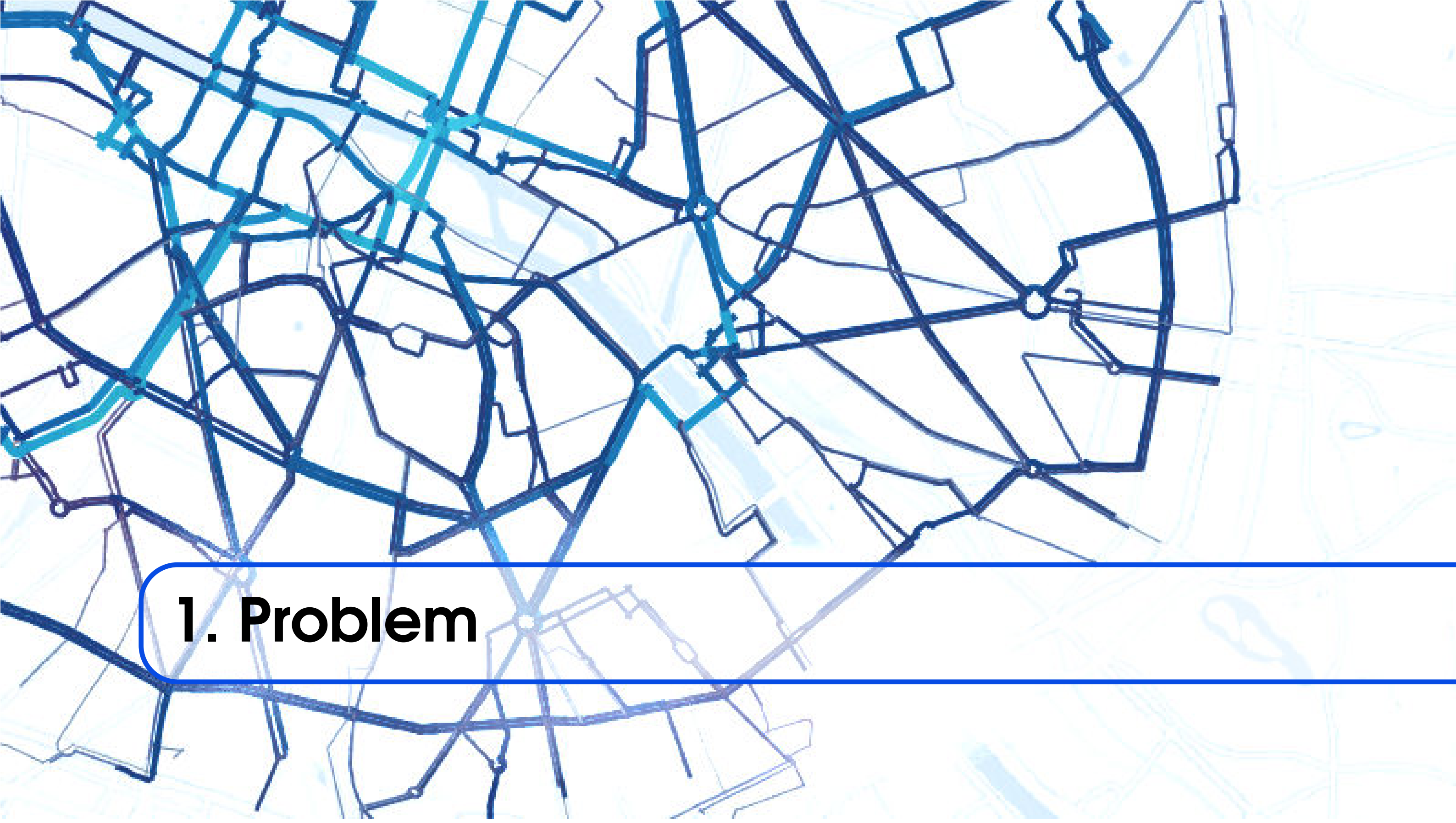
Operations Research project

Mathematics Department Efrei Paris

Spring Semester 2024 - S6

L3 INT



1. Problem

1.1 Introduction

Transportation problems are closely linked to social, economic and ecological issues. Through the algorithms we have seen in class, we try to understand how to lower the costs of transport in a network. The nature of these costs may be human, monetary or environmental.

This project involves writing a program to solve a transportation problem. Once you've built the code, we'll ask you to test it on the problems in the appendices, providing us with execution traces. Finally, you'll have to use it to analyze the complexity generated.

1.2 Solving transportation problems

We ask you to code the solution to the following problem : let n suppliers have provisions, called $(P_i)_{i\in[\![1;n]\!]}$ and m customers have orders, called $(C_j)_{j\in[\![1;m]\!]}$. Each unit transport of an object between supplier i and customer j costs $a_{i,j}$, which forms the matrix $A = (a_{i,j})_{(i,j)\in[\![1;n]\!]\times[\![1;m]\!]}$.

The goal is to find the best way of transporting objects from suppliers to customers that minimizes the total cost of transport. In other words, we want to find the numbers $(b_{i,j})_{(i,j)\in[\![1;n]\!]\times[\![1;m]\!]}$ of objects transported from each supplier i to each customer j such that $\sum_{i=1}^{n}\sum_{j=1}^{m} a_{i,j}\times b_{i,j}$ be minimal, under the constraint of provisions $\sum_{j=1}^{m} b_{i,j} = P_i$ and orders $\sum_{i=1}^{n} b_{i,j} = C_j$. This is, of course, the problem studied in class.

For the purposes of this project, we'll restrict our program writing to the balanced case, i.e. such that $\sum_{i=1}^{n} P_i = \sum_{j=1}^{m} C_j$. You will also work with the programming language of your choice : C, C++, Python, Java.

1.3 The constraint table

The first step is to create a .txt file for each transport problem, organized as follows :

n	m			
$a_{1,1}$	$a_{1,2}$	...	$a_{1,m}$	Provision P_1
$a_{2,1}$	$a_{2,2}$	...	$a_{2,m}$	Provision P_2
$\vdots$			$\vdots$	$\vdots$
$a_{n,1}$	$a_{n,2}$	...	$a_{n,m}$	Provision P_n
Order C_1	Order C_2	...	Order C_m	

	C_1	C_2	C_3	Provisions P_i
P_1	30	20	20	450
P_2	10	50	20	250
P_3	50	40	30	250
P_4	30	20	30	450
Orders C_j	500	600	300	

4	3		
30	20	20	450
10	50	20	250
50	40	30	250
30	20	30	450
500	600	300	

FIGURE 1.1 – An example of a transportation problem and its .txt table. Transportation prices per unit are in bleu.

In the appendices, you'll find the 12 tables of the 12 transportation problems you're asked to solve with your program. You'll need to edit the 12 tables in 12 different .txt files in the same way. These files should be attached to your report.

2. Code

2.1 The functions

You will have to implement the following functions :

1. Read data from text file (.txt) and store in memory.
2. Display of the following tables :
   - ⋆ Cost matrix
   - ⋆ Transportation proposal
   - ⋆ Potential costs table
   - ⋆ Marginal costs table

   Please note : the display function must be absolutely accurate. Any table with columns that shift will be severely penalized. Table legibility is fundamental.
3. Algorithm for setting the initial proposal : North-West.
4. Algorithm for setting the initial proposal : Balas-Hammer.
   - ⋆ Calculation of penalties.
   - ⋆ Display of row(s) (or columns) with the maximum penalty.
   - ⋆ Choice of edge to fill.
5. Total cost calculation for a given transport proposal.
6. Solving algorithm : the stepping-stone method with potential.
   - ⋆ Test whether the proposition is acyclic : we'll use a Breadth-first algorithm. During the

algorithm run, as the vertices are discovered, we check that we're returning to a previously visited vertex and that this vertex isn't the parent of the current vertex ; if it is, then a cycle exists. The cycle is then displayed.

⋆ Transportation maximization if a cycle has been detected. The conditions for each box are displayed. Then we display the deleted edge (possibly several) at the end of maximization.

⋆ Test whether the proposition is connected : we'll use a Breadth-first algorithm. If it is not connected : display of all connected sub-graphs.

⋆ Modification of the graph if it is unconnected, until a non-degenerate proposition is obtained.

⋆ Calculation and display of potentials per vertex.

⋆ Display of both potential costs table and marginal costs table. Possible detection of the best improving edge.

⋆ Add this improving edge to the transport proposal, if it has been detected.

Please note : Each function must be highlighted and clearly explained orally, using pseudo-code.

You must have the following functions working :

⋆ Read data and store it in memory.

⋆ North-West and Balas-Hammer algorithms.

⋆ Calculation of the total cost for a given transport proposal.

⋆ Test whether the proposition is acyclic using a Breadth-first algorithm.

⋆ Transport maximization on a detected cycle.

⋆ Test whether the proposition is connected using a Breadth-first algorithm.

⋆ Display all tables.

2.2 Overall structure

The overall structure of your program is illustrated by the following pseudo-code :

Start

While the user decides to test a *transportation problem*, do :

Choice of the problem number to be processed.

Read the table of constraints from a file and store it in memory

Create the corresponding matrice representing this table and display it

Ask the user to choose the algorithm to fix the initial proposal and execute it.

Display the elements mentioned above when running the two algorithms.

Run the stepping-stone method with potential, displaying at each iteration :

⋆ Displays the transport proposal and the total transport cost.

⋆ Test to know if the transport proposal is degenerate.

⋆ Modification of the transport graph to obtain a tree, in the cyclic or non connected.

⋆ Potentials calculation and display.

⋆ Table display : potential costs and marginal costs.

⋆ If not optimal :

Displays the edge to be added.

Transport maximization on the formed cycle and a new iteration.

⋆ Else exit the loop

⋆ End if

Display the minimal transportation proposal and its cost.

Suggest to the user that he/she should change transportation problem

End while

End

2.3 Possible improvements

Once the algorithm has been sufficiently tested, we suggest the following improvements :

1. When "Modification of the transport graph to obtain a tree, in the cyclic or non connected case", we must first detect whether the graph has a cycle. After maximizing the transport proposal on this cycle, other cycles may remain. In this case, the "Cycle detection" and "Maximization on cycle" functions must be repeated until an acyclic proposal is obtained. Only then will we carry out the connexity test, where we will, if necessary, complete the graph with edges ranked according to increasing costs, until we obtain a connected and acyclic proposition.
2. When executing the function "Transportation maximization if a cycle has been detected", it may happen that $\delta = 0$, i.e. no change is made to the cycle. You can then detect this special case. We'll do like this : we'll keep the detected improving edge with the marginal costs table (if this is the general loop) and remove all the last edges added during the last connected test, in the

same iteration. The function "Modification of the transport graph to obtain a tree, in the non connected case" that follows will propose a different set of edges.

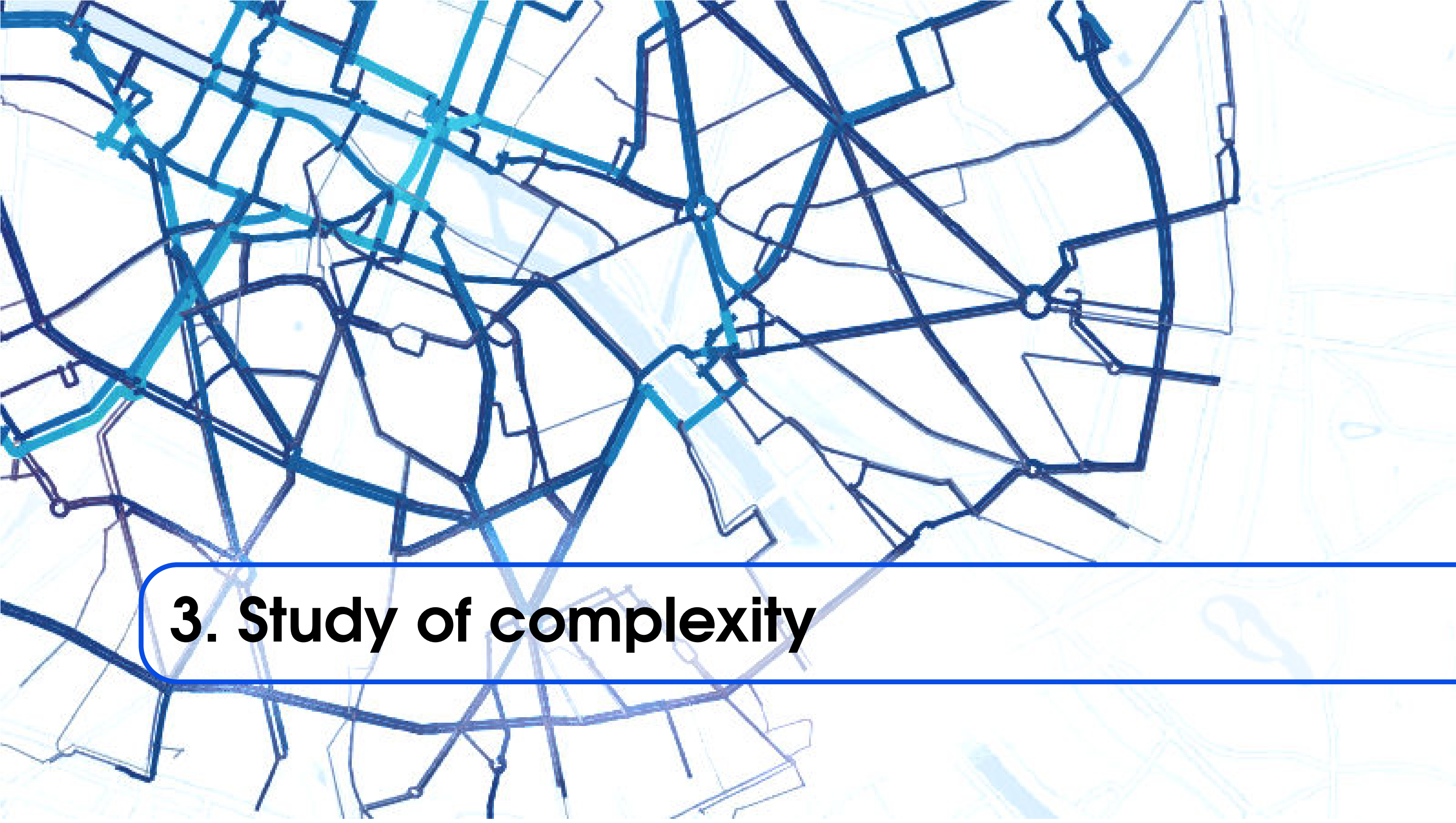
2.4 Execution traces

Execution traces for the 12 graphs provided in the appendices are requested in the rendering. An execution trace is what is displayed by the console. They cannot be replaced by screen copies.

You have to run your program with the 12 problems in both cases : initial proposal North-West NW, then Balas-Hammer BH. The you'll display stepping-stone algorithm with all potentials, tables and information previously requested.

Files will be stored as follows :

- ⋆ Group B - Team 4 - Problem 5 - North-West : "B4-trace5-nw.txt"
- ⋆ Group D - Team 2 - Problem 12 - Balas-Hammer : "D2-trace12-bh.txt"



3. Study of complexity

3.1 Introduction

This part must be done by all teams, as soon as some of your functions are up and running. We now propose to study the *complexity* of algorithms in this project.

First of all, what is the complexity of an algorithm ? It's the evaluation of the resources required to run an algorithm (essentially the amount of memory required) and the calculation time to be expected. These two notions depend on numerous hardware parameters that are that goes beyond the Algorithmic : we cannot assign an absolute value either to the amount of memory required or to the execution time of a given algorithm. However, it is often possible to evaluate the *rder of magnitude* of these two quantities in order to identify the most efficient algorithm within a set of algorithms solving the same problem.

This is what we propose to do here, by comparing the transport proposals derived from the North-West, Balas-Hammer algorithms when solving the stepping-stone method with potential.

3.2 A short view of théory

Most algorithms have an execution time that depends not only on the size of the input data, but also on the data itself. In this case, there are several types of complexity :

Définition 3.1 (worst-case complexity)
Worst-case complexity is a majorant of the possible execution time for all possible inputs of the same size. It is generally expressed in O notation.

Définition 3.2 (best-case complexity)
Best-case complexity is a minority of the possible execution time for all possible inputs of the same size. It is generally expressed using the notation Ω.
However, this notion is rarely used, as it is often irrelevant to worst-case and average complexities.

Définition 3.3 (average complexity)
Average complexity is an evaluation of the average execution time for all possible inputs of the same size, assumed to be equiprobable.

Définition 3.4 (spatial complexity)
Spatial complexity evaluates the consumption of memory space. The principle is the same, except that here the aim is to evaluate the order of magnitude of the memory volume used : it's not a question of evaluating precisely how many bytes are consumed by an algorithm, but of specifying its growth rate as a function of the size n of the input.

3.3 Study

In this project, we'll be analyzing **worst-case complexity**. To do this, we ask you to generate random transportation problems. Then look at the execution times of the algorithms.

3.3.1 Inputs of transportation problems

To simplify the problem, you will work with transport problems of size $n = m$. The matrix $A = (a_{i,j})_{(i,j)\in[\![1;n]\!]^2}$ is square.

In order to generate a sampling of all possible inputs of the same size n, you will write a function to edit random transport problems. This will be done as follows :

1. An integer random number between 1 and 100 (inclusive) is generated for each $a_{i,j}$.
2. An integer random number between 1 et 100 (inclusive) is generated for each $(temp_{i,j})_{i,j}$ of an $n \times n$ size matrix. You will thus fill in the n values of provisions $(P_i)_{i\in[\![1;n]\!]}$ and orders $(C_j)_{j\in[\![1;n]\!]}$ in the following ways :
$$P_i = \sum_{j=1}^{n} temp_{i,j} \text{ and } C_j = \sum_{i=1}^{n} temp_{i,j}$$

3.3.2 Measuring time

Once the problem has been generated, i.e. once the P_i, C_j and $a_{i,j}$ have been set, we need to store the time value for each portion of code we're interested in. In Python, for example, we simply use the function time.clock(), which returns the CPU time in seconds, and store this value. The difference between 2 values will give the execution time of the framed code portion.

With this n sized problem generated, you'll have to measure the execution time of :

1. the North-West algorithm. We'll call this time $\theta_{NW}(n)$,
2. the Balas-Hammer algorithm. We'll call this time $\theta_{BH}(n)$,
3. the steppin-stone algorithm with the proposal from Northwest. We'll call this time $t_{NW}(n)$,
4. the steppin-stone algorithm with the proposal from Balas-Hammer. We'll call this time $t_{BH}(n)$,

3.3.3 Scatter plot

For each value of n, you will run your program 100 times with different random values for the transportation problem. So, for a fixed n, you'll obtain 100 values of $\theta_{NO}(n)$, for example.

n values to be tested	10	40	10^2	4.10^2	10^3	4.10^3	10^4

Please note : to perform this task, you will be working on a single processor machine.
Instructions will be executed one after the other, without simultaneous operations.
So don't use your machine for anything else during the entire runtime.

Once the values have been stored, you can plot the scatter plots (the 100 values for the same abscissa) as a function of n :

- ⋆ $\theta_{NO}(n)$.
- ⋆ $\theta_{BH}(n)$.
- ⋆ $t_{NO}(n)$.

* $t_{BH}(n)$.
* $(\theta_{NO} + t_{NO})(n)$.
* $(\theta_{BH} + t_{BH})(n)$.

3.3.4 Worst-case complexity per algorithm

Worst-case complexity is assumed to be the upper envelope of the scatter plot. For each value of n, determine this maximum value across the 100 realizations for a fixed n. Then plot this maximum value as a function of n.

For θ_{NO}, θ_{BH}, t_{NO}, t_{BH}, $(\theta_{NO} + t_{NO})$ and $(\theta_{BH} + t_{BH})$ identify the type of worst-case complexity using table 3.1 :

O($log(n)$)	logarithmic
O(n)	linear
O($nlog(n)$)	quasi-linear
O(n^2)	quadratic
O(n^k) (k > 2)	polynomial
O(k^n) (k > 1)	exponential

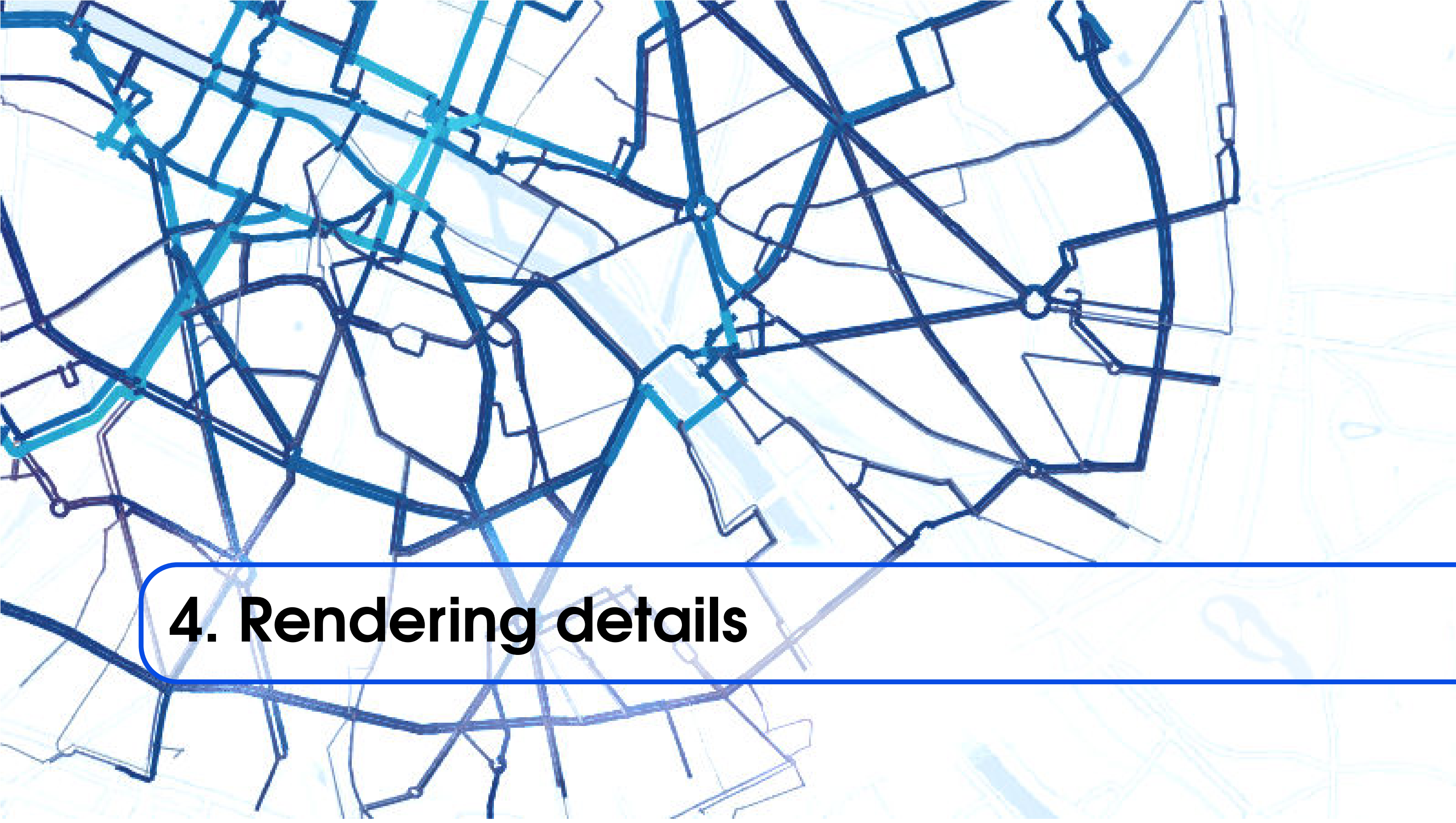
FIGURE 3.1 – Usual qualifier for complexity.

3.3.5 Worst-case complexity comparison

Now let's compare the two algorithms solving <u>the same</u> problem for n fixed by plotting :

$$\frac{t_{NO} + \theta_{NO}}{t_{BH} + \theta_{BH}}(n)$$

Then plot the maximum value found for each value of n and discuss the results.



4. Rendering details

4.1 Rendering

You must submit your project on Moodle by Saturday, May 4 at 11h59 p.m. No additional time will be accepted.

In the repository, you'll give all your programs, the 12 execution traces and the 12 .txt files files of the transportation problems to be solved. Grading will take into account the quality of the algorithms and traces submitted. All teams are required to submit a complexity report of no more than 5 pages.

The rendering file will be titled as follows : for group B and team 4 : "B4".

4.2 Oral presentation

For the oral presentation, slides are expected. Pedagogy and clarity are the aim of this 10 min presentation. We will not accept any code on the slides : we want pseudo-code. Slides must not contain any hand-written or photographed paper.

In the oral presentation, you will be asked to summarize the most important points of each part. A slide summarizing the work you've done, i.e. the functions you've succeeded to do and those you haven't, is expected. Please note : if you exceed the 10-minute presentation time, your teacher will stop you.

At the end of the oral, you'll have to answer some questions during 20 min : each student will be questioned individually on the project or on a point from the course. The code must be clearly understood by all team members.

Good luck with this project,

The Operations Research teaching team.

4.3 Appendices : the transportation proposals to be tested

Transport unit prices are written in blue.

1	C_1	C_2	Provisions
P_1	30	20	100
P_2	10	50	100
Orders	100	100	

2	C_1	C_2	Provisions
P_1	10	20	100
P_2	30	10	100
Orders	100	100	

3	C_1	C_2	Provisions
P_1	30	20	600
P_2	10	50	500
Orders	100	1000	

4	C_1	C_2	Provisions
P_1	30	1	600
P_2	1	30	500
Orders	100	1000	

5	C_1	C_2	C_3	Provisions
P_1	5	7	8	25
P_2	6	8	5	25
P_3	6	7	7	25
Orders	35	20	20	

6	C_1	C_2	C_3	C_4	Provisions
P_1	11	12	10	10	60
P_2	17	16	15	18	30
P_3	19	21	20	22	90
Orders	50	75	30	25	

7	C_1	C_2	Provisions
P_1	50	20	100
P_2	10	50	200
P_3	50	40	100
P_4	45	35	200
Orders	300	300	

8	C_1	C_2	Provisions
P_1	50	20	100
P_2	10	50	200
P_3	55	40	100
P_4	35	45	200
P_5	12	8	200
Orders	300	500	

9	C_1	C_2	C_3	P
P_1	30	20	15	100
P_2	10	50	2	100
P_3	9	10	30	100
P_4	6	2	29	100
P_5	50	40	3	100
P_6	5	38	27	100
P_7	50	4	22	100
C	400	200	100	

10	C_1	C_2	C_3	C_4	C_5	C_6	C_7	P
P_1	300	20	15	16	17	18	20	500
P_2	1	50	24	30	22	27	19	500
P_3	50	40	30	3	25	26	3	2500
C	500	500	500	500	500	500	500	

11	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	P
P_1	1	2	3	4	5	6	7	8	9	10	10
P_2	11	12	13	14	15	16	17	18	19	20	20
P_3	21	22	23	24	25	26	27	28	29	30	30
P_4	31	32	33	34	35	36	37	38	39	40	40
P_5	41	41	43	44	45	46	47	48	49	50	50
P_6	51	52	53	54	55	56	57	58	59	60	60
P_7	61	62	63	64	65	66	67	68	69	70	70
P_8	71	72	73	74	75	76	77	78	79	80	80
P_9	81	82	83	84	85	86	87	88	89	90	90
P_{10}	91	92	93	94	95	96	97	98	99	100	100
P_{11}	101	102	103	104	105	106	107	108	109	110	110
P_{12}	111	112	113	114	115	116	117	118	119	120	120
P_{13}	121	122	123	124	125	126	127	128	129	130	130
P_{14}	131	132	133	134	135	136	137	138	139	140	140
P_{15}	141	142	143	144	145	146	147	148	149	150	150
P_{16}	151	152	153	154	155	156	157	158	159	160	160
P_{17}	161	162	163	164	165	166	167	168	169	170	170
P_{18}	171	172	173	174	175	176	177	178	179	180	180
P_{19}	181	182	183	184	185	186	187	188	189	190	190
P_{20}	191	192	193	194	195	196	197	198	199	200	200
C	120	140	160	180	200	220	240	260	280	300	

12	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	P
P_1	186	185	184	183	182	181	180	179	178	177	176	175	174	173	172	171	160
P_2	166	165	164	163	162	161	160	159	158	157	156	155	154	153	152	151	160
P_3	156	155	154	153	152	151	150	149	148	147	146	145	144	143	142	141	160
P_4	136	135	134	133	132	131	130	129	128	127	126	125	124	123	122	121	160
P_5	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	160
P_6	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	160
P_7	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	160
P_8	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	160
P_9	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	160
P_{10}	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	160
C	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	