Metaheuristics (1/2)

Outline

- Metaheuristics
- The problems and the assumptions
  - The neighbourhoods
  - Random numbers
  - The instances
- How to escape from a local optimum
- Simulated annealing
  - The cooling schedule and the parameters
  - Examples
  - Infeasible solutions: simulated annealing with a penalty
  - An experiment on the medium TSP instance
  - Properties
- Tabu search
  - Example
  - Tabu search for the TSP, and the effect of the tenure
- Iterated local search
  - Example

Learning objectives

At the end of this chapter you will be able to:

- execute by hand simulated annealing, tabu search, iterated local search and variable neighbourhood search, with given random numbers;
- execute GRASP, large neighbourhood search and a genetic or memetic algorithm on small instances;
- explain how each method escapes from a local optimum, and which parameters tune it;
- compare the methods on the same instances, over several seeds;
- say what metaheuristics do not give: a dual bound.

Prerequisites

- From this module, the chapter «Local search»: neighbourhoods, first and best improvement, variable neighbourhood descent and multi-start.
- From this module, the chapters «Constructive heuristics» (randomized construction) and «Heuristic algorithms: definitions» (intensification and diversification, experimental evaluation).

This first part covers the setting of the chapter and the first three metaheuristics:
simulated annealing, tabu search and iterated local search.

Metaheuristics

Introduction: Metaheuristics

- A local search stops at the first local optimum it meets, which can be far from the optimum
- In this section we define what a **metaheuristic** is: a strategy that drives the local search beyond that first local optimum
- All the methods of the chapter (simulated annealing, tabu search, iterated local search, VNS, GRASP, LNS, genetic and memetic algorithms) are metaheuristics

What a metaheuristic is

Definition 1: Metaheuristic

*A **metaheuristic** is a general strategy that guides a local search beyond the first local optimum, to explore other regions of the set of feasible solutions.*

- A metaheuristic is a *scheme*, not an algorithm: an algorithm is obtained by choosing the representation, the neighbourhoods, the local search and the parameters.
- Every metaheuristic balances **intensification** and **diversification**, defined in the chapter «Heuristic algorithms: definitions».

Three ways to escape

A local search stops in the first local optimum;

a metaheuristic escapes from it

- by **accepting worse solutions**,
- by **jumping** to a new starting solution,
- or by **combining** several good solutions.

Summary

Metaheuristic:

- a general strategy that guides a local search beyond the first local optimum
- a scheme, not an algorithm: representation, neighbourhoods, local search and parameters must be chosen

How it escapes:

- accepting worse solutions, jumping to a new start, or combining good solutions
- always balancing intensification and diversification

The problems and the assumptions

Introduction: The problems and the assumptions

- We fix the setting of the whole chapter: the four problems, their neighbourhoods, the random numbers and the instances
- Every example of the chapter can be checked by hand, because the random numbers are given and the instances are fixed
- The medium instances, with known optimal values, are used to measure how good the metaheuristics are

The four problems

- The metaheuristics are illustrated on the four problems of the chapter «Heuristic algorithms: definitions»:
  - the 0/1 knapsack problem (KP),
  - the bin packing problem (BPP),
  - the travelling salesman problem (TSP),
  - $P \,||\, C_{\max}$.
- Their definitions and assumptions are on their pages; we recall the notation.

The notation of the four problems

problem	data	solution	objective
KP	items $\mathscr{I}$, profits p_j, weights w_j, capacity c	subset of items	max total profit
BPP	items $\mathscr{I}$, weights w_j, capacity c	patterns P_b	min number of bins
TSP	cities $\mathscr{V}$, distances d_{ij}	tour σ	min length $L(\sigma)$
$P \mid\mid C_{\max}$	jobs $\mathscr{J}$, times t_j, machines m	sets $J_1, \ldots, J_m$	min makespan

Assumptions and notation

- Every instance satisfies the assumptions of its problem:
  - $w_j \leq c$ for all items and $\sum_j w_j > c$ for the KP and the BPP;
  - $n \geq 3$ and symmetric distances for the TSP;
  - $n > m$ for $P \,||\, C_{\max}$.
- For $P \,||\, C_{\max}$, $L_h = \sum_{j \in J_h} t_j$ is the **load** of machine h and $C_{\max} = \max_h L_h$.
- As in the chapter «Heuristic algorithms: definitions», $\mathscr{A}(I)$ is the value of the solution computed by the algorithm $\mathscr{A}$ on the instance I, and $z(I)$ is the optimal value.
- We write $f(S)$ for the value of a solution S, and we describe the algorithms for a **minimization** problem; for the KP, a maximization problem, the inequalities are reversed.

The neighbourhoods (1/2)

- All the metaheuristics of this chapter are built on the neighbourhoods of the chapter «Local search». We recall them by name, with the moves that define them; n is the number of items, cities or jobs.

problem	name	move	size
KP	flip	add one item to the knapsack, or remove one	n
	swap	remove one selected item and add one unselected item	$\leq \frac{n^2}{4}$
BPP	move	move one item to another bin	$\leq n \cdot nb$
	swap	exchange two items packed in different bins	$\leq \frac{n(n-1)}{2}$

The neighbourhoods (2/2)

problem	name	move	size
TSP	2-opt	remove two non-adjacent edges of the tour and reconnect it with the two other edges, reversing the segment between them	$\frac{n(n-3)}{2}$
	Or-opt	move a segment of 1, 2 or 3 consecutive cities to another position	$O(n^2)$
$P \mid\mid C_{\max}$	move	move one job to another machine	$n(m-1)$
	swap	exchange two jobs assigned to different machines	$\leq \frac{n(n-1)}{2}$

The 2-opt move

- A TSP tour is stored as a permutation $\sigma = (\sigma_1, \ldots, \sigma_n)$ with $\sigma_1 = 1$.
- The **2-opt move** (i, j), with $1 \leq i$, $i + 2 \leq j \leq n$ and $(i, j) \neq (1, n)$:
  - removes the edges (σ_i, σ_{i+1}) and (σ_j, σ_{j+1}) (with $\sigma_{n+1} = \sigma_1$),
  - adds the edges (σ_i, σ_j) and $(\sigma_{i+1}, \sigma_{j+1})$,
  - reverses the segment $\sigma_{i+1}, \ldots, \sigma_j$.
- Its effect on the length is computed in $O(1)$:

$$\delta(i, j) = d_{\sigma_i \sigma_j} + d_{\sigma_{i+1} \sigma_{j+1}} - d_{\sigma_i \sigma_{i+1}} - d_{\sigma_j \sigma_{j+1}}.$$

Flat objectives and auxiliary objectives

- The objectives of the BPP and of $P \,||\, C_{\max}$ are **flat**: most moves do not change the number of bins, or the makespan.
- Two solutions with the same value are then compared with an **auxiliary objective**:
  - BPP: the sum of the squared loads $\sum_b \left(\sum_{j \in P_b} w_j \right)^2$, to be *maximized*: it grows when the items concentrate in fewer, fuller bins;
  - $P \,||\, C_{\max}$: the sum of the squared loads $\sum_h L_h^2$, to be *minimized*: it decreases when the loads become balanced.

Lexicographic comparison for $P \,||\, C_{\max}$

- For $P \,||\, C_{\max}$ we compare two solutions **lexicographically**:
  - first by $C_{\max}$,
  - and if the makespans are equal, by $\sum_h L_h^2$.

- For move and swap this selects the same improving moves as the auxiliary objective $\sum_h L_h^2$ of the chapter «Local search»:
  - a transfer that decreases $\sum_h L_h^2$ never increases $C_{\max}$,
  - and a transfer that decreases $C_{\max}$ also decreases $\sum_h L_h^2$.

Infeasible solutions and the local searches

- For the KP the flip neighbourhood contains infeasible solutions (adding an item may exceed the capacity). A local search either discards them or accepts them with a **penalty** on the excess weight; both options appear in this chapter.

- The **local searches** used inside the metaheuristics are the descents of the chapter «Local search»:
  - TwoOptLS (2-opt, best improvement) for the TSP,
  - MoveDescent (move, best improvement, lexicographic comparison) for $P \,||\, C_{\max}$,
  - KPLocalSearch (feasible flips and swaps, best improvement) for the KP.

- In every best-improvement descent, ties between equally good moves are broken in favour of the first move found, scanning the indices in increasing order.

Random numbers

- Most metaheuristics make **random choices**. To make every example and every exercise reproducible, all the random numbers of this chapter, and of its C++ code, are produced by the **minimal standard generator** of Park and Miller (1988):

$$seed \leftarrow 16807 \cdot seed \bmod (2^{31}-1), \qquad u = \frac{seed}{2^{31}-1} \in (0,1).$$

- A call $\textsc{Rand}()$ updates *seed* and returns u.
- A random integer in $\{0, 1, \ldots, K-1\}$ is $\lfloor u \cdot K \rfloor$, and a random element of a list of K elements is the element in position $1 + \lfloor u \cdot K \rfloor$.

The seed

- The initial value of *seed*, the **seed**, is an integer in $\{1, \ldots, 2^{31} - 2\}$.
- The same seed always gives the same sequence of numbers; different seeds give different runs of the same algorithm.
- With $seed = 1$ the first numbers are, rounded to four decimals,

$$0.0000,\ 0.1315,\ 0.7556,\ 0.4587,$$
$$0.5328,\ 0.2190,\ 0.0470,\ 0.6789,\ \ldots$$

  The first number produced from a small seed is always very close to 0; it is a property of the generator, not an error.
- In the worked examples the random numbers are written in the text, rounded to four decimals, so that every step can be checked by hand; the rounding never changes a decision.

The reference instances

- The worked examples use the four **reference instances** of the chapter «Heuristic algorithms: definitions», whose optimal values are known:

problem	reference instance	optimal value and solution
KP	$p = (10, 7, 6, 4)$, $w = (5, 4, 3, 3)$, $c = 9$	$z(I) = 17$, items $\{1, 2\}$
BPP	$w = (4, 4, 5, 3, 2, 3)$, $c = 7$	$z(I) = 3$, patterns $\{1, 4\}, \{2, 6\}, \{3, 5\}$
TSP	5 cities, distances d_{ij} on the next slide	$z(I) = 18$, tour $(1, 3, 5, 2, 4)$
$P \mid\mid C_{\max}$	$t = (5, 5, 4, 4, 3, 3, 3)$, $m = 3$	$z(I) = 9$, sets $\{1, 3\}, \{2, 4\}, \{5, 6, 7\}$

The TSP reference instance

The distances d_{ij} of the 5 cities of the TSP reference instance:

d_{ij}	1	2	3	4	5
1	—	5	2	2	9
2	5	—	4	3	4
3	2	4	—	4	7
4	2	3	4	—	7
5	9	4	7	7	—

The optimal tour is $(1, 3, 5, 2, 4)$, of length $z(I) = 18$.

The medium instances

- The reference instances are small enough to follow every iteration by hand, but too small to show the behaviour of a metaheuristic over thousands of iterations.
- For the experiments we use one **medium instance** per problem, generated once with the random number generator above and fixed: TSP20, KP20, PCM15 and BPP24.
- Their optimal values have been computed with an exact algorithm, and are used only to measure the quality of the heuristic solutions.

Example: the medium TSP instance TSP20 (1/2)

- 20 cities in the plane, with integer coordinates (x_i, y_i), and distances

$$d_{ij} = \left\lfloor \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + \frac{1}{2} \right\rfloor$$

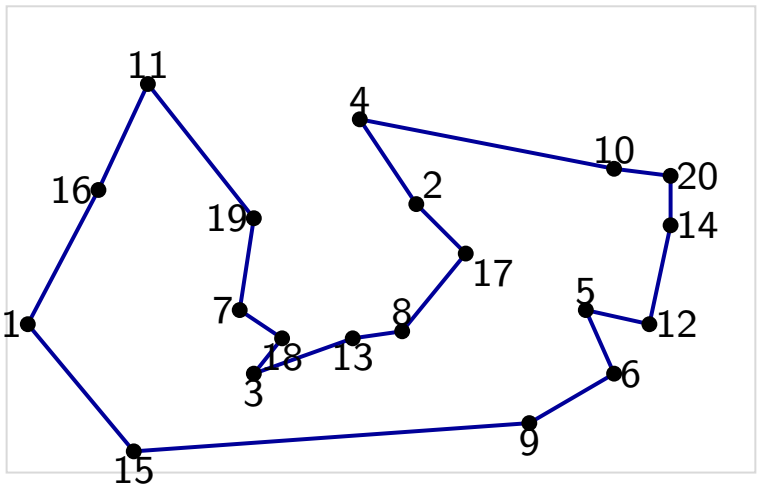
(the Euclidean distance rounded to the nearest integer):

i	1	2	3	4	5	6	7	8	9	10
x_i	0	55	32	47	79	83	30	53	71	83
y_i	18	35	11	47	20	11	20	17	4	40

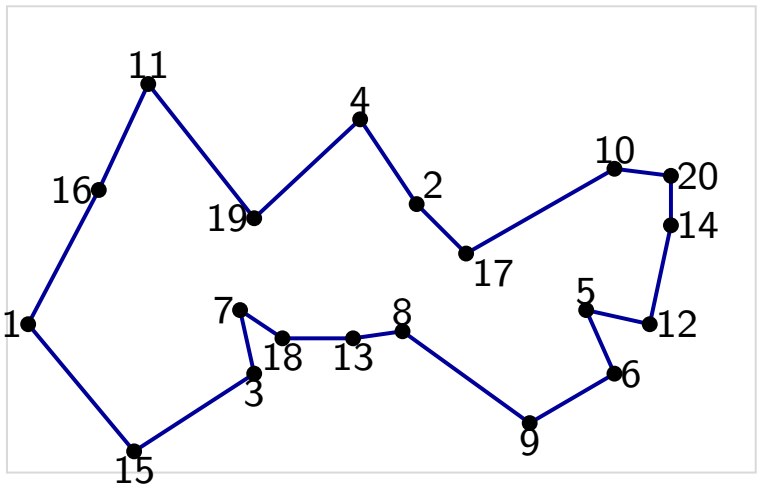
i	11	12	13	14	15	16	17	18	19	20
x_i	17	88	46	91	15	10	62	36	32	91
y_i	52	18	16	32	0	37	28	16	33	39

Example: the medium TSP instance TSP20 (2/2)

nearest neighbour from city 1: $L(\sigma) = 326$



an optimal tour: $L(\sigma) = 291$



- The nearest neighbour tour from city 1 has length 326; the 2-opt local search started from it stops at a local optimum of length 319, after two improving moves.
- The optimal value is $z(I) = 291$: the local optimum is 9.6% above it.

Example: the medium instances of the KP, the BPP and $P \,||\, C_{\max}$ (1/2)

KP20: $n = 20$ items, capacity $c = 389$ (half of the total weight, rounded down),

j	1	2	3	4	5	6	7	8	9	10
p_j	54	55	55	52	58	42	53	53	22	64
w_j	39	43	49	47	44	35	43	44	16	50

j	11	12	13	14	15	16	17	18	19	20
p_j	54	51	39	49	53	46	28	64	58	34
w_j	40	42	28	38	46	39	14	50	45	27

- The greedy algorithm by efficiency finds profit 490, and the local search with flips and swaps started from it reaches 514; the optimal value is $z(I) = 521$.

Example: the medium instances of the KP, the BPP and $P \,||\, C_{\max}$ (2/2)

PCM15: $m = 4$ machines and $n = 15$ jobs with processing times

$$t = (10, 23, 26, 16, 23, 11, 14, 18, 18, 24, 29, 22, 13, 13, 15).$$

The total time is 275, so $z(I) \geq \lceil 275/4 \rceil = 69$, and an assignment of makespan 69 exists: $z(I) = 69$. LPT finds makespan 73.

BPP24: bins of capacity $c = 100$ and $n = 24$ items with weights

$$w = (30, 40, 57, 22, 37, 23, 34, 34, 55, 51, 21, 47,$$
$$22, 24, 22, 34, 20, 51, 25, 54, 35, 27, 55, 48).$$

The total weight is 868, so at least $\lceil 868/100 \rceil = 9$ bins are needed, and a packing into 9 bins exists: $z(I) = 9$. FF and FFD both use 10 bins.

Summary (1/2)

The four problems:

- KP, BPP, TSP and $P \,||\, C_{\max}$, with the notation of their pages
- algorithms described for minimization; for the KP the inequalities are reversed

Neighbourhoods and local searches:

- flip and swap (KP), move and swap (BPP, $P \,||\, C_{\max}$), 2-opt and Or-opt (TSP)
- flat objectives: auxiliary objective, lexicographic comparison for $P \,||\, C_{\max}$
- TwoOptLS, MoveDescent, KPLocalSearch: best improvement, first move on ties

Summary (2/2)

Random numbers:

- minimal standard generator of Park and Miller: the same seed gives the same run
- in the examples the numbers are written in the text, rounded to four decimals

Instances:

- four reference instances, small enough to follow by hand
- four medium instances, TSP20, KP20, PCM15 and BPP24, with optimal values 291, 521, 69 and 9

How to escape from a local optimum

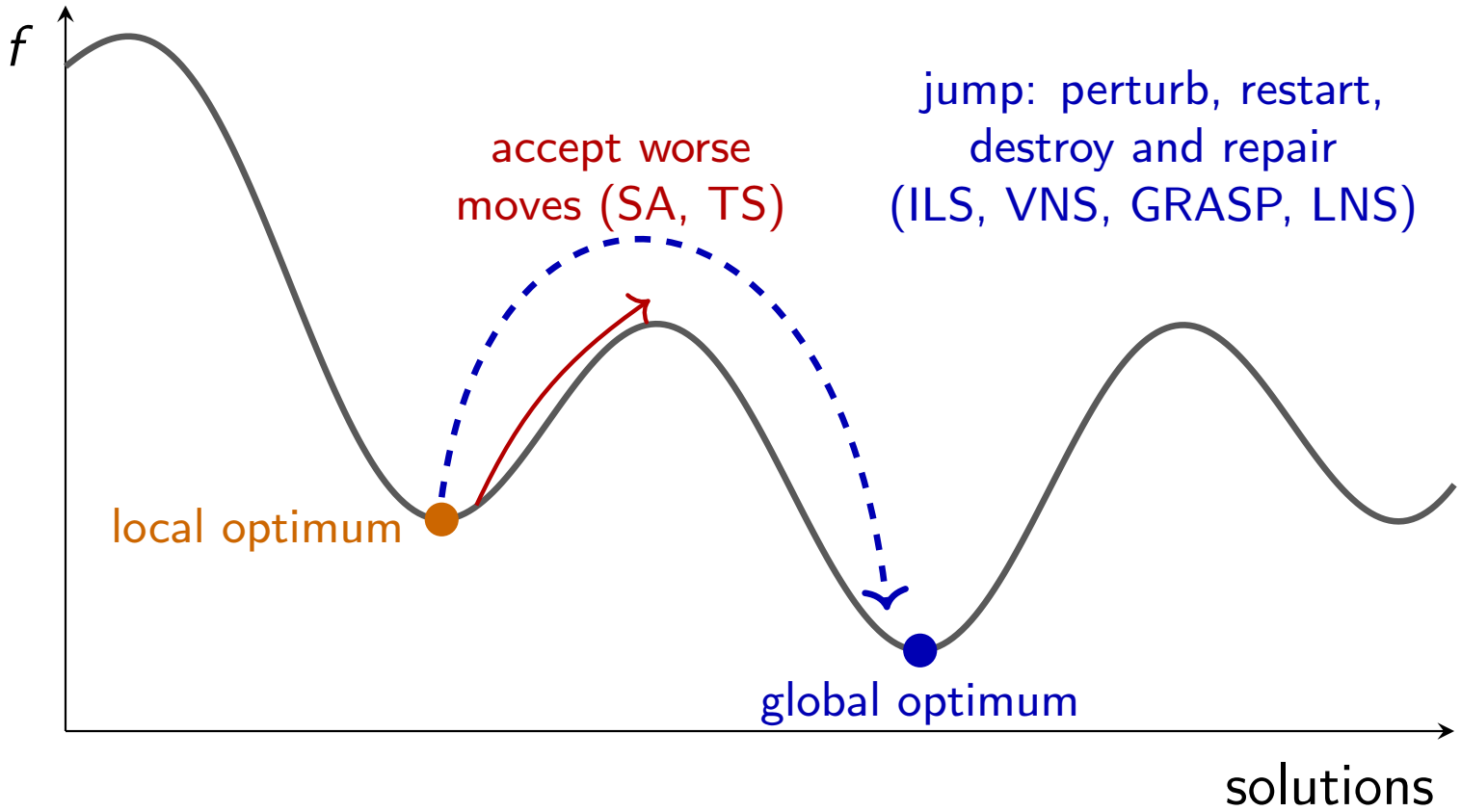
Introduction: How to escape from a local optimum

- A local search gets stuck at the bottom of the valley where it starts
- In this section we list the ways to escape from a local optimum, and the metaheuristic that uses each of them
- The list is the map of the chapter: each of the following sections presents one way

Local search and multi-start

- A local search moves from a solution to a better neighbour, and stops at the first **local optimum**: a solution with no better neighbour.
- In the landscape of the objective function, it descends to the bottom of the valley where it starts, and stays there. This is pure **intensification**.
- Repeating the local search from many starting solutions (multi-start) explores many valleys, but
  - each start ignores everything learnt before,
  - and good valleys are usually found close to other good valleys.

The landscape of the objective function



The ways to escape (1/3)

how to escape	idea	metaheuristic
accept worse moves *at random*	a worse neighbour is accepted with a probability that decreases with the worsening and with time	simulated annealing
accept worse moves *deterministically, with memory*	move to the best neighbour even if it is worse, and forbid going back to the solutions just visited	tabu search

The ways to escape (2/3)

how to escape	idea	metaheuristic
perturb the local optimum and restart	apply a random modification that the local search cannot undo, and descend again	iterated local search
change the neighbourhood	when the search is stuck, use a larger neighbourhood, of which the local optimum is no longer a local optimum	variable neighbourhood search

The ways to escape (3/3)

how to escape	idea	metaheuristic
restart from a randomized construction	build a different good solution at every restart, and improve it	GRASP
destroy and repair	remove a large part of the solution and rebuild it with a heuristic or an exact algorithm	large neighbourhood search
work with a *population*	combine the parts of several good solutions into new ones	genetic and memetic algorithms

Walk, jump, or keep many solutions

- The first two ways keep **one current solution** and change the **acceptance rule** of the local search: they walk out of the valley through worse solutions.
- The next four **jump**: they generate a new starting point, far enough to leave the valley but close enough to keep what is good.
- The last one keeps **many solutions** at the same time.
- Every metaheuristic must balance **intensification** (descend to the bottom of the current valley) and **diversification** (leave it for another one). For each method we point out which component does what.

A scheme, not an algorithm

- A metaheuristic is a *scheme*, not an algorithm: to obtain an algorithm one must choose
  - the representation,
  - the neighbourhoods,
  - the local search,
  - and the parameters,

  and these choices matter at least as much as the scheme.

- We present every metaheuristic on the problems where its idea is most natural, not on all four.

Summary

Why escape:

- a local search stops at the bottom of its valley: pure intensification
- multi-start explores many valleys but ignores what it has learnt

How to escape:

- walk through worse solutions: simulated annealing, tabu search
- jump: iterated local search, VNS, GRASP, large neighbourhood search
- keep a population: genetic and memetic algorithms

Simulated annealing

Introduction: Simulated annealing

- Simulated annealing escapes from a local optimum by accepting worse moves *at random*
- We see its pseudo-code, its acceptance rule and its parameters, then we run it by hand on the TSP and on the KP, and on the medium TSP instance
- It is the first metaheuristic of the chapter, and the simplest to implement

The idea of simulated annealing

- *Simulated annealing* (denoted by *SA*) is a local search that chooses a *random* neighbour of the current solution and moves to it if it is better.
- If it is worse, it moves to it anyway with a probability that decreases with the worsening and with a parameter T, the *temperature*.
- The temperature starts high, so that at the beginning almost every move is accepted, and is lowered gradually, so that at the end only improving moves are accepted.
- The best solution visited is returned.

The pseudo-code of *SA*, for a minimization problem with objective f and neighbourhood N, is on the next slide.

Algorithm: Simulated annealing (1/2)

Input : starting solution S, initial temperature T_0, cooling factor $\alpha \in (0, 1)$, number L of moves per temperature, final temperature $T_{\min}$

Output : best solution found

```
1 function SimulatedAnnealing(S, T_0, α, L, T_min)
2 |   cur := S;  fcur := f(cur)          // current solution
3 |   best := cur;  fbest := fcur     // best solution found
4 |   T := T_0                                  // temperature
5 |   while T > T_min do
6 |   |   for l := 1 to L do
7 |   |   |   S' := random neighbour in N(cur)
8 |   |   |   δ := f(S') − fcur
9 |   |   |   v := RAND()
```

Algorithm: Simulated annealing (2/2)

```
function SimulatedAnnealing(S, T_0, α, L, T_min) (continued)
    while T > T_min (continued) do
        for l := 1 to L (continued) do
10          if δ ≤ 0 or v < e^{−δ/T} then
11              cur := S′;  fcur := fcur + δ          // move
                 accepted
12              if fcur < fbest then
13                  best := cur;  fbest := fcur
14              end
15          end
16      end
17      T := α · T                                    // cooling
18  end
19  return best
20 end
```

Where the name comes from

- The name comes from metallurgy: *annealing* is the process of heating a metal and cooling it slowly, so that its atoms settle into a state of minimum energy.
- Cooled too fast, the metal freezes into a disordered state of higher energy.
- Kirkpatrick, Gelatt and Vecchi (1983) and, independently, Černý (1985) proposed to simulate this process on an optimization problem, with the objective f in the role of the energy.
- The acceptance rule is the one used by Metropolis et al. (1953) to simulate the atoms.

Metropolis acceptance rule

- At temperature T, a neighbour S' of the current solution, with $\delta = f(S') - f(cur)$, is accepted with probability

$$P(\text{accept } S') = \begin{cases} 1 & \text{if } \delta \leq 0, \\ e^{-\delta/T} & \text{if } \delta > 0. \end{cases}$$

- The probability is implemented by drawing $v = \textsc{Rand}()$, uniform in $(0, 1)$, and accepting if $v < e^{-\delta/T}$, which happens with probability $e^{-\delta/T}$.

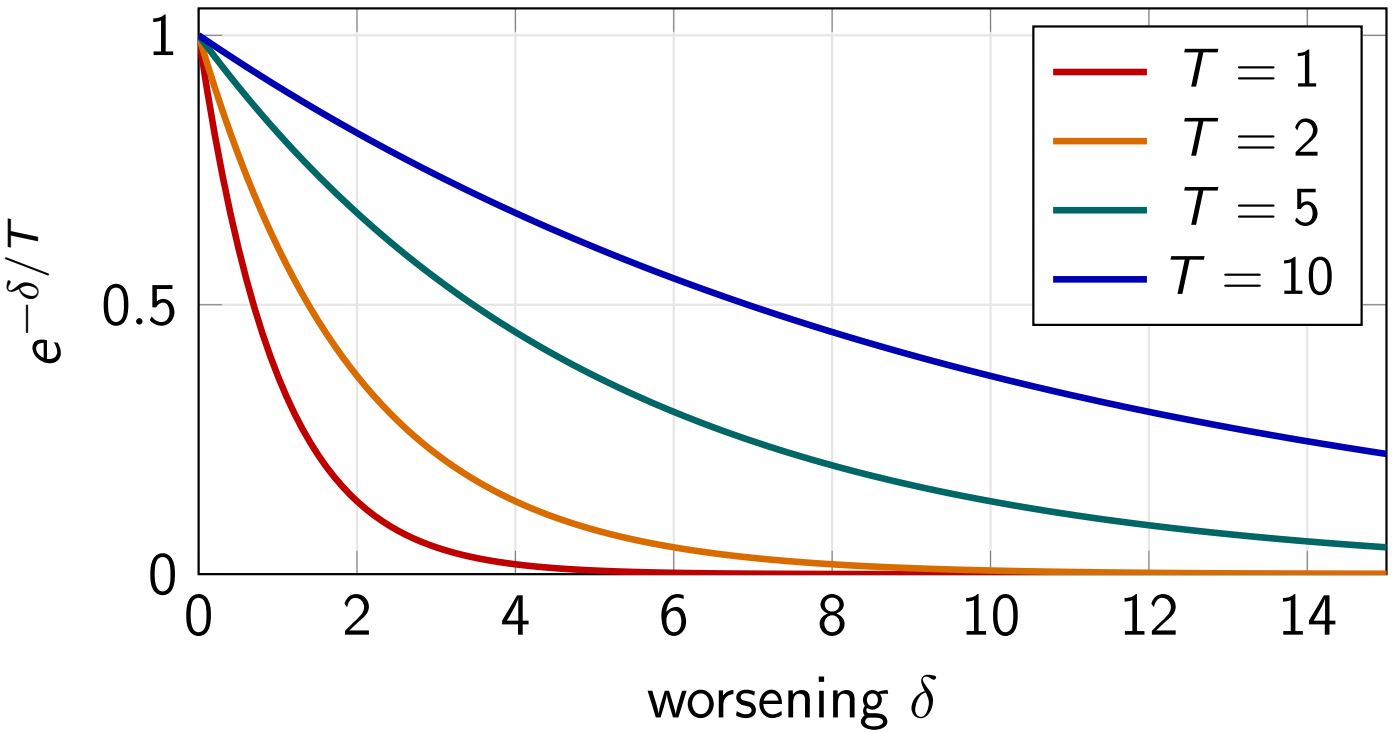
How the rule works (1/2)

- A worsening move is accepted more easily if the worsening δ is **small**, and if the temperature T is **high**.
- As $T \to \infty$ every move is accepted, and *SA* becomes a random walk (pure diversification).
- As $T \to 0$ only the moves with $\delta \leq 0$ are accepted, and *SA* becomes a local search with random choice of the neighbour (pure intensification).
- Lowering the temperature moves gradually from the first behaviour to the second.
- Moves with $\delta = 0$ are always accepted: *SA* walks freely on the plateaus of flat objectives.

How the rule works (2/2)

- *SA* never stops at a local optimum: a local optimum is left as soon as a worsening move is drawn and accepted.
- For this reason the **best** solution visited must be stored separately: the current solution at the end is not necessarily the best one.
- Only **one random neighbour** is evaluated per iteration, not the whole neighbourhood: an iteration of *SA* is very cheap, and *SA* performs many more iterations than a local search.
- For the TSP, a random 2-opt move is drawn by numbering the $M = \frac{n(n-3)}{2}$ moves (i, j) in lexicographic order and choosing the one in position $1 + \lfloor \textsc{Rand}() \cdot M \rfloor$; its δ is computed in $O(1)$.

The acceptance probability



- At $T = 10$ a worsening of 5 is accepted with probability $e^{-0.5} \approx 0.61$; at $T = 1$ with probability $e^{-5} \approx 0.007$.

Temperature and worsening

- The figure shows the acceptance probability as a function of the worsening, for four temperatures.
- The temperature has the same unit as the objective: T is "large" or "small" only compared with the typical values of δ.

Next: how to choose the sequence of temperatures and the other parameters.

The cooling schedule

- The **cooling schedule** is the sequence of temperatures.
- The pseudo-code uses the **geometric** schedule $T \leftarrow \alpha T$, the most common in practice, with α between 0.8 and 0.99, and L moves at each temperature.
- Other schedules are
  - the *linear* one, $T_k = T_0 - k\beta$,
  - and the *logarithmic* one, $T_k = \frac{\Gamma}{\ln(k+1)}$, which matters in the theory of *SA* but is far too slow in practice.

The parameters (1/2)

- **Initial temperature T_0.** It must be high enough to accept most worsening moves at the beginning. A common rule
  - fixes a target initial acceptance ratio χ_0 (for instance $\chi_0 = 0.8$),
  - estimates the average worsening $\bar{\delta}$ of a sample of random moves,
  - and sets $T_0 = -\bar{\delta}/\ln \chi_0$, so that $e^{-\bar{\delta}/T_0} = \chi_0$.

- **Moves per temperature L.** The search must be given time to adapt to each temperature; L is often taken proportional to the size of the neighbourhood.

The parameters (2/2)

- **Stopping rule.** The pseudo-code stops when $T \leq T_{\text{min}}$.
  - With the geometric schedule the number of temperatures is the smallest integer K with $T_0 \alpha^K \leq T_{\text{min}}$, i.e., $K = \lceil \ln(T_{\text{min}}/T_0)/\ln \alpha \rceil$ when the ratio is not an integer,
  - and the number of iterations is $K \cdot L$.

  Other common rules stop after a number of temperatures without any accepted move, or at a time limit.

- Slow cooling (α close to 1) gives better solutions but more iterations: the parameters trade quality for time, as every heuristic does.

Example: *SA* on the TSP reference instance (1/3)

- We start from the tour computed by the nearest neighbour heuristic from city 1 in the chapter «Constructive heuristics», $\sigma = (1, 3, 2, 4, 5)$ of length 25.
- We use $T_0 = 5$, $\alpha = 0.8$, $L = 2$ and $T_{\min} = 3$: the temperatures are 5, 4 and 3.2 (then $2.56 \leq 3$ stops the search), for a total of 6 iterations.
- The $M = 5$ moves of 2-opt on 5 cities are numbered

$$1 : (1, 3), \quad 2 : (1, 4), \quad 3 : (2, 4), \quad 4 : (2, 5), \quad 5 : (3, 5).$$

- The random numbers, with *seed* $= 1$, are those of the subsection «Random numbers»: the first number of each iteration (u) chooses the move $1 + \lfloor 5u \rfloor$, the second (v) is used in the acceptance test.

Example: *SA* on the TSP reference instance (2/3)

it	T	u	move	δ	v	$e^{-\delta/T}$	decision	current tour	L
1	5	0.0000	$1:(1,3)$	4	0.1315	0.4493	accept	$(1,2,3,4,5)$	29
2	5	0.7556	$4:(2,5)$	-7	—	—	accept	$(1,2,5,4,3)$	22
3	4	0.5328	$3:(2,4)$	2	0.2190	0.6065	accept	$(1,2,4,5,3)$	24
4	4	0.0470	$1:(1,3)$	-6	—	—	accept	$(1,4,2,5,3)$	18
5	3.2	0.6793	$4:(2,5)$	4	0.9347	0.2865	reject	$(1,4,2,5,3)$	18
6	3.2	0.3835	$2:(1,4)$	4	0.5194	0.2865	reject	$(1,4,2,5,3)$	18

Example: *SA* on the TSP reference instance (3/3)

- At iterations 1 and 3 a **worse** tour is accepted ($25 \to 29$ and $22 \to 24$), because v is smaller than the acceptance probability.
- At iterations 5 and 6 the temperature is lower, the worsening moves are less likely to be accepted, and both are rejected.
- The best tour, $(1, 4, 2, 5, 3)$, is the optimal tour $(1, 3, 5, 2, 4)$ traversed in the opposite direction: $SA(I) = 18 = z(I)$.
- On this small instance the optimal tour is the only 2-opt local optimum among the 12 tours, so a 2-opt local search would reach it as well: the example shows the mechanics of *SA*; its ability to escape from local optima shows on instances with bad local optima.

Simulated annealing with a penalty

- For the KP, the flip neighbourhood of a solution close to the capacity contains many infeasible solutions. Instead of discarding them, *SA* can **visit** them, with a penalty on the excess weight that makes them less attractive:

$$f_\beta(x) = \sum_{j \in \mathscr{I}} p_j x_j - \beta \cdot \max\left\{0, \sum_{j \in \mathscr{I}} w_j x_j - c\right\}, \qquad \beta > 0.$$

- *SA* maximizes f_β over all $x \in \{0,1\}^n$.
- The worsening of a move from x to y is $\delta = f_\beta(x) - f_\beta(y)$.
- The best solution is updated with **feasible** solutions only.

The infeasible region as a bridge

- On a feasible solution f_β is the profit.
- Passing through an infeasible solution, *SA* can reach feasible solutions that are not reachable through feasible flips only: the infeasible region acts as a **bridge** between valleys.
- The **penalty coefficient** β is a parameter:
  - if it is too small the search wanders in the infeasible region, where the profit is larger;
  - if it is too large the infeasible solutions are never accepted, and the bridge is closed.

Example: *SA* with a penalty on the KP reference instance (1/3)

- We start from $x = (1, 0, 1, 0)$, the solution $\{1, 3\}$ of the greedy algorithm by efficiency, of profit 16.
- It is a local optimum for the flip neighbourhood (chapter «Heuristic algorithms: definitions»): its feasible neighbours have profit 6 and 10.
- We use $\beta = 3$, $T_0 = 4$, $\alpha = 0.8$, $L = 2$, $T_{\min} = 2.5$ (temperatures 4, 3.2, 2.56: 6 iterations) and $seed = 3$.
- At each iteration u chooses the item $k = 1 + \lfloor 4u \rfloor$ to flip, and v is used in the acceptance test.

Example: *SA* with a penalty on the KP reference instance (2/3)

it	T	u	k	y	$\sum w_j y_j$	$f_3(y)$	δ	v vs $e^{-\delta/T}$	decision
1	4	0.0000	1	$(0,0,1,0)$	3	6	10	$0.3946 \geq 0.0821$	reject
2	4	0.2668	2	$(1,1,1,0)$	12	14	2	$0.3760 < 0.6065$	accept
3	3.2	0.5983	3	$(1,1,0,0)$	9	17	-3	—	accept
4	3.2	0.1411	1	$(0,1,0,0)$	4	7	10	$0.0366 < 0.0439$	accept
5	2.56	0.0379	1	$(1,1,0,0)$	9	17	-10	—	accept
6	2.56	0.1505	1	$(0,1,0,0)$	4	7	10	$0.5582 \geq 0.0201$	reject

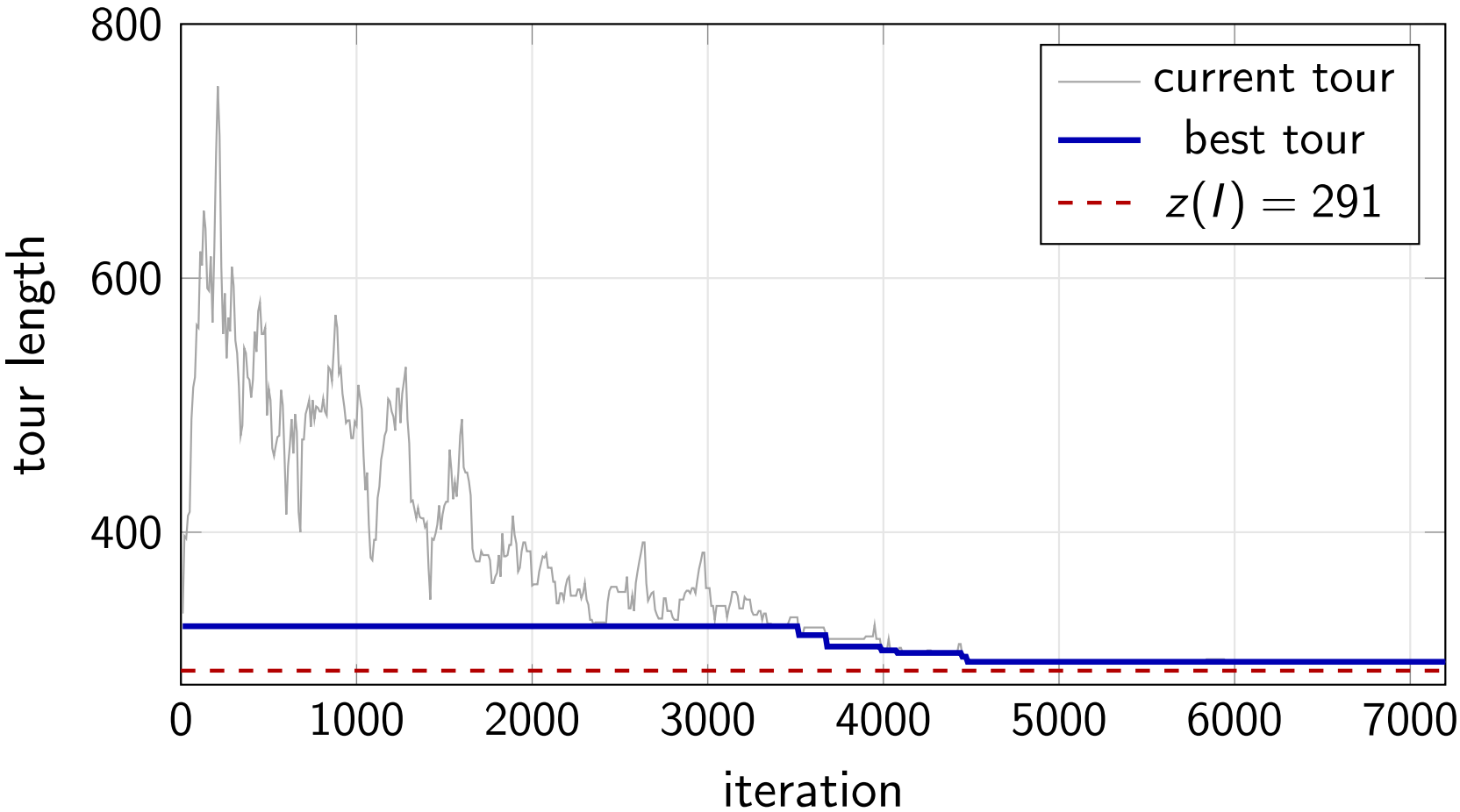
Example: *SA* with a penalty on the KP reference instance (3/3)

- At iteration 2 the search moves to the **infeasible** solution $(1, 1, 1, 0)$, of weight $12 > 9$: its profit is 23 and its penalized value $23 - 3 \cdot 3 = 14$, a worsening of 2 accepted with probability 0.6065.
- From there, removing item 3 gives the feasible solution $(1, 1, 0, 0)$ of profit 17: the optimum, $SA(I) = 17 = z(I)$.
- Iteration 4 shows that at this temperature even a worsening of 10 can be accepted, with a small probability; iteration 5 goes back to the optimum.

An experiment on TSP20

- We run *SA* with the 2-opt neighbourhood on TSP20, from the nearest neighbour tour of length 326, with $L = 100$ and $T_{\min} = 0.5$.
- The figure on the next slide shows one run, with $T_0 = 20$, $\alpha = 0.95$ and $seed = 1$:
  - the length of the current tour (gray) and of the best tour (blue),
  - along the 7200 iterations, every 10 iterations.
- The data are produced by the C++ code of this chapter (`make data`).

One run of *SA* on TSP20



What the run shows

- At the beginning the temperature is high compared with the typical worsening of a 2-opt move, and the current tour gets much **worse** than the starting one (up to length 765): the search is almost a random walk.
- As T decreases, fewer worsening moves are accepted and the current tour descends into a valley; at the end the search is a local search.
- The best tour, of length 298, is found at iteration 4479: it is better than the 2-opt local optimum (319), but it is not optimal.
- The table on the next slide reports, for 10 runs with $seed = 1, \ldots, 10$ and several parameters, the number of iterations, the minimum, the average and the maximum length of the best tour, and the number of runs that reach the optimum 291.

SA on TSP20 over ten seeds

α	T_0	iterations	min	average	max	optimal runs
0.9	1	700	291	310.3	322	1
0.9	5	2200	291	297.3	305	4
0.9	20	3600	291	300.3	323	4
0.9	100	5100	291	297.8	311	3
0.95	1	1400	291	302.3	319	3
0.95	5	4500	291	294.3	298	5
0.95	20	7200	291	294.3	298	5
0.95	100	10400	291	293.2	299	7
0.99	1	6900	291	298.8	305	4
0.99	5	23000	291	291.7	298	9
0.99	20	36800	291	291.0	291	10
0.99	100	52800	291	291.0	291	10

The effect of the parameters

- With $T_0 = 1$ the temperature is too low from the beginning: few worsening moves are accepted, and *SA* behaves like a randomized local search.
- Slower cooling ($\alpha = 0.99$) spends more iterations at every temperature range and reaches the optimum in almost every run, at the price of many more iterations.
- A single run says little: the comparison must look at the distribution over several seeds, as recalled in the chapter «Heuristic algorithms: definitions».

The Boltzmann distribution

- At a fixed temperature T, the sequence of current solutions of SA is a Markov chain: the next solution depends only on the current one.
- If the neighbourhood is
  - symmetric ($S' \in N(S)$ if and only if $S \in N(S')$, all neighbourhoods having the same size)
  - and connected (every solution can reach every other one through moves),

  the chain has a unique stationary distribution, the **Boltzmann distribution**:

$$\pi_T(S) = \frac{e^{-f(S)/T}}{\sum_{S'} e^{-f(S')/T}} \qquad \text{for every solution } S.$$

Convergence in the limit

- For two solutions S_1 and S_2, $\frac{\pi_T(S_1)}{\pi_T(S_2)} = e^{(f(S_2)-f(S_1))/T}$: better solutions are more probable, and the more so the lower the temperature.
- As $T \to 0$, π_T concentrates on the optimal solutions.
- Hajek (1988) proved that with the logarithmic schedule $T_k = \frac{\Gamma}{\ln(k+1)}$, with Γ at least the depth of the deepest valley that does not contain an optimal solution, the probability that the current solution is optimal tends to 1.
- The result is a guarantee *in the limit*: the number of iterations needed can be larger than the number of solutions, and in practice the faster geometric schedule is used, without guarantees.

Summary (1/2)

Simulated annealing:

- one random neighbour per iteration, accepted with the Metropolis rule: always if $\delta \leq 0$, with probability $e^{-\delta/T}$ otherwise
- high temperature: random walk; low temperature: local search
- the best solution visited is stored apart

Parameters:

- geometric cooling $T \leftarrow \alpha T$, initial temperature T_0, L moves per temperature, T_{min}
- slow cooling: better solutions, more iterations

Summary (2/2)

Examples and experiments:

- on the TSP and on the KP reference instances *SA* reaches the optimum, accepting worse moves on the way
- a penalty on the excess weight lets *SA* cross the infeasible region, a bridge between valleys
- on TSP20 the result depends on T_0 and α: look at several seeds

Properties:

- at fixed T the stationary distribution is the Boltzmann distribution
- the logarithmic schedule converges to an optimum only in the limit

Tabu search

Introduction: Tabu search

- Tabu search escapes from a local optimum by accepting worse moves *deterministically*, with a memory of the recent moves
- We see its pseudo-code for $P \,||\, C_{\max}$, its components (attributes, aspiration, tenure), and we run it by hand on the reference instance
- Then we measure the effect of the tenure on the medium instances of the TSP and of $P \,||\, C_{\max}$

The idea of tabu search

- *Tabu search* (denoted by *TS*) explores the neighbourhood of the current solution completely and moves to the *best* neighbour, even if it is worse than the current solution.
- To avoid going back to the solutions just visited, it keeps a short-term memory, the *tabu list*: the moves that would undo the recent moves are forbidden (*tabu*) for a number of iterations, the *tenure*,
- unless they lead to a solution better than the best one found (*aspiration criterion*).

Next: the pseudo-code of *TS* for $P \,||\, C_{\max}$, with the move neighbourhood and the lexicographic comparison; the attribute (j, h) forbids moving job j back to machine h.

Algorithm: Tabu search for $P \,||\, C_{\max}$ (1/3)

Input : times $t[1..n]$, m machines, starting assignment $mach[1..n]$, $tenure$, $maxIter$

Output : best assignment found

```
1 function TabuSearch(t, m, mach, tenure, maxIter)
2 |   L := loads of mach;  tabu[j][h] := 0 for all j, h
3 |   best := mach;  bestC := max_h L[h]
4 |   LB := max{max_j t[j], ⌈∑_j t[j]/m⌉}          // stop if
  |    reached
5 |   it := 0
```

Algorithm: Tabu search for $P \,||\, C_{\max}$ (2/3)

function *TabuSearch*$(t, m, mach, tenure, maxIter)$ (continued)

```
6   |   while it < maxIter and bestC > LB do
7   |   |   it := it + 1;  bj := 0          // best non-tabu move
    |   |    (bj, bh)
8   |   |   for j := 1 to n do
9   |   |   |   for h := 1 to m, h ≠ mach[j] do
10  |   |   |   |   L' := the loads after moving job j from
    |   |   |   |    mach[j] to h
11  |   |   |   |   if tabu[j][h] ≥ it and not max_k L'[k] < bestC
    |   |   |   |    then continue                    // tabu
12  |   |   |   |   if bj = 0 or L' is lexicographically better than
    |   |   |   |    bL then (bj, bh, bL) := (j, h, L')
13  |   |   |   end
14  |   |   end
```

Algorithm: Tabu search for $P \,||\, C_{\max}$ (3/3)

function *TabuSearch*$(t, m, mach, tenure, maxIter)$ (continued)

    while $it < maxIter$ ***and*** $bestC > \mathrm{LB}$ (continued) **do**

15   **if** $bj = 0$ **then break**

16        $tabu[bj][mach[bj]] := it + tenure$ `// forbid going back`

17        $mach[bj] := bh$; $L := bL$ `// even if worse`

18   **if** $\max_k L[k] < bestC$ **then**

19            $bestC := \max_k L[k]$; $best := mach$

20   **end**

21   **end**

22   **return** *best*

23 **end**

Deterministic, with memory

- Tabu search was proposed by Glover (1986); the reference book is Glover and Laguna, *Tabu Search*, Kluwer, 1997.
- Unlike *SA*, it is **deterministic**: it escapes from a local optimum because it always moves, to the least bad neighbour if no neighbour is better.
- Without memory, this rule would **cycle**:
  - from a local optimum S the search moves to the best neighbour S', which is worse,
  - and from S' the best neighbour is typically S again.
- The tabu list breaks the cycle by forbidding the reverse move.

Attributes and aspiration

- **Attributes.** Storing the visited solutions and comparing every neighbour with them would be too expensive. The tabu list stores **attributes** of the recent moves instead:
  - here the pair (j, h) "job j has left machine h";
  - for the TSP, the edges just removed from the tour.

- An attribute forbids more than the single solution just left: every solution with job j on machine h is tabu, even the ones never visited.

- The **aspiration criterion** repairs this excess of caution: a tabu move is allowed if it leads to a solution better than the best one found, which certainly has not been visited.

The tenure

- **Tenure.** If it is too short, the search can still cycle through longer cycles; if it is too long, too many moves are forbidden and the search is driven far from the good regions. Typical values are between 5 and 20, or proportional to $\sqrt{n}$; drawing the tenure at random in an interval at every iteration (robust tabu search, Taillard 1991) makes cycles unlikely.

Stopping rule and long-term memory

- **Stopping rule.** A maximum number of iterations, or of iterations without improvement of the best solution. For $P \,||\, C_{\max}$ we can also stop when the makespan reaches the lower bound LB: then the solution is optimal, and the search can only waste time.

- **Long-term memory.** Besides the tabu list (short-term memory), *TS* can record how often each attribute appears in the visited solutions. *Diversification* penalizes the frequent attributes, to push the search toward unexplored regions; *intensification* restarts from the best solutions found, and fixes the attributes they share.

Example: TS on the $P \,||\, C_{\max}$ reference instance (1/4)

- We start from the solution computed by LPT in the chapter «Constructive heuristics», $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$, with loads $(11, 8, 8)$, $C_{\max} = 11$ and $\sum_h L_h^2 = 249$.
- It is a local optimum for the move neighbourhood: at the first iteration the 14 neighbours are

move	$1 \to M_2$	$1 \to M_3$	$2 \to M_1$	$2 \to M_3$	$3 \to M_1$
$(C_{\max}, \sum_h L_h^2)$	$(13, 269)$	$(13, 269)$	$(16, 329)$	$(13, 299)$	$(15, 305)$
move	$3 \to M_2$	$4 \to M_1$	$4 \to M_2$	$5 \to M_2$	$5 \to M_3$
$(C_{\max}, \sum_h L_h^2)$	$(12, 281)$	$(15, 305)$	$(12, 281)$	$(11, 249)$	$(11, 249)$
move	$6 \to M_1$	$6 \to M_3$	$7 \to M_2$	$7 \to M_3$	
$(C_{\max}, \sum_h L_h^2)$	$(14, 285)$	$(11, 267)$	$(11, 249)$	$(11, 249)$	

Example: *TS* on the $P \,||\, C_{\max}$ reference instance (2/4)

- No neighbour has $C_{\max} < 11$, and the four best ones have $C_{\max} = 11$ and $\sum_h L_h^2 = 249$, as the current solution: a local search stops here.
- *TS*, with *tenure* $= 6$, moves instead to the best non-tabu neighbour (the first of the four, job 5 to M_2), and continues as in the table on the next slide.

Example: TS on the $P \,||\, C_{\max}$ reference instance (3/4)

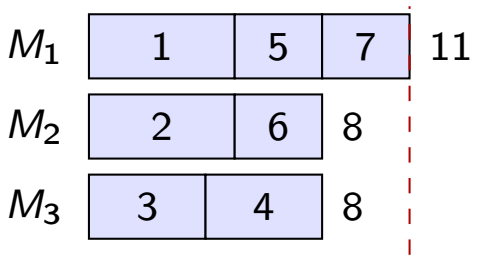
it	move	loads	$C_{\max}$	$\sum_h L_h^2$	$bestC$	new tabu attribute
1	job 5: $M_1 \to M_2$	$(8, 11, 8)$	11	249	11	$(5, M_1)$ until it. 7
2	job 5: $M_2 \to M_3$	$(8, 8, 11)$	11	249	11	$(5, M_2)$ until it. 8
3	job 6: $M_2 \to M_1$	$(11, 5, 11)$	11	267	11	$(6, M_2)$ until it. 9
4	job 7: $M_1 \to M_2$	$(8, 8, 11)$	11	249	11	$(7, M_1)$ until it. 10
5	job 3: $M_3 \to M_1$	$(12, 8, 7)$	12	257	11	$(3, M_3)$ until it. 11
6	job 6: $M_1 \to M_3$	$(9, 8, 10)$	10	245	10	$(6, M_1)$ until it. 12
7	job 4: $M_3 \to M_2$	$(9, 12, 6)$	12	261	10	$(4, M_3)$ until it. 13
8	job 7: $M_2 \to M_3$	$(9, 9, 9)$	9	243	9	$(7, M_2)$ until it. 14

Example: TS on the $P \,||\, C_{\max}$ reference instance (4/4)

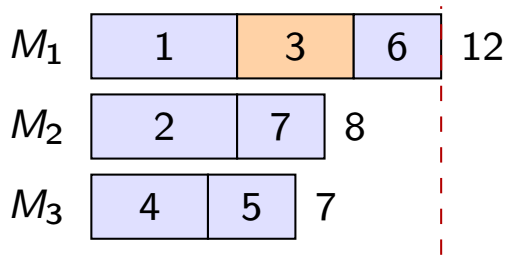
- At iteration 2 the best neighbour would move job 5 back to M_1, returning to the starting solution: the move is tabu (attribute $(5, M_1)$ until iteration 7), and the search moves job 5 to M_3.
- At iteration 5 the four neighbours with $C_{\max} = 11$ are all tabu, and none satisfies the aspiration criterion ($C_{\max} < bestC = 11$): the best non-tabu move **worsens** the makespan to 12. This is how TS leaves the plateau.
- At iteration 6 the makespan drops to 10, and at iteration 8 to 9: $C_{\max} = \mathrm{LB} = 9$, so the solution $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$ is optimal and the search stops: $TS(I) = 9 = z(I)$.

Three solutions visited by *TS*

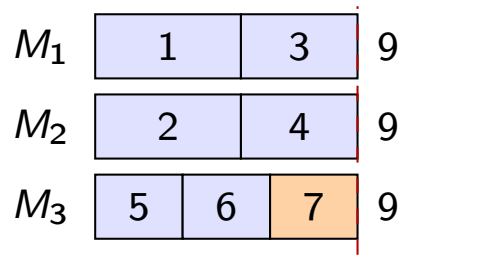
start (LPT): $C_{max} = 11$



iteration 5: $C_{max} = 12$



iteration 8: $C_{max} = 9$



- The Gantt charts of the starting solution, of the solution after the worsening move of iteration 5 (the moved job in orange), and of the optimal solution reached at iteration 8.
- With *tenure* ≤ 5, *TS* does not reach the optimum on this instance within 50 iterations: the tabu list is too short to prevent the search from going back to the plateau of makespan 11.

Tabu search for the TSP

- For the TSP with the 2-opt neighbourhood, the natural attributes are the **edges**: when a move removes the edges (σ_i, σ_{i+1}) and (σ_j, σ_{j+1}), these two edges may not be added back to the tour for *tenure* iterations, unless the move gives a tour shorter than the best one.

- Everything else is as in the pseudo-code for $P \,||\, C_{\max}$: at every iteration all the $\frac{n(n-3)}{2}$ moves are evaluated in $O(1)$ each, and the best non-tabu one is performed.

- The tables on the next slide report the result of *TS* on the medium instances, for several values of the tenure:
  - from the nearest neighbour tour (326) for TSP20, with 200 iterations;
  - from the LPT solution (73) for PCM15, with 300 iterations; the table also gives the number of iterations performed (300 unless the lower bound is reached).

The effect of the tenure

TSP20: *TS* with 2-opt, 200 iterations ($z(I) = 291$)						
tenure	0	1	2	5	10	20
best length	319	291	291	291	291	291

PCM15: *TS* with the move neighbourhood, at most 300 iterations ($z(I) = 69$)										
tenure	0	1	2	3	5	7	10	15	20	30
best $C_{\max}$	73	73	73	73	73	72	71	69	70	70
iterations	300	300	300	300	300	300	300	231	300	300

The tenure must be tuned

- With *tenure* $= 0$ there is no memory, and *TS* cycles: on TSP20 it never leaves the first 2-opt local optimum.
- On PCM15 the move neighbourhood has a large plateau, and *TS* needs a long tabu list to cross it;
- but a tabu list that is too long forbids too much, and with *tenure* ≥ 20 the optimum is lost again.
- The tenure must be tuned, and its best value depends on the problem and on the instance.

Summary

Tabu search:

- moves to the best non-tabu neighbour, even if it is worse: deterministic
- the tabu list stores attributes of the recent moves; the aspiration criterion allows a tabu move that improves on the best solution

Tenure:

- too short: the search cycles; too long: it forbids too much
- on the $P \,||\, C_{\max}$ reference instance $\mathit{tenure} = 6$ reaches the optimum, $\mathit{tenure} \leq 5$ does not
- on PCM15 only $\mathit{tenure} = 15$ reaches 69 within 300 iterations

Iterated local search

Introduction: Iterated local search

- Iterated local search escapes from a local optimum by *perturbing* it and descending again
- We see its pseudo-code for the TSP, with the double-bridge kick, and the roles of its three components
- Then we follow two iterations on the medium TSP instance, where it reaches the optimum

The idea of iterated local search

- *Iterated local search* (denoted by *ILS*) applies a local search, and then repeats:
  - it *perturbs* the current local optimum with a random modification (a *kick*) that the local search cannot easily undo,
  - applies the local search to the perturbed solution,
  - and decides with an *acceptance criterion* whether the new local optimum replaces the current one.

- It is a walk among the local optima.

Next: the pseudo-code of *ILS* for the TSP,
with the 2-opt local search and the double-bridge kick.

Algorithm: Iterated local search for the TSP (1/2)

Input : distances d, starting tour $\sigma[1..n]$, number of iterations *maxIter*

Output : the last accepted local optimum

```
1  function IteratedLocalSearch(d, σ, maxIter)
2      cur := TwoOptLS(σ); lcur := L(cur)          // first local optimum
3      for it := 1 to maxIter do
4          nw := DoubleBridge(cur)                 // perturbation
5          nw := TwoOptLS(nw); lnew := L(nw)       // local search
6          if lnew < lcur then
7              cur := nw; lcur := lnew             // acceptance
8          end
9      end
10     return cur
11 end
```

Algorithm: Iterated local search for the TSP (2/2)

10 **function** *DoubleBridge*(σ)
11 $\quad p_1 := 1 + \lfloor \textsc{Rand}() \cdot (n-3) \rfloor$
12 $\quad p_2 := p_1 + 1 + \lfloor \textsc{Rand}() \cdot (n-2-p_1) \rfloor$
13 $\quad p_3 := p_2 + 1 + \lfloor \textsc{Rand}() \cdot (n-1-p_2) \rfloor$
14 $\quad A := \sigma[1..p_1]$; $B := \sigma[p_1{+}1..p_2]$; $C := \sigma[p_2{+}1..p_3]$; $D := \sigma[p_3{+}1..n]$
15 $\quad$ **return** (A, C, B, D)
16 **end**

The three components of *ILS*

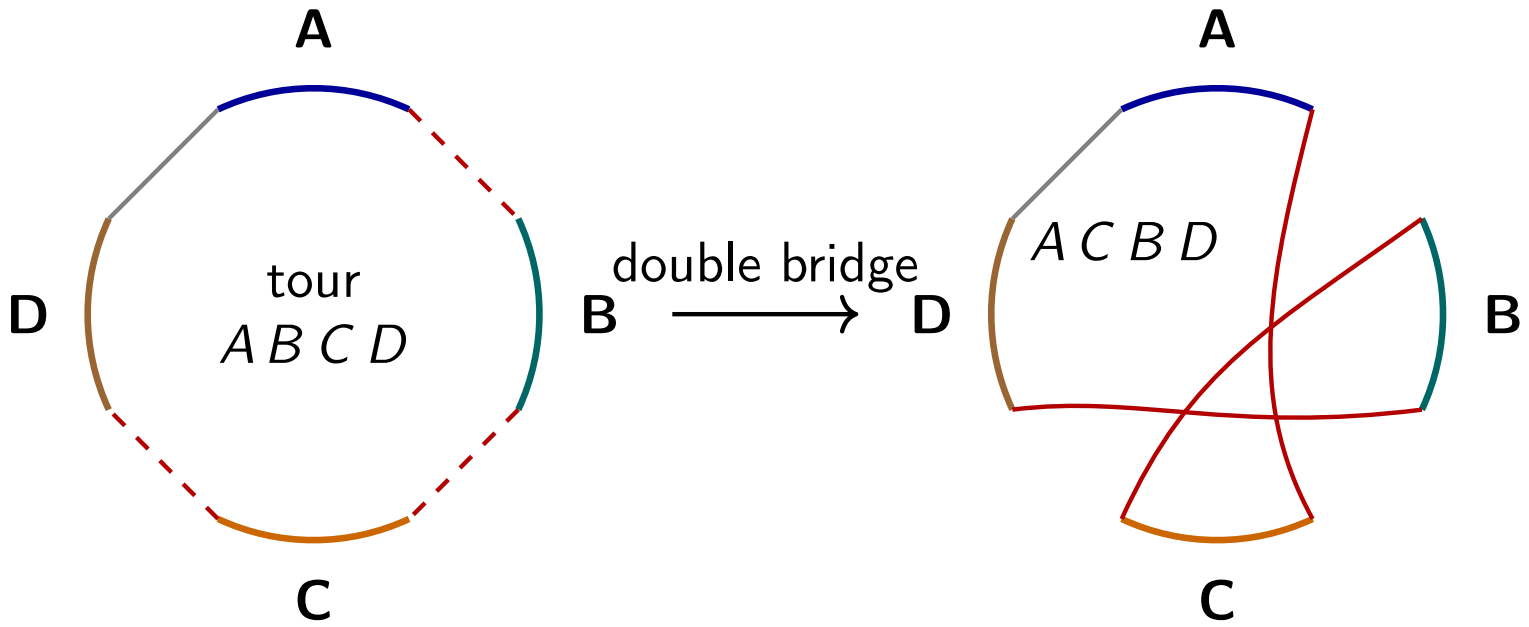
- The scheme was described in its general form by Lourenço, Martin and Stützle (2003), in the *Handbook of Metaheuristics*; for the TSP it goes back to the "large-step Markov chains" of Martin, Otto and Felten (1991), who introduced the double-bridge kick.

- *ILS* has three components, and each one has a role:
  - the **local search** intensifies: it brings every perturbed solution to the bottom of its valley;
  - the **perturbation** diversifies: it moves the search to another valley;
  - the **acceptance criterion** balances the two: accepting only better local optima (as in the pseudo-code) intensifies, accepting every new local optimum diversifies; intermediate rules accept a worse local optimum with a probability, as in *SA*.

The strength of the perturbation

- The **strength** of the perturbation is the key parameter.
- If it is too weak, the local search undoes it and returns to the same local optimum.
- If it is too strong, the perturbed solution has nothing in common with the local optimum, and *ILS* becomes a random multi-start.
- A good kick changes the solution in a way that the local search *cannot* undo, but keeps most of it.

The double bridge



- The **double bridge** cuts the tour into four segments A, B, C, D and reconnects them as $A\,C\,B\,D$: it removes three edges (dashed) and adds three new ones (red), while the edge between the end of D and the beginning of A is kept.

Why 2-opt cannot undo the double bridge

- No segment is reversed, whereas every 2-opt move reverses a segment.
- When B and C have at least two cities each, a single 2-opt move cannot undo the kick, and the 2-opt local search, which accepts only improving moves, rarely goes back to the same local optimum.
- If B and C are single cities, the kick changes only two edges and is itself a 2-opt move: the exercise «Iterated local search and VNS 2» uses this case.

Example: *ILS* on the medium TSP instance (1/3)

On TSP20, the 2-opt local search from the nearest neighbour tour (length 326) stops at the local optimum

$$\begin{aligned} cur = (&1, 16, 11, 19, 7, 18, 13, 8, 17, 2,\\ &4, 10, 20, 14, 12, 5, 6, 9, 3, 15),\\ L(cur) = &319. \end{aligned}$$

With $seed = 1$, the three random numbers of the first kick are 0.0000, 0.1315, 0.7556, so

$$\begin{aligned} p_1 &= 1 + \lfloor 0.0000 \cdot 17 \rfloor = 1,\\ p_2 &= 1 + 1 + \lfloor 0.1315 \cdot 17 \rfloor = 4,\\ p_3 &= 4 + 1 + \lfloor 0.7556 \cdot 15 \rfloor = 16. \end{aligned}$$

Example: *ILS* on the medium TSP instance (2/3)

The segments are

$$A = (1),$$
$$B = (16, 11, 19),$$
$$C = (7, 18, 13, 8, 17, 2, 4, 10, 20, 14, 12, 5),$$
$$D = (6, 9, 3, 15).$$

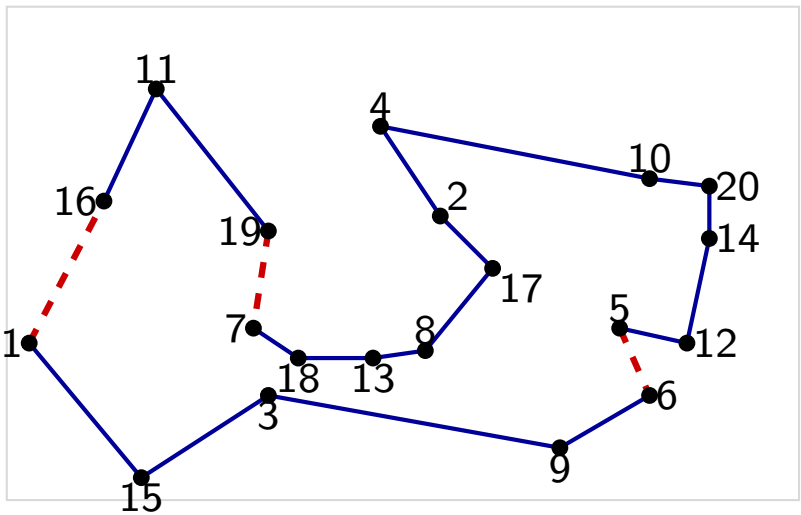
The kick removes the edges $(1, 16)$, $(19, 7)$, $(5, 6)$ and adds $(1, 7)$, $(5, 16)$, $(19, 6)$.

Example: *ILS* on the medium TSP instance (3/3)

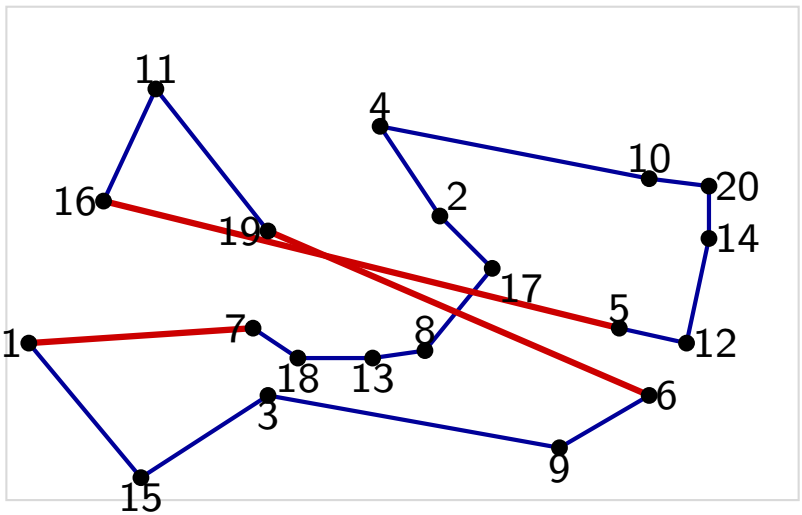
- The perturbed tour $A\,C\,B\,D$ has length 432, much worse than 319.
- The 2-opt local search brings it to a new local optimum of length $303 < 319$, which is accepted.
- The second kick ($p = (8, 14, 16)$) gives, after 2-opt, a tour of length $291 = z(I)$.
- The third gives again a tour of length 291, not better, and it is rejected.

The first iteration of *ILS* on TSP20

(a) local optimum: $L = 319$



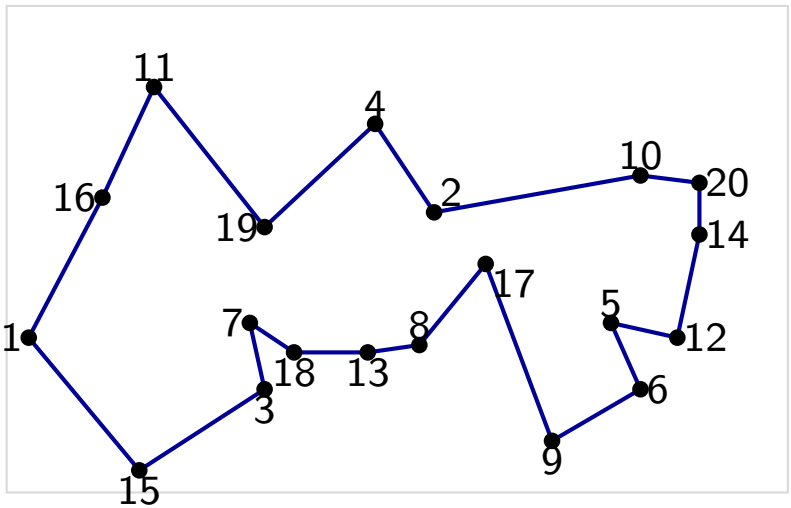
(b) after the double bridge: $L = 432$



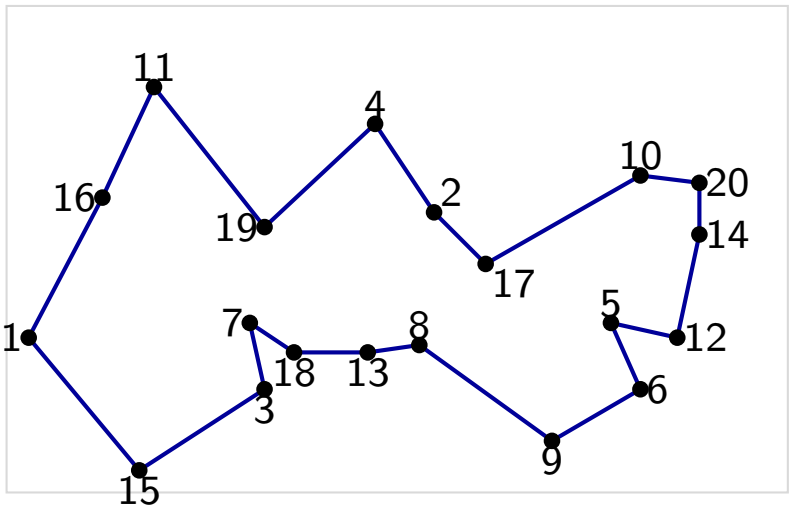
- Panel (a) shows the local optimum with the three edges removed by the kick (dashed), panel (b) the perturbed tour with the three new edges.

The local optima after the first and the second iteration

(c) after 2-opt: $L = 303$



(d) after the second iteration: $L = 291$



- Panels (c) and (d) show the local optima after the first and the second iteration. The kick moves the segment C before B; the 2-opt local search then repairs the long edges it created, keeping the good parts of the old tour.

ILS on TSP20 over ten seeds

- With 100 iterations and $seed = 1, \ldots, 10$, *ILS* reaches the optimum 291 in every run.
- The iteration at which the optimum is reached is

$$2,\ 3,\ 1,\ 7,\ 8,\ 11,\ 15,\ 6,\ 2,\ 3$$

  for the ten seeds.
- Each iteration costs one kick and one 2-opt local search, which starts from a solution close to a local optimum and needs few moves.

Summary

Iterated local search:

- a walk among the local optima: kick, local search, acceptance
- local search intensifies, perturbation diversifies, acceptance balances the two

The double bridge:

- reconnects $A\,B\,C\,D$ as $A\,C\,B\,D$ without reversing any segment
- a single 2-opt move cannot undo it when B and C have at least two cities
- on TSP20, *ILS* reaches the optimum 291 in every run, within 15 iterations

Summary of the first part

- A metaheuristic guides a local search beyond the first local optimum, balancing intensification and diversification.
- The four problems, their neighbourhoods, the random numbers and the reference and medium instances are the setting of the whole chapter.
- Simulated annealing and tabu search walk out of a valley through worse solutions, at random or with memory.
- Iterated local search jumps to another valley with a kick that the local search cannot undo.
- The second part presents VNS, GRASP, large neighbourhood search, genetic and memetic algorithms, and compares all the methods on the medium instances.