Heuristic algorithms: definitions

Sapienza Università di Roma

Outline

Why heuristic algorithms?

Heuristic algorithms

The problems and the assumptions

Primal and dual bounds

Classification of heuristic algorithms

Solution representation and search space

Neighbourhoods and local optima

Intensification and diversification

How to evaluate a heuristic algorithm

Typical mistakes

In summary

Learning objectives

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

- define a heuristic algorithm, and compute its gap with respect to a dual bound;
- tell primal bounds from dual bounds on the four reference instances;
- classify heuristic algorithms, and describe the solution representation and the search space;
- define neighbourhoods and local optima, and explain intensification and diversification;
- evaluate a heuristic algorithm by quality, running time and repeated runs.

Prerequisites

- From the module *Combinatorial Optimization Problems*, the 0/1 knapsack problem, the bin packing problem, the travelling salesman problem and $P \,||\, C_{\max}$: instances, feasible solutions and optimal values of the four problems used as examples.
- From the module *Integer linear programming*, Chapter 2 «Primal and dual bounds»: the LP relaxation as a source of dual bounds.
- A short recap is given here for the 0/1 knapsack problem.

Why heuristic algorithms?

Introduction: Why heuristic algorithms?

- We start from a practical question: what can we do when an optimization problem cannot be solved to optimality in the time we have?
- Many optimization problems of practical relevance are hard, and exact algorithms may be too slow on them
- Heuristic algorithms are the answer: they give good solutions quickly, without proving that they are optimal

Optimization problems and hard problems

- Let $\mathscr{P}$ be an **optimization problem**.
- For every **instance** I of $\mathscr{P}$, let $z(I)$ be the **optimal value** of $\mathscr{P}$ on I.
- Many optimization problems of practical relevance are **NP-hard**: no algorithm is known that solves all their instances to optimality in polynomial time, and it is widely believed that none exists.

When the time is not available

- The four problems used as examples in this chapter are all NP-hard: the knapsack problem, the bin packing problem, the travelling salesman problem and the scheduling problem $P \,||\, C_{\max}$.
- For such problems, an exact algorithm may require a running time that grows exponentially with the size of the instance.
- When this time is not available, we accept a solution that is *good*, even if it is not proved to be optimal.

Summary

The difficulty:

- many optimization problems of practical relevance are NP-hard
- an exact algorithm may need a running time that grows exponentially with the size of the instance

The way out:

- when the time is not available, we accept a good solution, not proved to be optimal
- this is the job of heuristic algorithms

Heuristic algorithms

Introduction: Heuristic algorithms

- In this section we define what a heuristic algorithm is, and the value $\mathscr{A}(I)$ that it computes
- We compare $\mathscr{A}(I)$ with the optimal value $z(I)$, for minimization and maximization problems
- We see where approximation algorithms fit, and the trade-off at the heart of every heuristic

Heuristic algorithm

Definition 1: Heuristic algorithm

Given an ***optimization problem*** $\mathscr{P}$, *let* $z(I)$ *be the* ***optimal value*** *of* $\mathscr{P}$ *on an* ***instance*** I. *A* ***heuristic algorithm*** *for* $\mathscr{P}$ *is an algorithm that, for every instance* I *of* $\mathscr{P}$, *computes a feasible solution of* I, *without guaranteeing that it is optimal. We denote by* $\mathscr{A}(I)$ *the value of the solution computed by the heuristic algorithm* $\mathscr{A}$ *on the instance* I.

Heuristic value and optimal value

- Since the solution computed by $\mathscr{A}$ is feasible, for every instance I

$$\begin{cases} \mathscr{A}(I) \geq z(I) & \text{if } \mathscr{P} \text{ is a minimization problem,} \\ \mathscr{A}(I) \leq z(I) & \text{if } \mathscr{P} \text{ is a maximization problem.} \end{cases}$$

- The heuristic value is never better than the optimal value: in the best case it is equal to it.

Why heuristic algorithms are needed

- The four problems used as examples are all **NP-hard**:
  - the 0/1 knapsack problem;
  - the bin packing problem;
  - the travelling salesman problem;
  - the scheduling problem $P \,||\, C_{\max}$.
- An exact algorithm may require a running time that grows exponentially with the size of the instance.

Approximation algorithms

- **Approximation algorithms** are heuristic algorithms with a *proved* worst-case guarantee on the ratio between $\mathscr{A}(I)$ and $z(I)$.
- Most of the heuristic algorithms used in practice have no such guarantee.
- Their quality is assessed on the instances at hand, as described in the section «How to evaluate a heuristic algorithm» of this chapter.

Quality versus running time

A heuristic algorithm trades the **quality** of the solution for the **running time**.

- The point of a heuristic is to compute, in a short time, a solution whose value is close to $z(I)$.

Summary

Heuristic algorithms:

- compute a feasible solution of every instance, without guaranteeing that it is optimal
- $\mathscr{A}(I) \geq z(I)$ for a minimization problem, $\mathscr{A}(I) \leq z(I)$ for a maximization problem

Approximation algorithms:

- heuristic algorithms with a proved worst-case guarantee
- most heuristics used in practice have none: they trade quality for running time

The problems and the assumptions

Introduction: The problems and the assumptions

- The heuristic algorithms are illustrated on four classical combinatorial optimization problems
- In this section we recall their notation and the assumptions on their instances
- We fix one reference instance for each problem, with known optimal value: on them the quality of every heuristic solution can be measured exactly

Four classical problems

- The heuristic algorithms are illustrated on four classical combinatorial optimization problems:
  - 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; the table on the next slide recalls 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	packing patterns	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

- We assume that every instance satisfies the assumptions of its problem:
  - KP and BPP: $w_j \leq c$ for all items and $\sum_j w_j > c$;
  - TSP: $n \geq 3$ and symmetric distances;
  - $P \,||\, C_{\max}$: $n > m$.
- The following four instances are used throughout the notes on heuristic algorithms.
- Their optimal values are known, so the quality of every heuristic solution can be measured exactly.

Example: KP reference instance

A knapsack of capacity $c = 9$ and 4 items with profits and weights

	$j = 1$	$j = 2$	$j = 3$	$j = 4$
p_j	10	7	6	4
w_j	5	4	3	3

- The optimal solution is $\tilde{S} = \{1, 2\}$, of weight 9 and profit $z(I) = 17$.

Example: BPP reference instance

Bins of capacity $c = 7$ and 6 items with weights

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
w_j	4	4	5	3	2	3

- The total weight is $21 = 3c$, so at least 3 bins are needed.
- The packing patterns $\{1, 4\}$, $\{2, 6\}$, $\{3, 5\}$ use exactly 3 bins, all of them full: $z(I) = 3$.

Example: TSP reference instance

5 cities with the symmetric distances

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 instance is metric.
- Among its 12 distinct tours, the unique optimal one is $\sigma = (1, 3, 5, 2, 4)$, of length $z(I) = 18$.

Example: $P \,||\, C_{\max}$ reference instance

$m = 3$ identical machines and 7 jobs with processing times

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$
t_j	5	5	4	4	3	3	3

- The total processing time is 27, so the makespan is at least $\lceil 27/3 \rceil = 9$.
- The sets $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$ attain it: $z(I) = 9$.

Summary

The four problems:

- KP and BPP (items, weights, capacity), TSP (cities, distances), $P \,||\, C_{\max}$ (jobs, times, machines)
- every instance satisfies the assumptions of its problem

The reference instances:

- one small instance for each problem, used throughout the notes on heuristic algorithms
- optimal values: $z(I) = 17$ (KP), 3 (BPP), 18 (TSP), 9 ($P \,||\, C_{\max}$)

Primal and dual bounds

Introduction: Primal and dual bounds

- A heuristic gives a solution, but how good is it? In this section we learn to answer this question even when the optimal value is unknown
- We distinguish primal bounds, given by feasible solutions, from dual bounds, given by relaxations or simple properties of the problem
- We define the gap, prove that it bounds the relative error, and compute it on the four reference instances

Primal bounds and dual bounds

- The value of *any* feasible solution bounds the optimal value from one side, and is called a **primal bound**.
- A heuristic algorithm is a method for computing primal bounds.
- A bound on the optimal value from the *other* side, obtained without computing a feasible solution, is called a **dual bound**.
- Dual bounds come from relaxations of the problem, such as the LP relaxation, or from simple properties of the problem.

Upper bounds and lower bounds

- For a minimization problem a primal bound is an **upper bound** UB and a dual bound is a **lower bound** LB; for a maximization problem the roles are exchanged.
- In both cases the optimal value is sandwiched between the two bounds:

$$\begin{cases} \mathrm{LB}(I) \leq z(I) \leq \mathscr{A}(I) & \text{if } \mathscr{P} \text{ is a minimization problem,} \\ \mathscr{A}(I) \leq z(I) \leq \mathrm{UB}(I) & \text{if } \mathscr{P} \text{ is a maximization problem.} \end{cases}$$

The gap

- The **gap** of a heuristic value $\mathscr{A}(I)$ is the relative gap $\frac{\mathrm{UB}-\mathrm{LB}}{\mathrm{LB}}$ defined in the module *Combinatorial Optimization Problems*, «Primal bounds, dual bounds and gap»: the difference of the two bounds divided by the **lower** one.

- With the heuristic value in place of the primal bound it reads

$$gap(I) = \begin{cases} \dfrac{\mathscr{A}(I) - \mathrm{LB}(I)}{\mathrm{LB}(I)} & \text{if } \mathscr{P} \text{ is a minimization problem,} \\ \dfrac{\mathrm{UB}(I) - \mathscr{A}(I)}{\mathscr{A}(I)} & \text{if } \mathscr{P} \text{ is a maximization problem.} \end{cases}$$

The relative error

- The **relative error** of $\mathscr{A}$ on I is

$$\begin{cases} \dfrac{\mathscr{A}(I) - z(I)}{z(I)} & \text{for a minimization problem,} \\ \dfrac{z(I) - \mathscr{A}(I)}{z(I)} & \text{for a maximization problem.} \end{cases}$$

- It cannot be computed when $z(I)$ is unknown, which is the typical situation.

- The gap, instead, uses only computed quantities.

The gap bounds the relative error

How are the gap, which we can compute, and the relative error, which we usually cannot, related?

Proposition 1

For every instance I, the relative error of $\mathscr{A}$ on I is at most $gap(I)$.

Proof (1/2)

For a minimization problem, $\mathrm{LB}(I) \leq z(I)$, hence

$$\frac{\mathscr{A}(I)}{z(I)} \leq \frac{\mathscr{A}(I)}{\mathrm{LB}(I)}$$

and

$$\frac{\mathscr{A}(I) - z(I)}{z(I)} = \frac{\mathscr{A}(I)}{z(I)} - 1$$

$$\leq \frac{\mathscr{A}(I)}{\mathrm{LB}(I)} - 1$$

$$= gap(I).$$

Proof (2/2)

For a maximization problem, $\mathscr{A}(I) \leq z(I) \leq \mathrm{UB}(I)$, hence

$$\frac{z(I) - \mathscr{A}(I)}{z(I)} \leq \frac{z(I) - \mathscr{A}(I)}{\mathscr{A}(I)}$$

$$\leq \frac{\mathrm{UB}(I) - \mathscr{A}(I)}{\mathscr{A}(I)}$$

$$= gap(I).$$

□

The gap is a certificate

- The gap is therefore a **certificate**:
  - if $gap(I) \leq 5\%$, the solution is within 5% of the optimum, even if the optimum is unknown;
  - if $gap(I) = 0$, the solution is optimal.
- We now compute primal bounds, dual bounds and gaps on the four reference instances.

Example: bounds and gaps on the reference instances (1/5)

KP.

- The solution $\{1, 3\}$ has weight $8 \leq 9$ and profit $\mathscr{A}(I) = 16$.
- The LP relaxation, with the items sorted by non-increasing efficiency $(1, 3, 2, 4)$, packs items 1 and 3 and a fraction $\frac{1}{4}$ of item 2, so

$$z(I) \leq 10 + 6 + 7 \cdot \tfrac{1}{4} = 17.75$$

  and, the profits being integer, $\mathrm{UB}(I) = 17$.
- Hence

$$gap(I) = \frac{17 - 16}{16} = 6.25\%,$$

  while the relative error is $\frac{17-16}{17} \approx 5.9\%$.

Example: bounds and gaps on the reference instances (2/5)

BPP.

- The packing patterns $\{1\}$, $\{2\}$, $\{3\}$, $\{4,5\}$, $\{6\}$ are feasible and use $\mathscr{A}(I) = 5$ bins.
- The dual bound is

$$\mathrm{LB}(I) = \left\lceil \frac{21}{7} \right\rceil = 3.$$

- Hence

$$gap(I) = \frac{5-3}{3} \approx 66.7\%.$$

Example: bounds and gaps on the reference instances (3/5)

TSP.

- The tour $(1, 3, 2, 4, 5)$ has length

$$\mathscr{A}(I) = 2 + 4 + 3 + 7 + 9 = 25.$$

- A dual bound is given by the proposition on the two smallest distances of every city, a few slides ahead: $\mathrm{LB}(I) = 17$.

- Hence

$$gap(I) = \frac{25 - 17}{17} \approx 47.1\%,$$

while the relative error is $\frac{25-18}{18} \approx 38.9\%$.

Example: bounds and gaps on the reference instances (4/5)

$P \,||\, C_{\max}$.

- The solution with $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$ has makespan $\mathscr{A}(I) = 11$.
- The dual bound is

$$\mathrm{LB}(I) = \max\{5, \lceil 27/3 \rceil\} = 9.$$

- Hence

$$gap(I) = \frac{11 - 9}{9} \approx 22.2\%.$$

Example: bounds and gaps on the reference instances (5/5)

- On the BPP and $P \,||\, C_{\max}$ the dual bound equals $z(I)$, so the gap equals the relative error.
- On the KP the dual bound also equals $z(I)$, but the gap divides by the smaller value $\mathscr{A}(I)$ and is slightly larger.
- On the TSP the dual bound is weaker, and the gap overestimates the relative error.

A dual bound for the TSP

Where does the dual bound 17 of the TSP come from? From the two shortest edges at every city.

Proposition 2

For every TSP instance I, let a_i and b_i be the two smallest distances from city i to the other cities. Then

$$z(I) \geq \left\lceil \frac{1}{2} \sum_{i \in \mathcal{V}} (a_i + b_i) \right\rceil .$$

Proof

- In every tour each city is the endpoint of exactly two edges, of length at least a_i and b_i.
- Summing the lengths of the two edges incident to each city counts every edge of the tour twice, hence

$$2\,L(\sigma) = \sum_{i \in \mathcal{V}} \big(\text{lengths of the two edges incident to } i\big)$$

$$\geq \sum_{i \in \mathcal{V}} (a_i + b_i)$$

  for every tour σ, and in particular for an optimal one.
- The distances being integer, $z(I)$ is an integer, and the bound can be rounded up.

□

The bound on the TSP reference instance

- The two smallest distances are:
  - $(2, 2)$ for city 1;
  - $(3, 4)$ for city 2;
  - $(2, 4)$ for city 3;
  - $(2, 3)$ for city 4;
  - $(4, 7)$ for city 5.

- Hence

$$\mathrm{LB}(I) = \left\lceil \frac{4 + 7 + 6 + 5 + 11}{2} \right\rceil = \lceil 16.5 \rceil = 17.$$

One side is not enough

A heuristic algorithm gives only one side of the gap.

- Without a dual bound on the other side, nothing can be said about the quality of the solution, however good it may look.

Summary (1/2)

Primal and dual bounds:

- primal bound: the value of a feasible solution, such as $\mathscr{A}(I)$
- dual bound: from a relaxation or a simple property, without a feasible solution
- the optimal value is sandwiched between the two

Gap and relative error:

- the gap divides the difference of the two bounds by the lower one
- the relative error needs $z(I)$; the gap uses only computed quantities, and bounds it

Summary (2/2)

On the reference instances:

- gaps 6.25% (KP), 66.7% (BPP), 47.1% (TSP), 22.2% ($P \,||\, C_{\max}$)
- the TSP dual bound 17 comes from the two smallest distances of every city

The lesson:

- a heuristic gives only one side of the gap
- without a dual bound, nothing can be said about the quality of a solution

Classification of heuristic algorithms

Introduction: Classification of heuristic algorithms

- There are many heuristic algorithms: to find our way among them, we group them into classes
- The classes depend on how the algorithms produce and use solutions
- Each class is studied in detail later in the course

Four classes (1/2)

Heuristic algorithms are classified according to how they produce and use solutions, in four classes.

class	idea
constructive heuristics	build one solution step by step, extending a partial solution with one element at a time; a choice, once made, is never revised
local search	start from a feasible solution and repeatedly replace it by a better solution obtained by a small modification, until no small modification improves it

Four classes (2/2)

class	idea
metaheuristics	general strategies that guide a local search beyond the point where it stops, exploring other regions of the solutions
MIP-based heuristics	use a mixed-integer programming formulation, its relaxation and a MIP solver to guide the search or to explore large sets of solutions

The classes work together

- The classes are not alternatives:
  - a local search needs a starting solution, typically computed by a constructive heuristic;
  - a metaheuristic contains one or more local searches;
  - a MIP-based heuristic may start from, and improve, the solution of any of the others.

Summary

Four classes:

- constructive heuristics: one solution, built step by step, choices never revised
- local search: small improving modifications, until none improves
- metaheuristics: guide a local search beyond the point where it stops
- MIP-based heuristics: a formulation, its relaxation and a MIP solver

Not alternatives:

- the classes are combined: each one can start from, or contain, the others

Solution representation and search space

Introduction: Solution representation and search space

- Before designing a heuristic, we must decide how to store a solution in a data structure
- This choice fixes the search space, and which modifications of a solution are easy to perform
- We fix one representation for each of the four problems, and see how large the search spaces are

Representation and search space

- A heuristic algorithm manipulates solutions, which must be stored in a data structure: the **representation** of the solutions.
- The set of all the objects that the representation can describe is the **search space**.
- The representation is a design choice, and it determines which modifications of a solution are easy to perform.
- For the four problems we use the representations on the next slide.

Representations of the four problems

problem	representation	size of the search space	feasible?
KP	binary vector $x \in \{0,1\}^n$	2^n	not always: the capacity can be exceeded
BPP	vector *bin*, with bin_j the bin of item j	at most n^n	not always: a bin can be overfull
TSP	permutation σ of the cities	$\frac{(n-1)!}{2}$ distinct tours	always
$P \mid\mid C_{\max}$	vector *mach*, with $mach_j$ the machine of job j	m^n	always

Infeasible objects and large search spaces

- When the representation can describe infeasible objects, a heuristic must either restrict itself to feasible ones or accept infeasible ones and penalize them.
- The search spaces grow very quickly: a TSP with $n = 20$ cities has

$$\frac{19!}{2} \approx 6 \cdot 10^{16}$$

  distinct tours.
- A heuristic explores only a tiny part of the search space, and the question is *which* part.

Summary

Representation and search space:

- the representation is the data structure that stores a solution: a design choice
- the search space is the set of all the objects it can describe

The four problems:

- KP and BPP: the representation can describe infeasible objects; TSP and $P \,||\, C_{\max}$: always feasible
- the search spaces grow very quickly: a heuristic explores only a tiny part of them

Neighbourhoods and local optima

Introduction: Neighbourhoods and local optima

- Local search and metaheuristics move from a solution to a "close" one: in this section we make this idea precise
- We define neighbourhoods, moves and local optima, and compare two neighbourhoods on the KP reference instance
- We see why, for NP-hard problems, a neighbourhood is always a compromise between size and quality

Feasible solutions and their values

- Let $\mathscr{F}(I)$ be the set of the feasible solutions of the instance I.
- Let $f(S)$ be the value of the solution $S \in \mathscr{F}(I)$.
- Then $z(I)$ is the minimum (or maximum) of f over $\mathscr{F}(I)$.

Neighbourhood

Definition 2: Neighbourhood

*A **neighbourhood** is a function N that associates with every feasible solution $S \in \mathscr{F}(I)$ a set $N(S) \subseteq \mathscr{F}(I)$ of feasible solutions, the **neighbours** of S. A **move** is the transformation of S into one of its neighbours.*

- A neighbourhood is usually defined by a family of small modifications of the representation: change one component of the vector x, move one job to another machine, replace two edges of a tour.

Local optimum

Definition 3: Local optimum

*A feasible solution $S \in \mathscr{F}(I)$ is a **local optimum** with respect to the neighbourhood N if no neighbour of S is better than S, i.e., for all $S' \in N(S)$:*

$$\begin{cases} f(S) \leq f(S') & \text{if } \mathscr{P} \text{ is a minimization problem,} \\ f(S) \geq f(S') & \text{if } \mathscr{P} \text{ is a maximization problem.} \end{cases}$$

*An optimal solution is also called a **global optimum**.*

Local and global optima

- A global optimum is a local optimum for every neighbourhood.
- A local optimum need not be a global optimum.
- Moreover, **being a local optimum depends on the neighbourhood**: the same solution can be a local optimum for one neighbourhood and not for another.
- The next example shows it on the KP reference instance.

Example: two neighbourhoods for the KP (1/2)

On the KP reference instance ($c = 9$), consider the solution $x = (1, 0, 1, 0)$, i.e., the items $\{1, 3\}$, of weight 8 and profit 16.

The **flip** neighbourhood changes one component of x, i.e., adds or removes one item:

neighbour	weight	profit	feasible
$(0, 0, 1, 0)$	3	6	yes
$(1, 1, 1, 0)$	12	—	no
$(1, 0, 0, 0)$	5	10	yes
$(1, 0, 1, 1)$	11	—	no

The feasible neighbours have profit 6 and 10, smaller than 16: x is a **local optimum** with respect to the flip neighbourhood.

Example: two neighbourhoods for the KP (2/2)

The **swap** neighbourhood removes one selected item and adds one unselected item:

neighbour	weight	profit	feasible
$(0,1,1,0)$: out 1, in 2	7	13	yes
$(0,0,1,1)$: out 1, in 4	6	10	yes
$(1,1,0,0)$: out 3, in 2	9	17	yes
$(1,0,0,1)$: out 3, in 4	8	14	yes

The neighbour $(1,1,0,0)$ has profit $17 > 16$: x is **not** a local optimum with respect to the swap neighbourhood, and the move leads to the global optimum.

Exact neighbourhood

Definition 4: Exact neighbourhood

*A neighbourhood N is **exact** if every local optimum with respect to N is a global optimum.*

- Exact neighbourhoods are rare, and they exist for problems that can be solved in polynomial time:
  - the neighbourhood of the simplex method, which moves between adjacent vertices of the polyhedron, is exact for linear programming;
  - the neighbourhood that exchanges one edge in and one edge out is exact for the minimum spanning tree problem.

Size versus quality

- For NP-hard problems, small neighbourhoods are not exact.
- The neighbourhoods that are exact, such as the one containing all the feasible solutions, are too large to be explored.
- The design of a neighbourhood is a compromise between:
  - its **size**, which determines the time needed to explore it;
  - the **quality** of its local optima.

Summary

Neighbourhoods:

- a neighbourhood $N(S)$ is a set of feasible solutions close to S; a move goes to one of them
- usually defined by small modifications of the representation

Local optima:

- no neighbour is better; being a local optimum depends on the neighbourhood (flip and swap on the KP)
- exact neighbourhoods are rare: size versus quality of the local optima

Intensification and diversification

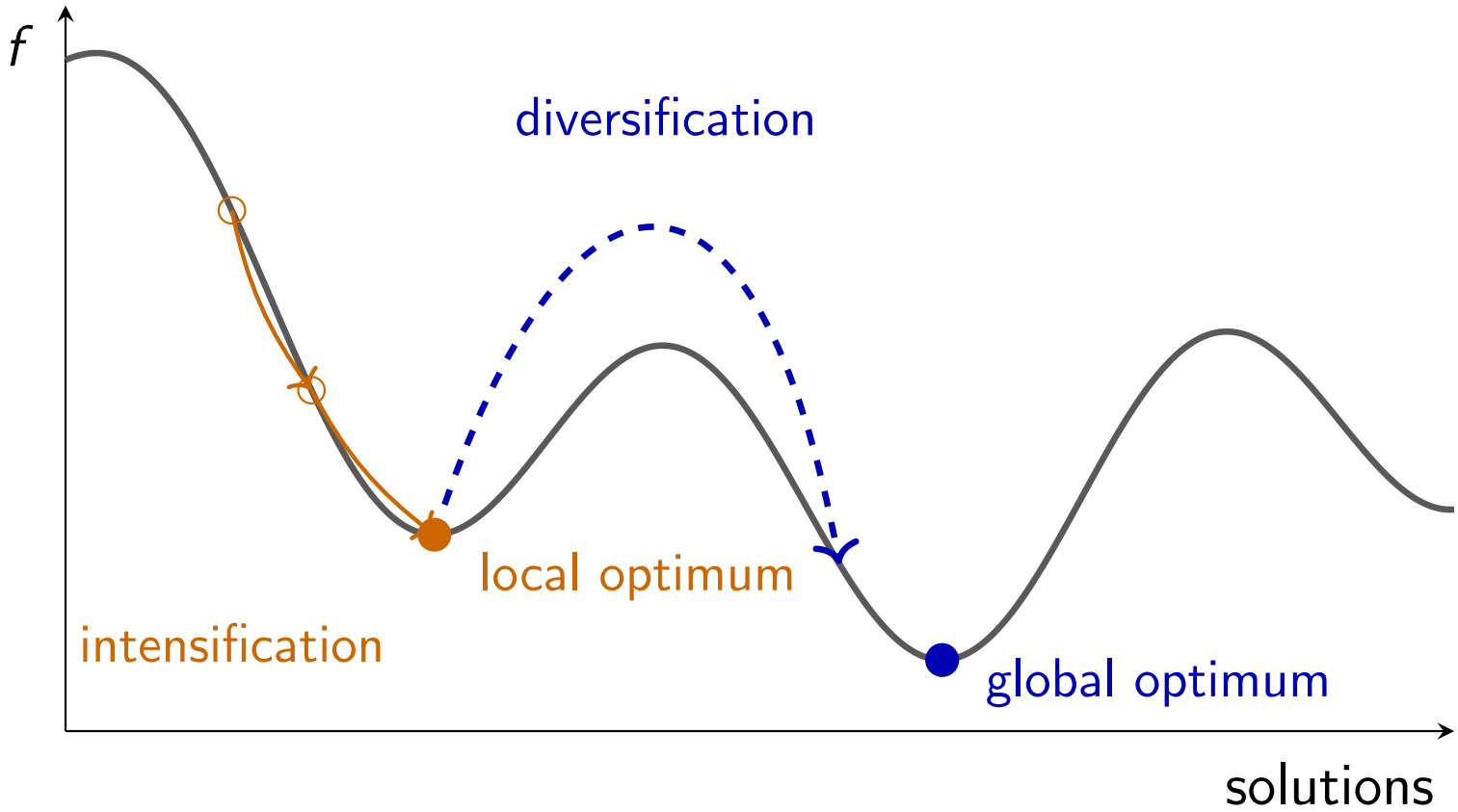
Introduction: Intensification and diversification

- A picture helps to see how a heuristic moves among the solutions: the landscape of their values
- On it we see the two forces of every search: going down into a valley, and leaving it to find a deeper one
- Balancing them is the key idea of the metaheuristics, later in the course

The landscape

- The values of the solutions, seen as a function on the search space, form the **landscape** explored by a heuristic.
- For a minimization problem the local optima are the bottoms of the valleys.
- The global optimum is the bottom of the deepest valley.

The landscape of a minimization problem



Intensification and diversification

- **Intensification** is the search for better solutions in the region where good solutions have already been found: moving downhill, toward the bottom of the current valley.
- Local search is pure intensification, and stops at the first local optimum it reaches.
- **Diversification** is the move toward regions of the search space that have not been explored yet: leaving the current valley, possibly through worse solutions, to reach another one.

Balancing the two

- Too much intensification traps the search in the first valley.
- Too much diversification makes it wander without ever exploring a valley to its bottom.
- Every metaheuristic is a way of balancing the two.

Summary

The landscape:

- the values of the solutions over the search space
- local optima at the bottoms of the valleys, the global optimum at the bottom of the deepest one

Two forces:

- intensification: down into the current valley (local search is pure intensification)
- diversification: out of the valley, toward unexplored regions
- every metaheuristic balances the two

How to evaluate a heuristic algorithm

Introduction: How to evaluate a heuristic algorithm

- Most heuristics have no proved guarantee: how do we decide whether one is good?
- In this section we see the two criteria, quality and running time, and how to measure them
- We see why one run on one instance proves nothing, and how to deal with random choices

Quality and running time

- A heuristic algorithm is evaluated by the **quality** of its solutions and by its **running time**, which are conflicting goals.
- **Quality** is measured by the gap with respect to the best available dual bound, or by the relative error when the optimal value is known, as on the reference instances of this chapter.

Worst-case and experimental analysis

- **Worst-case analysis**, as for approximation algorithms, gives guarantees valid for all the instances, but they are often far from the typical behaviour.
- **Experimental analysis** measures the gaps and the running times on a set of benchmark instances.
- The benchmark must be large and varied enough to avoid conclusions that hold only for a few instances.

A single instance proves nothing

- A single instance proves nothing: on the TSP reference instance the same rule, applied from different starting cities, gives tours of length between 18 and 25.
- Comparisons between heuristics must use the same instances and report the distribution of the results, not only the best one.

Random choices and seeds

- For heuristics that use random choices, the results depend on the random numbers.
- Each instance must be solved several times, with different **seeds** of the random number generator.
- The report gives the average and the spread of the results.

What makes a heuristic good

A heuristic algorithm is good if,
on the instances it is meant for,
it finds solutions with a small gap
in the time available, consistently.

Summary

Two conflicting criteria:

- quality (gap, or relative error when $z(I)$ is known) and running time
- worst-case analysis gives guarantees; experimental analysis measures the typical behaviour

Sound experiments:

- many varied instances, the same for all the heuristics, the whole distribution of the results
- random methods: several seeds, with the average and the spread

Typical mistakes

Introduction: Typical mistakes

- We collect the mistakes that are made most often with heuristic algorithms and their bounds
- Each of them comes from a definition or a result of this chapter
- Check them before solving the exercises

Typical mistakes (1/3)

Observation 1: Judging a heuristic from one run or one instance

The same rule, from different starting cities, gives tours of length between 18 *and* 25 *on the TSP reference instance.*

Observation 2: Confusing the gap with the relative error

The gap is measured against a dual bound, so it is never smaller than the relative error; in a minimization problem they coincide only when the dual bound equals $z(I)$.

Typical mistakes (2/3)

Observation 3: Using the wrong denominator

The gap always divides by the lower bound: for a maximization problem it is

$$\frac{\mathrm{UB}(I) - \mathscr{A}(I)}{\mathscr{A}(I)},$$

for a minimization problem

$$\frac{\mathscr{A}(I) - \mathrm{LB}(I)}{\mathrm{LB}(I)}.$$

Typical mistakes (3/3)

Observation 4: Forgetting to round the dual bound

With integer data the bound 17.75 *of the KP is rounded down to* 17*, and the bound* 16.5 *of the TSP is rounded up to* 17*.*

Observation 5: Trusting a good-looking solution

Without a dual bound on the other side, nothing can be said about the quality of a solution.

Summary

Evaluating a heuristic:

- never from one run or one instance
- without a dual bound, a good-looking solution proves nothing

Computing the gap:

- the gap is not the relative error: it is never smaller
- divide by the lower bound, and round the dual bound with integer data

In summary

Introduction: In summary

- We collect the key ideas of the chapter on heuristic algorithms
- The dashboard gathers the bounds found on the four reference instances

Summary (1/2)

- A heuristic algorithm returns a feasible solution without guaranteeing its optimality: its value is a *primal* bound.
- The gap compares the heuristic value with the best dual bound, and it is an upper bound on the relative error.
- Heuristic algorithms are constructive, local search, metaheuristic or MIP-based; a neighbourhood defines the local optima, and the search balances intensification and diversification.
- Quality is assessed on many instances and, for random methods, on many seeds.

Summary (2/2)

Dashboard of the reference instances. Heuristic solutions of this chapter and their dual bounds:

problem	primal bound	dual bound	gap	$z(I)$
KP (max)	16	$\mathrm{UB}(I) = 17$	6.25%	17
BPP (min)	5	$\mathrm{LB}(I) = 3$	66.7%	3
TSP (min)	25	$\mathrm{LB}(I) = 17$	47.1%	18
$P \mid\mid C_{\max}$ (min)	11	$\mathrm{LB}(I) = 9$	22.2%	9