.1016/j.cor.2004.09.011>
-  Stützle T., Hoos H.H.: MAX MIN Ant System. Future Generation Computer Systems **16**(8), 889–914 (2000). [https://doi.org/10.1016/S0167-739X\(00\)00043-1](https://doi.org/10.1016/S0167-739X(00)00043-1)
-  Taillard É.D.: Parallel Iterative Search Methods for Vehicle Routing Problems. Networks **23**(8), 661–673 (1993). <https://doi.org/10.1002/net.3230230804>
-  Taillard É.D.: Comparison of Iterative Searches for the Quadratic Assignment Problem. Location Science **3**(2), 87–105 (1995). [https://doi.org/10.1016/0966-8349\(95\)00008-6](https://doi.org/10.1016/0966-8349(95)00008-6)
-  Taillard É.D.: La programmation à mémoire adaptative et les algorithmes pseudo-gloutons: nouvelles perspectives pour les méta-heuristiques. HDR Thesis, Université de Versailles-Saint-Quentin-en-Yvelines (1998)

-  Taillard É.D.: Heuristic Methods for Large Centroid Clustering Problems. J. Heuristics **9**(1), 51–73 (2003). <https://doi.org/10.1023/A:1021841728075>
-  Taillard É.D.: Tutorial : Few guidelines for analyzing methods. Metaheuristic International Conference (MIC'05) proceedings, Wien, Austria (2005)
-  Taillard É.D.: A Linearithmic Heuristic for the Travelling Salesman Problem. European Journal of Operational Research **297**(2), 442–450 (2022). <https://doi.org/10.1016/j.ejor.2021.05.034>
-  Taillard É.D., Waelti P., Zuber J.: Few Statistical Tests for Proportions Comparison. European Journal of Operational Research **185**(3), 1336–1350 (2008). <https://doi.org/10.1016/j.ejor.2006.03.070>
-  Toth P., Vigo D.: The Granular Tabu Search and Its Application to the Vehicle-Routing Problem. INFORMS J. on Computing **15**(4), 333–346 (2003). <https://doi.org/10.1287/ijoc.15.4.333.24890>
-  Voigt(ed.) Der Handlungsreisende wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein ; Mit einem Titelkupf. Voigt, Ilmenau (1832)
-  Wolpert D.H., Macready W.G.: No Free Lunch Theorems for Optimization. IEEE Transaction on Evolutionary computation **1**(1), 67–82 (1997). <https://doi.org/10.1109/4235.585893>
-  Xavier A.E., Xavier V.L.: Flying Elephants: a General Method for Solving non-differentiable Problems. Journal of Heuristics **22**(4), 649–664 (2016). <https://doi.org/10.1007/s10732-014-9268-8>