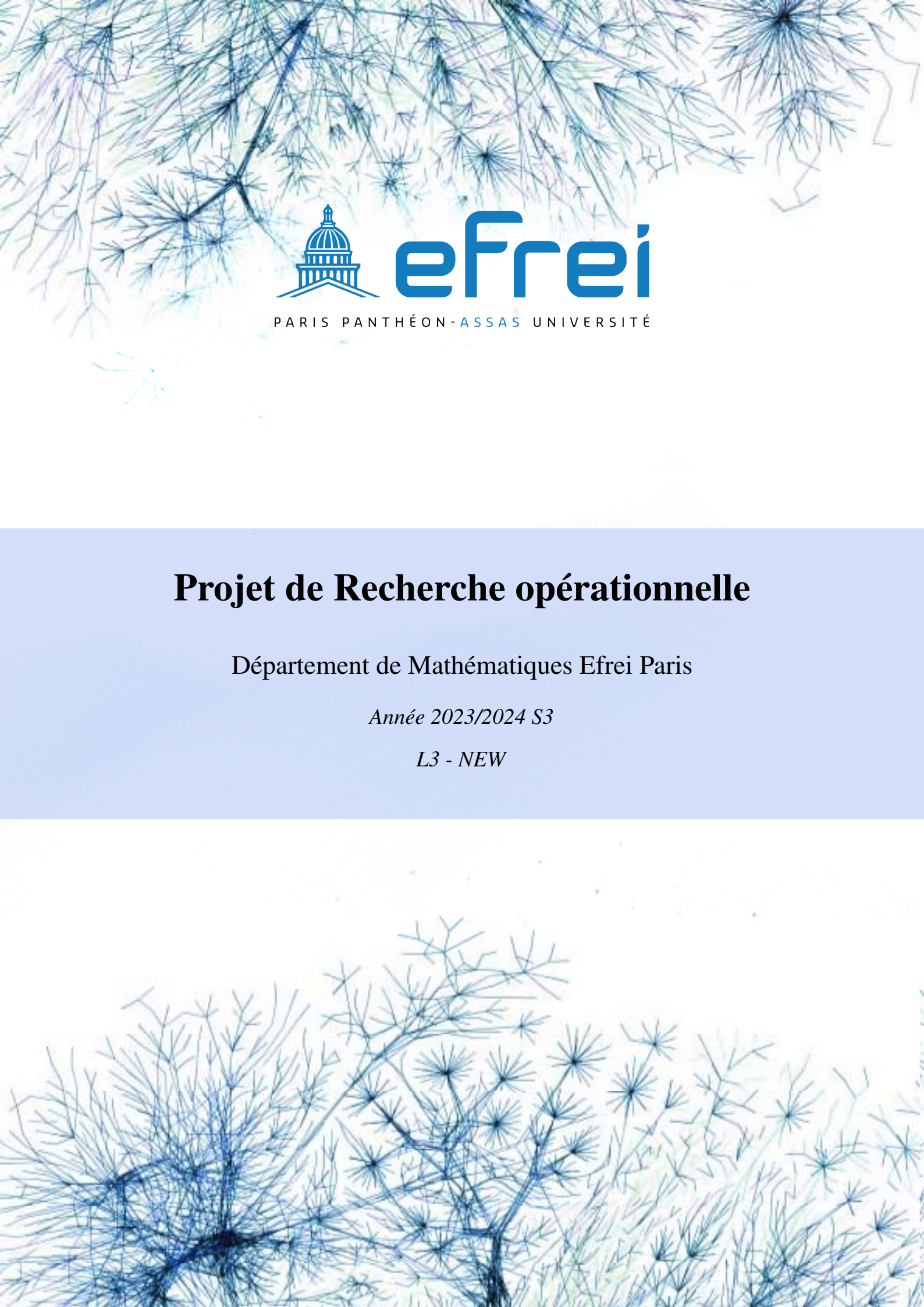
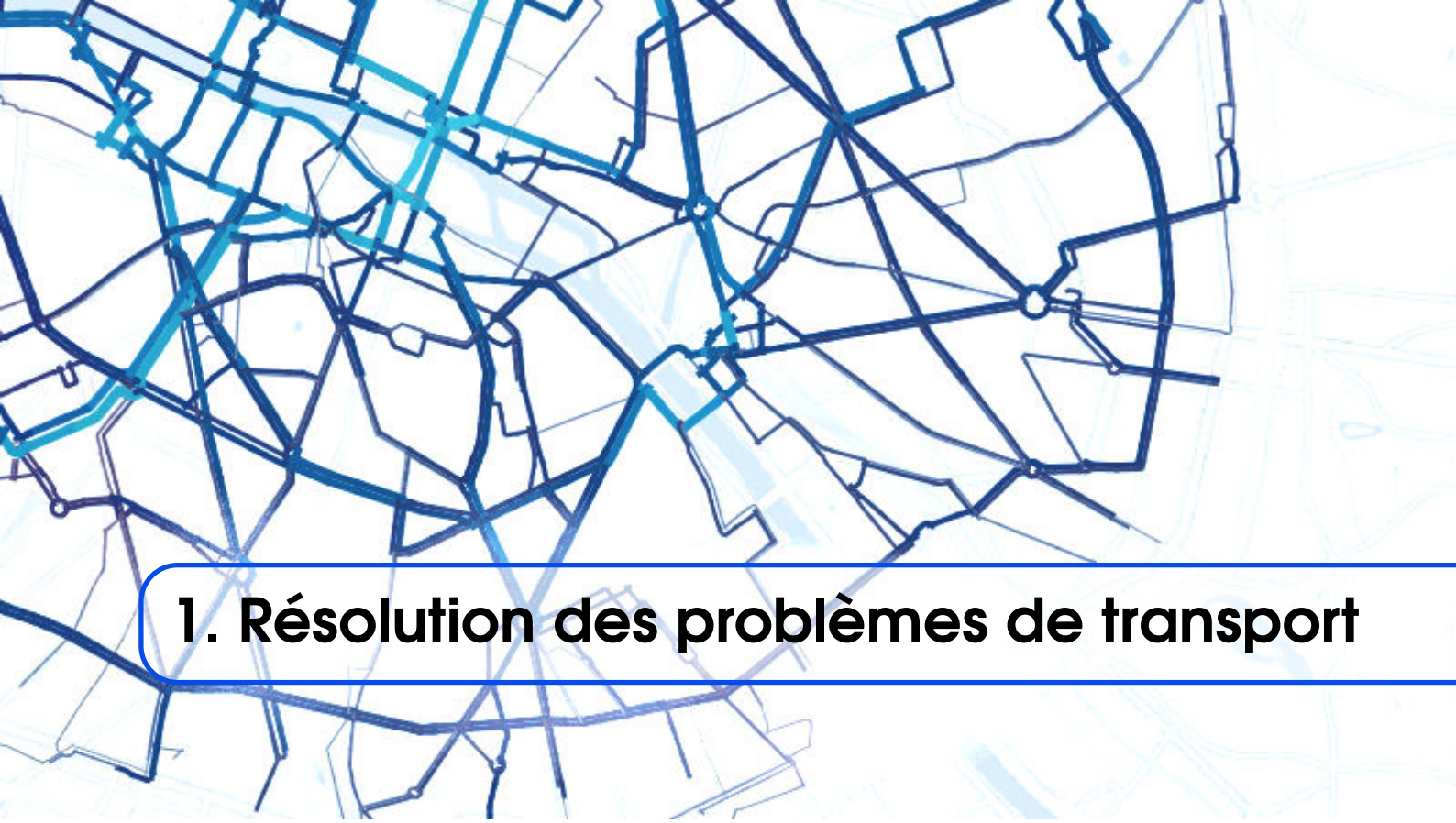


Projet de Recherche opérationnelle

Département de Mathématiques Efrei Paris

Année 2023/2024 S3

L3 - NEW



1. Résolution des problèmes de transport

1.1 Introduction

La question des problèmes de transport est intimement liée aux questions sociales, économiques ou écologiques. A travers les algorithmes vus en cours, nous cherchons à comprendre comment abaisser les coûts de transports dans un réseau. La nature de ces coûts peut être humaine, monétaire ou liée à l'environnement.

Dans ce projet, il est question de la rédaction d'un programme qui résout un problème de transport. Une fois le code construit, nous vous demanderons de le tester sur les problèmes qui se trouvent en annexes et dont il faudra nous fournir les traces d'exécution. Enfin, si et seulement si votre programme fonctionne parfaitement, vous pourrez l'utiliser afin d'analyser la complexité générée.

1.2 Le code

1.2.1 Le problème à résoudre

Nous vous demandons de coder la résolution du problème suivant : soient n fournisseurs ayant des provisions, appelées $(P_i)_{i\in[\![1;n]\!]}$ et m clients ayant fait des commandes, appelées $(C_j)_{j\in[\![1;m]\!]}$. Chaque transport unitaire d'un objet entre le fournisseur i et le client j coûte $a_{i,j}$, ce qui forme la matrice $A = (a_{i,j})_{(i,j)\in[\![1;n]\!]\times[\![1;m]\!]}$.

L'objectif est de trouver la meilleure façon de transporter les objets des fournisseurs vers les clients qui minimise le coût total du transport. C'est à dire que l'on cherche à trouver les nombres $(b_{i,j})_{(i,j)\in[\![1;n]\!]\times[\![1;m]\!]}$ d'objet transportés depuis chaque fournisseur i vers chaque client j tels que

$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{i,j} \times b_{i,j}$ soit minimal, sous la contrainte des provisions $\sum_{j=1}^{m} b_{i,j} = P_i$ et des commandes $\sum_{i=1}^{n} b_{i,j} = C_j$. Il s'agit bien évidemment du problème étudié en cours.

Dans ce cadre de ce projet, nous restreindrons l'écriture de notre programme au cas équilibré, c'est à dire tel que $\sum_{i=1}^{n} P_i = \sum_{j=1}^{m} C_j$. Aussi, vous travaillerez avec le langage de programmation de votre choix : C, C++, Python, Java.

1.2.2 Le tableau de contraintes

Pour chaque problème de transport, il faudra dans un premier temps créer un fichier .txt qui sera organisé sous la forme suivante :

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
Commande C_1	Commande C_2	...	Commande 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
Commandes 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 – Un exemple d'un problème de transport et de son tableau .txt
Les prix unitaires de transport sont en bleu.

En annexes, vous trouverez les 12 tableaux des 12 problèmes de transport que l'on vous demande de résoudre avec votre programme. Il vous faudra éditer les 12 tableaux dans 12 fichiers .txt différents de la façon ainsi définie. Ces fichiers devront être joints à votre rendu.

1.2.3 Les fonctions

Dans le programme que vous rédigerez, vous devrez mettre en place les fonctions suivantes :

1. Lecture des données issues du fichier texte (.txt) et son stockage en mémoire.
2. Affichage des tableaux suivants :

- ⋆ matrice des coûts
- ⋆ proposition de transport
- ⋆ table des coûts potentiels
- ⋆ table des coûts marginaux

Attention : la fonction affichage doit être absolument soignée. Toute table comportant des colonnes qui se décalent sera très lourdement sanctionné. La lisibilité des tables est fondamentale.

3. Algorithme pour fixer la proposition initiale : Nord-Ouest.
4. Algorithme pour fixer la proposition initiale : Balas-Hammer.
   - ⋆ Calcul des pénalités.
   - ⋆ Affichage de la (ou des) ligne(s) (ou colonnes) de pénalité maximale.
   - ⋆ Choix de l'arête à remplir.
5. Calcul du coût total pour une proposition de transport donnée.
6. Algorithme de résolution : la méthode du marche-pied avec potentiel.
   - ⋆ Test pour savoir si la proposition est acyclique : on utilisera un parcours en largeur. Lors de ce parcours, en découvrant les sommets, on vérifiera si on retourne sur un sommet déjà visité et que ce sommet n'est pas le parent du sommet courant, si c'est le cas, alors il existe un cycle. Puis, on affiche le cycle.
   - ⋆ Maximisation du transport si un cycle a été détecté. On affiche les conditions pour chacune des cases. Puis on affiche l'arête supprimée (éventuellement plusieurs) à l'issue de la maximisation.
   - ⋆ Test pour savoir si la proposition est connexe : on utilisera un parcours en largeur. Si elle est non connexe : affichage des sous graphes connexes composant la proposition.
   - ⋆ Modification du graphe s'il est non connexe, jusqu'à l'obtention d'une proposition non dégénérée.
   - ⋆ Calcul et affichage des potentiels par sommet.
   - ⋆ Affichage des deux tables des coûts potentiels et des marginaux. Détection éventuelle de la meilleure arête améliorante.
   - ⋆ Ajout de cette arête améliorante à la proposition de transport, si elle a été détectée.

Attention : Chaque fonction devra être mise en évidence et expliquée clairement à l'oral à l'aide d'un pseudo-code.

Pour avoir la moyenne à ce projet, vous devrez impérativement avoir les fonctions suivantes qui marchent :

- ⋆ Lecture des données et son stockage en mémoire.
- ⋆ Algorithmes Nord-Ouest et Balas-Hammer.
- ⋆ Calcul du coût total pour une proposition de transport donnée.
- ⋆ Test pour savoir si la proposition est acyclique en utilisant un parcours en largeur.
- ⋆ Maximisation du transport sur un cycle détecté.
- ⋆ Test pour savoir si la proposition est connexe en utilisant un parcours en largeur.
- ⋆ Affichage de toutes les tables.

1.2.4 La structure globale

La structure globale de votre programme est illustrée par le pseudo-code suivant :

Début
    Tant que l'utilisateur décide de tester un *problème de transport* faire
        Choisir le numéro du problème à traiter
        Lire le tableau de contraintes sur fichier et le stocker en mémoire
        Créer les matrices correspondantes représentant ce tableau et l'afficher
        Demander à l'utilisateur de choisir l'algorithme pour fixer la proposition initiale et l'exécuter.
        Afficher les éléments précédemment évoqués lors de l'exécution des deux algorithmes.
        Dérouler la méthode du marche-pied avec potentiel en affichant à chaque itération :
            ⋆ Affichage de la proposition de transport, ainsi que son coût de transport total.
            ⋆ Test pour savoir si la proposition de transport est dégénérée.
            ⋆ Modifications du graphe de transport pour obtenir un arbre, dans les cas cyclique ou non connexe.
            ⋆ Calcul et affichage des potentiels.
            ⋆ Affichage des tables : coûts potentiels et coûts marginaux.
                ⋆ Si elle n'est pas optimale.
                ⋆ Affichage de l'arête à ajouter.
                ⋆ Maximisation du transport sur le cycle formé et une nouvelle itération.
                ⋆ Sinon sortir de la boucle
                ⋆ Fin si
        Afficher la proposition de transport optimale, ainsi que son coût.
        Proposer à l'utilisateur de changer de problème de transport
    fin Tant que
Fin

1.2.5 Améliorations possibles

Une fois l'algorithme suffisamment testé, nous vous proposons les améliorations suivantes :

1. Lors du "Modifications du graphe de transport pour obtenir un arbre, dans les cas cyclique ou non connexe", il faudra d'abord détecter si le graphe présente un cycle. Après avoir maximisé sur ce cycle la proposition de transport, il se peut qu'il reste d'autres cycles. Il faudra alors relancer, de manière répétée, les fonctions de "Détection de cycle", puis "Maximisation sur le cycle" jusqu'à l'obtention d'une proposition acyclique. Ce n'est qu'après que l'on effectuera le test de connexité où l'on complétera, au besoin, le graphe avec des arêtes classées selon des coûts croissants jusqu'à l'obtention d'une proposition connexe et acyclique.
2. Lors de l'exécution de la fonction "Maximiser le transport sur un cycle", il se peut que $\delta = 0$, c'est à dire que cela n'induit pas de modification sur le cycle. Vous pouvez alors détecter ce cas particulier. On agira ainsi : on conservera l'arête améliorante détectée avec la table des coûts marginaux (s'il s'agit de la boucle générale) et on enlèvera l'entièreté des dernières arêtes ajoutées lors du dernier test de connexité, à la même itération. La fonction "Modification du graphe s'il est non connexe" qui suivra proposera alors un ensemble différent d'arêtes.

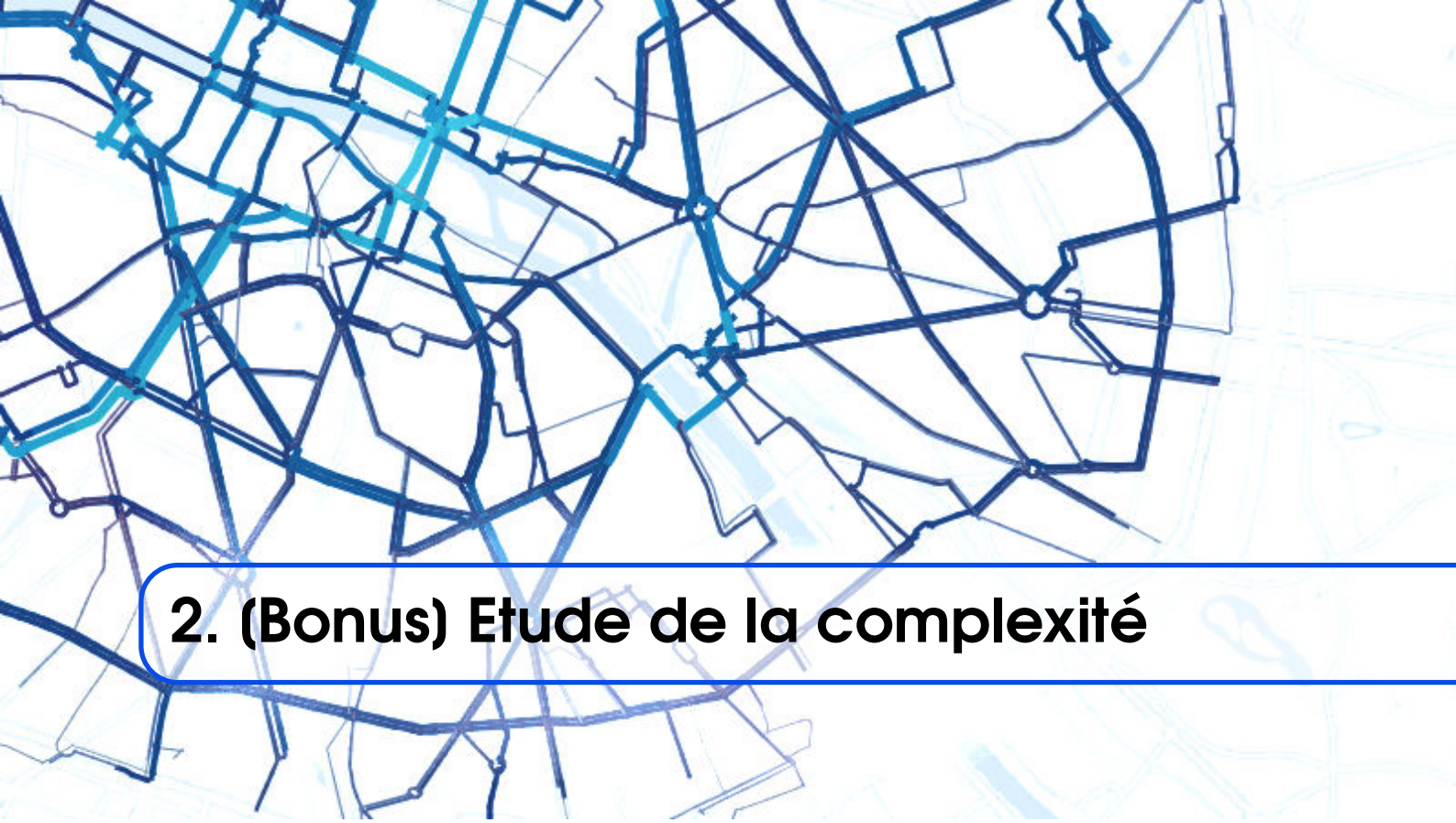
1.3 Les traces d'exécution

Les traces d'exécution pour les 12 graphes fournis en annexes sont demandés dans le rendu. On appelle trace d'exécution ce qui est affiché par la console.

Il faudra exécuter avec votre programme les 12 problèmes dans les deux cas : proposition initiale Nord-Ouest NO puis Balas-Hammer BH. Puis on affichera le déroulé du marche-pied avec potentiel avec l'entièreté des tableaux et des informations précédemment demandées.

Les fichiers seront stockés de la façon suivante :

- ⋆ Groupe B - Equipe 4 - Problème 5 - Nord-Ouest : "B4-trace5-no.txt"
- ⋆ Groupe D - Equipe 2 - Problème 12 - Balas-Hammer : "D2-trace12-bh.txt"



2. (Bonus) Etude de la complexité

2.1 Introduction

Cette partie est une partie bonus qui ne doit être abordée que si votre programme fonctionne parfaitement dans les 12 cas des annexes. Nous vous proposons maintenant d'étudier la *complexité* des algorithmes de ce projet.

Mais d'abord qu'est-ce que la complexité d'un algorithme ? Il s'agit de l'évaluation des ressources nécessaires à exécution d'un algorithme (essentiellement la quantité de mémoire requise) et le temps de calcul à prévoir. Ces deux notions dépendent de nombreux paramètres matériels qui sortent du domaine de l'algorithmique : nous ne pouvons attribuer une valeur absolue ni à la quantité de mémoire requise ni au temps d'exécution d'un algorithme donné. En revanche, il est souvent possible d'évaluer l'*ordre de grandeur* de ces deux quantités de manière à identifier l'algorithme le plus efficace au sein d'un ensemble d'algorithmes résolvant le même problème.

C'est ce que nous nous proposons de faire ici en comparant les propositions de transports issues des algorithmes de Nord-Ouest, Balas-Hammer lors de la résolution de la méthode du marche-pied avec potentiel.

2.2 Un peu de théorie

La plupart des algorithmes ont un temps d'exécution qui dépend non seulement de la taille des données en entrée mais des données elles-mêmes. Dans ce cas on distingue plusieurs types de complexités :

Définition 2.1 (complexité dans le pire des cas)
La **complexité dans le pire des cas** est un majorant du temps d'exécution possible pour toutes les entrées possibles d'une même taille. On l'exprime en général à l'aide de la notation O.

Définition 2.2 (complexité dans le meilleur des cas)
La **complexité dans le meilleur des cas** est un minorant du temps d'exécution possible pour toutes les entrées possibles d'une même taille. On l'exprime en général à l'aide de la notation Ω. Cependant cette notion n'est que rarement utilisée car souvent peu pertinente au regard des complexités dans le pire des cas et en moyenne.

Définition 2.3 (complexité en moyenne)
La **complexité en moyenne** est une évaluation du temps d'exécution moyen portant sur toutes les entrées possible d'une même taille supposées équiprobable.

Définition 2.4 (complexité spatiale)
La **complexité spatiale** évalue la consommation en espace mémoire. Le principe est le même sauf qu'ici on cherche à évaluer l'ordre de grandeur du volume en mémoire utilisé : il ne s'agit pas d'évaluer précisément combien d'octets sont consommés par un algorithme mais de préciser son taux de croissance en fonction de la taille n de l'entrée.

2.3 L'étude

Dans ce projet, on analysera la **complexité dans le pire des cas**. Pour cela, nous vous demandons de générer des problèmes de transport aléatoires. Puis de regarder les temps d'exécution des algorithmes.

2.3.1 Les problèmes de transport en entrée

Afin de simplifier le problème, vous travaillerez avec des problèmes de transport de taille $n = m$. La matrice $A = (a_{i,j})_{(i,j)\in[\![1;n]\!]^2}$ est alors carrée.

Afin de générer un échantillonnage mimant toutes les entrées possibles d'une même taille n, vous écrirez une fonction pour éditer des problèmes de transport aléatoires. Elle se fera de la manière suivante :

1. On génère un nombre aléatoire entier entre 1 et 100 inclus pour chaque $a_{i,j}$.
2. On génère un nombre aléatoire entier entre 1 et 100 inclus pour chaque $(temp_{i,j})_{i,j}$ d'une matrice de taille $n \times n$. Vous remplirez ainsi les n valeurs des provisions $(P_i)_{i\in[\![1;n]\!]}$ et des commandes $(C_j)_{j\in[\![1;n]\!]}$ de la façons suivante :

$$P_i = \sum_{j=1}^{n} temp_{i,j} \text{ et } C_j = \sum_{i=1}^{n} temp_{i,j}$$

2.3.2 La mesure du temps

Une fois le problème généré, c'est à dire une fois que les P_i, les C_j et les $a_{i,j}$ sont fixés, il faudra stocker la valeur du temps de chaque portion de code qui nous intéresse. Par exemple en Python, il suffit simplement d'utiliser la fonction time.clock() qui renvoie le temps CPU en secondes et de stocker cette valeur. La différence entre 2 valeurs relevées donnera le temps d'exécution de la portion de code encadrée.

Avec ce problème de taille n généré, vous devrez mesurer le temps d'exécution de :

1. l'algorithme Nord-Ouest. On appellera ce temps $\theta_{NO}(n)$,
2. l'algorithme Balas-Hammer. On appellera ce temps $\theta_{BH}(n)$,
3. l'algorithme du marche-pied avec la proposition issue de Nord-Ouest. On appellera ce temps $t_{NO}(n)$,
4. l'algorithme du marche-pied avec la proposition issue de Balas-Hammer. On appellera ce temps $t_{BH}(n)$,

2.3.3 Le nuage de points

Pour chaque valeurs de n, vous ferez tourner 100 fois votre programme avec des valeurs aléatoires différentes pour le problème de transport. On obtiendra donc, pour n fixé, 100 valeurs de $\theta_{NO}(n)$ par exemple.

Valeurs de n à tester	10	40	10^2	4.10^2	10^3	4.10^3	10^4

Attention : pour effectuer cette tâche, vous travaillerez sur une seule machine à processeur unique. Les instructions seront exécutées l'une après l'autre, sans opération simultanées. Il ne faudra donc pas utiliser votre machine pour faire autre chose pendant toute l'exécution.

Une fois les valeurs stockées, vous tracerez en fonction de n, les nuages de points (les 100 valeurs pour une même abscisse) suivants :

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

2.3.4 La complexité dans le pire des cas par algorithme

On suppose que la complexité dans le pire des cas est l'enveloppe supérieure du nuage de points. Pour chaque valeur de n, déterminer cette valeur maximale au travers des 100 réalisations pour n fixé. Puis tracer en fonction de n la valeur maximale trouvée.

Pour θ_{NO}, θ_{BH}, t_{NO}, t_{BH}, $(\theta_{NO}+t_{NO})$ et $(\theta_{BH}+t_{BH})$ identifier le type de complexité dans le pire des cas à l'aide du tableau tableau 2.1 :

O($log(n)$)	logarithmique
O(n)	linéaire
O($nlog(n)$)	quasi-linéaire
O(n^2)	quadratique
O(n^k) (k > 2)	polynomiale
O(k^n) (k > 1)	exponentielle

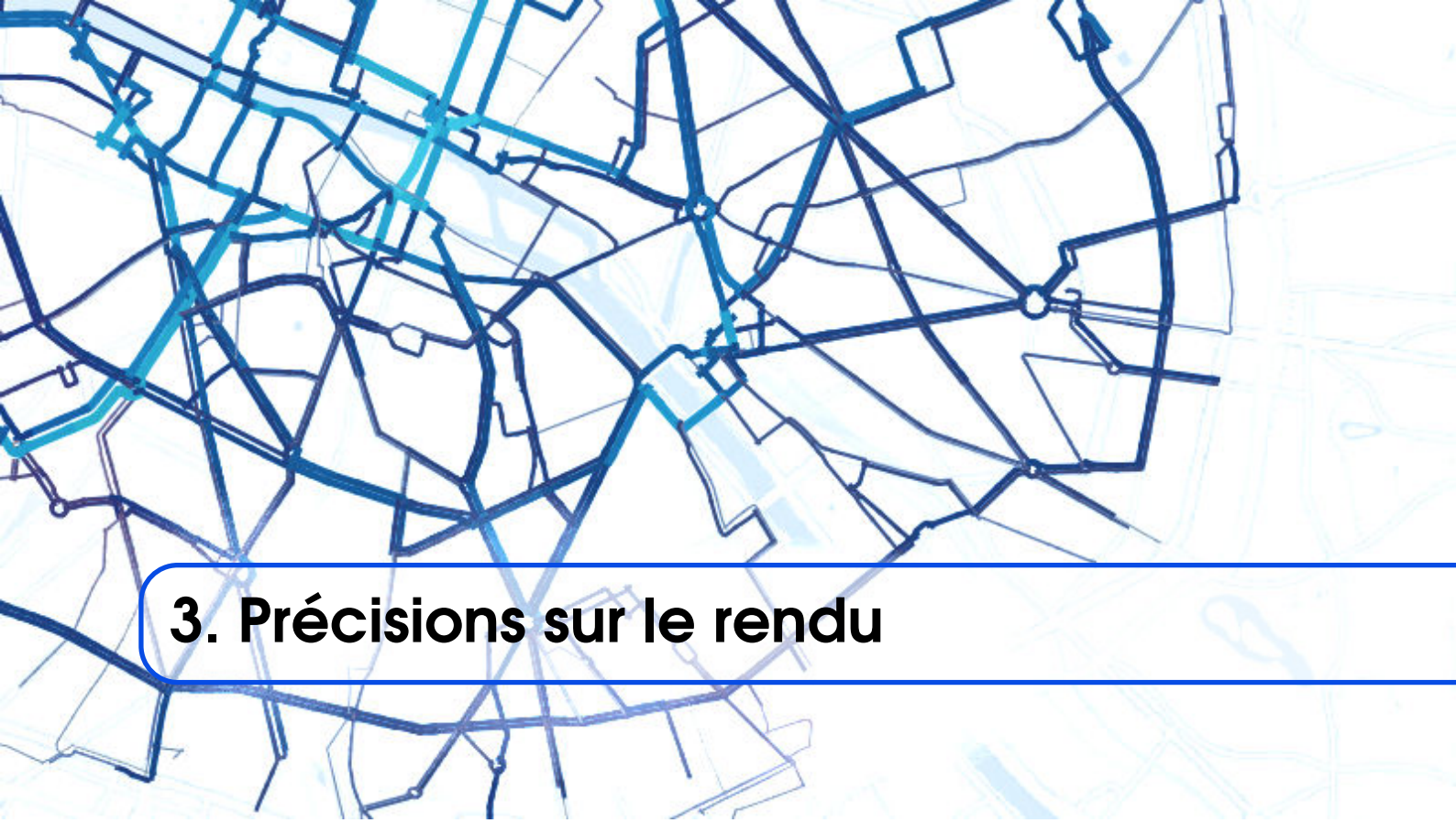
FIGURE 2.1 – Qualifications usuelles des complexités.

2.3.5 Comparaison de la complexité dans le pire des cas

Comparons maintenant les deux algorithmes résolvant le même problème pour n fixé en traçant :

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

Tracez ensuite la valeur maximale trouvée pour chaque valeur de n et discutez des résultats.



3. Précisions sur le rendu

3.1 Le rendu

Le dépôt de votre projet se fera sur Moodle jusqu'au Samedi 16 décembre à 23h59. Aucun délais supplémentaire ne pourra être accepté.

Dans le dépôt, vous joindrez l'entièreté de vos programmes, les 12 traces d'exécutions et les 12 fichiers .txt des problèmes de transport à résoudre. La notation tiendra compte de la qualité des algorithmes et des traces présentées. Un rapport est seulement demandé pour les équipes ayant fait l'étude bonus sur la complexité et qui fera au maximum 5 pages.

Le dossier de rendu sera intitulé de la façon suivante : pour le groupe B et l'équipe 4 : "B4".

3.2 L'oral

Pour l'oral, il est attendu une présentation avec des diapositives. La pédagogie et la clarté sont le but de cette présentation de 15 min. Nous n'accepterons pas de code sur les diapositives : nous demandons seulement du pseudo-code. Les diapositives ne doivent évidemment pas comporter de texte écrit à la main et photographié. Toutes les formules et matrices doivent être tapées.

A l'oral, il est demandé de restituer avec précision les points les plus importants de chaque partie. Une diapositive qui résume le travail effectué à savoir les fonctions que vous avez réussies à faire fonctionner et celles non réalisées est attendue. Attention, si vous dépassez les 15 min de présentation, votre enseignant vous arrêtera.

A l'issue de l'oral, des questions vous seront posées pendant 15 min : chaque étudiant sera interrogé individuellement sur le projet ou sur un point du cours.

Bon courage pour la rédaction de ce projet,

L'équipe des enseignants de Recherche Opérationnelle.

3.3 Annexes : les douze propositions de transport à tester

Les prix unitaires de transport sont en bleu.

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

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

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

4	C_1	C_2	Provisions
P_1	30	1	600
P_2	1	30	500
Commandes	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
Commandes	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
Commandes	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
Commandes	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
Commandes	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	