

INFO-F-3010

TSP Problem Project
Algorithmique et recherche
opérationnelle

Chris Badi Budu / B-INFO / 3^e : 000569082

Pietro Narcisi / B-INFO / 3^e : 000567641

Université Libre de Bruxelles

Table des matières

1	Introduction	2
2	Compréhension des mécanismes d'élimination de sous-tours	2
2.1	Formulation MTZ	2
2.1.1	Exemple	2
2.2	Formulation DFJ	3
2.2.1	Exemple	4
3	Compréhension des formulations	6
3.1	Résultats obtenus pour chaque instance	6
3.2	Analyse des résultats	6
4	Analyse de la relaxation continue	9
4.1	Analyse des résultats	9
5	Questions de réflexion	10
5.1	2^n vs $(n - 1)!/2$	10
5.2	Paradoxe DFJ-MTZ	11
6	Bonus	11

1 Introduction

Dans ce projet, nous avons implémenté des fonctions permettant de résoudre le problème du voyageur de commerce via les formulations mathématiques de Miller-Tucker-Zemlin (MTZ) et celle de Dantzig-Fulkerson-Johnson (DFJ). Sur base de ces implémentations, nous allons analyser et expliquer les résultats que nous obtenons en exécutant les différentes formulations demandées qui sont :

1. MTZ classique
2. La relaxation continue de MTZ
3. DFJ énumératif
4. La relaxation continue DFJ énumératif
5. DFJ itératif

2 Compréhension des mécanismes d'élimination de sous-tours

2.1 Formulation MTZ

La formulation MTZ modélise le problème du voyageur de commerce en utilisant des variables de décision binaires x_{ij} (valant 1 si l'arc (i,j) est utilisé) et des variables continues u_i représentant l'ordre de visite. Le modèle s'écrit :

$$\begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j \neq i, j=1}^n c_{ij} x_{ij} \\ \text{s.c.} \quad & \sum_{i=1, i \neq j}^n x_{ij} = 1 \quad \forall j \in \{1, \dots, n\} \\ & \sum_{j=1, j \neq i}^n x_{ij} = 1 \quad \forall i \in \{1, \dots, n\} \\ & u_i - u_j + 1 \leq (n-1)(1-x_{ij}) \quad \forall i, j \in \{2, \dots, n\}, i \neq j \\ & 2 \leq u_i \leq n \quad \forall i \in \{2, \dots, n\} \\ & x_{ij} \in \{0, 1\} \end{aligned}$$

L'idée intuitive de MTZ est d'attribuer un « ordre de passage » (ou un potentiel) à chaque ville i , représenté par la variable continue u_i .

La contrainte spécifique impose que si l'on voyage de la ville i vers la ville j (c'est-à-dire si $x_{ij} = 1$), alors la ville j doit avoir un numéro d'ordre strictement supérieur à celui de la ville i . Mathématiquement, la contrainte se simplifie en $u_i - u_j + 1 \leq 0$, soit $u_j \geq u_i + 1$.

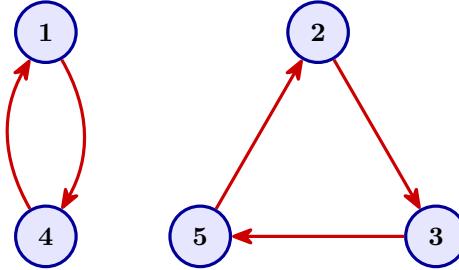
Cela force le tour à suivre une progression croissante des indices u . Comme il est impossible d'avoir une suite de nombres strictement croissants qui tourne en boucle, les sous-tours isolés qui ne passent pas par la ville de départ (ville 1) sont interdits.

2.1.1 Exemple

Considérons une solution irréalisable, composée de deux sous-tours disjoints :

1. $S_1 = \{1, 4\}$ (contient la ville de départ 1).
2. $S_2 = \{2, 3, 5\}$ (ne contient pas la ville de départ).

La formulation MTZ empêche cette configuration grâce aux variables de potentiels u_i (l'ordre de visite). Les contraintes $u_i - u_j + 1 \leq (n - 1)(1 - x_{ij})$ doivent être respectées pour tout arc actif ($x_{ij} = 1$) ne partant pas de la base.



Sous-tour $\{1, 4\}$ Sous-tour $\{2, 3, 5\}$
 (Connecté à la base) (Violé : Incompatibilité des u_i)

Le sous-tour $\{2, 3, 5\}$ isolé qui ne passe pas par la ville 1 viole les contraintes du MTZ. Car comme expliqué plus tôt, celles-ci imposent que si $x_{ij} = 1$, alors $u_i - u_j \leq -1$ (on doit "monter" dans l'ordre de visite).

Appliquons cela aux trois arcs du cycle :

$$\begin{aligned} \text{Arc } 2 \rightarrow 3 : \quad & u_2 - u_3 \leq -1 \\ \text{Arc } 3 \rightarrow 5 : \quad & u_3 - u_5 \leq -1 \\ \text{Arc } 5 \rightarrow 2 : \quad & u_5 - u_2 \leq -1 \end{aligned}$$

En additionnant ces trois inéquations membre à membre, les variables u_i s'annulent :

$$\begin{aligned} (u_2 - u_3) + (u_3 - u_5) + (u_5 - u_2) & \leq -1 - 1 - 1 \\ 0 & \leq -3 \end{aligned}$$

Ce résultat est donc impossible. Le modèle rejette donc cette solution, forçant le solveur à chercher un tour unique connecté.

2.2 Formulation DFJ

Soient les variables de décision binaires :

$$x_{ij} = \begin{cases} 1 & \text{si le chemin va de la ville } i \text{ à la ville } j \\ 0 & \text{sinon} \end{cases} \quad i, j = 1, \dots, n$$

L'objectif est de minimiser la distance totale parcourue :

$$\min \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n c_{ij} x_{ij}$$

La formulation mathématique DFJ génère une solution sans sous-tours grâce à sa contrainte qui les élimine. Par la suite, on ne récupère que les villes qui sont encore marquées et qui constituent la solution la plus courte.

Les deux contraintes suivantes sont les contraintes de degré. Elles garantissent que chaque ville est visitée exactement une fois : le voyageur doit obligatoirement arriver dans la ville j (première équation) et en repartir (deuxième équation).

$$\sum_{\substack{i=1 \\ i \neq j}}^n x_{ij} = 1 \quad j = 1, \dots, n$$

$$\sum_{\substack{j=1 \\ j \neq i}}^n x_{ij} = 1 \quad i = 1, \dots, n$$

La condition $j \neq i$ assure simplement qu'on ne considère pas les mouvements immobiles (boucles sur soi-même), correspondant à la diagonale de zéros dans la matrice des distances.

0.00	36.65	90.76	30.91	73.40	46.81	88.36	57.07	47.95	47.94
36.65	0.00	38.92	79.27	97.66	41.15	22.80	88.31	82.72	19.26
90.76	38.92	0.00	76.53	36.95	81.91	43.67	29.42	41.25	97.46
30.91	79.27	76.53	0.00	49.55	29.23	83.42	34.15	33.99	42.86
73.40	97.66	36.95	49.55	0.00	97.29	51.04	24.10	20.63	69.76
46.81	41.15	81.91	29.23	97.29	0.00	49.76	38.39	56.06	70.98
88.36	22.80	43.67	83.42	51.04	49.76	0.00	58.20	21.32	46.98
57.07	88.31	29.42	34.15	24.10	38.39	58.20	0.00	36.26	11.40
47.95	82.72	41.25	33.99	20.63	56.06	21.32	36.26	0.00	10.36
47.94	19.26	97.46	42.86	69.76	70.98	46.98	11.40	10.36	0.00

FIGURE 1 – Diagonale de 0

Enfin, la contrainte de ci-dessous garantit l'élimination de sous tours pour chaque sous-ensemble Q . Pour ce faire, on va compter le nombre de villes qui se trouvent dans le sous-ensemble et vérifier qu'il y ait moins d'arcs que de villes, sinon cela voudrait dire qu'un cycle se forme.

$$\sum_{i \in Q} \sum_{\substack{j \in Q \\ j \neq i}} x_{ij} \leq |Q| - 1 \quad \forall Q \subsetneq \{1, \dots, n\}, |Q| \geq 2$$

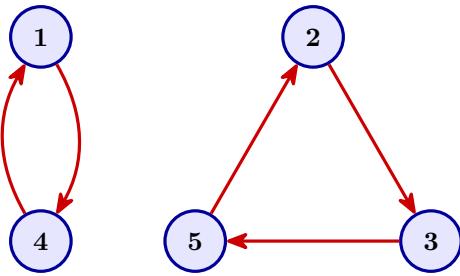
Par exemple, prenons un sous-ensemble Q de 4 villes ($|Q| = 4$). La contrainte impose que la somme des arcs internes soit ≤ 3 . Si le solveur proposait une solution avec 4 arcs connectant ces 4 villes (par exemple $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$), la somme serait de 4. Comme $4 \not\leq 3$, cette configuration de sous-tour est détectée et rejetée par la contrainte.

2.2.1 Exemple

Dans cet exemple ci-dessous, on aura 5 villes et deux sous-tours :

1. $S_1 = \{1, 4\}$ avec comme arcs $4 \rightarrow 1 \rightarrow 4$.
2. $S_2 = \{2, 3, 5\}$ avec comme arcs $2 \rightarrow 3 \rightarrow 5 \rightarrow 2$.

On a pas d'arcs qui va de $i \rightarrow i$, donc la seule contrainte violée est celle contre la formation de sous-tours.



$$\begin{array}{ll} \text{Sous-tour } S_1 = \{1, 4\} & \text{Sous-tour } S_2 = \{2, 3, 5\} \\ (\text{Violé : } 2 \not\leq 1) & (\text{Violé : } 3 \not\leq 2) \end{array}$$

On voit donc que si nous ne gérons pas le cas où seulement les contraintes communes sont respectées, des boucles peuvent être formées dans notre solution car il faut que le nombre d'arcs sélectionnées dans un sous-ensemble soit inférieur au nombre de villes de ce sous-ensemble ($|S| \leq |Q|-1$). Dans le cas contraire, la résolution du problème n'est pas satisfaitte.

3 Compréhension des formulations

3.1 Résultats obtenus pour chaque instance

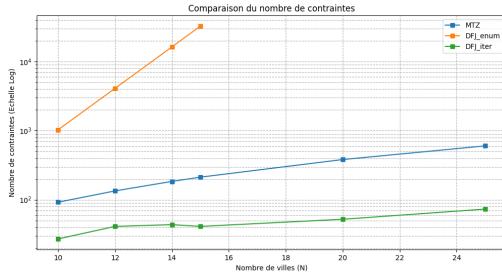


FIGURE 2

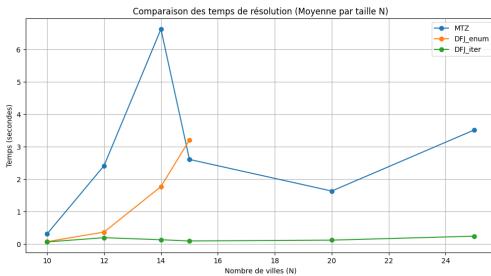


FIGURE 3

Nous avons donc testé toutes (pas les $n > 15$ sur dfj_enum) les instances données sur notre version des algos. on observe plusieurs points évidents qui sortent de ces tests :

Premièrement, pour toutes les instances et sur toutes les méthodes on obtient la même valeur de fonction objectif. Cette première observation a donc confirmé ma validité de nos implementations.

Ensuite, on observe aussi une grande disparité du nombre de contraintes entre les trois méthodes. DJF itératif avec la plus petite quantité de contraintes et reste plutôt faible avec la taille de l'instance grandissante, même pour $n=25$, le nombre total de contraintes de sous-tours ajoutées n'a jamais dépassé 80. Suivi de MTZ qui possède une quantité de contraintes facilement distinguable, elle augmente de manière quadratique et correspond parfaitement au résultat mathématiques attendus. $n=10$ à $n=15$, le nombre de contraintes passe de 92 à 212. Et pour finir pour la formulation DFJ Énumérative subit une croissance exponentielle violente. Pour les mêmes tailles ($n=10$, $n=15$), on passe de 1 032 à 32 781 contraintes

Au niveau des performances observées, on voit que DFJ Itératif est systématiquement l'approche la plus rapide (reste constamment sous la seconde). MTZ montre des signes de faiblesse sur certaines instances spécifiques (ex : instance_14_line_1 prend constamment plus de 10 secondes contre 0.3s pour DFJ). Le temps d'exécution du DFJ énumératif reste très stable pour un même n et est globalement plus rapide que MTZ jusqu'à $n=15$.

3.2 Analyse des résultats

L'analyse de ces résultats et de nos observations met en évidence un compromis important entre la taille du modèle et la stratégie de résolution.

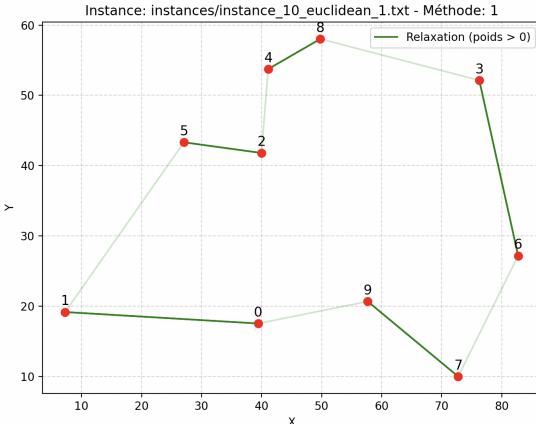


FIGURE 4

Premièrement, le fait que nous ne sommes pas appelés à tester DFJ Énumérative sur les instances $n > 15$ due à son immense taille et son temps d'exécution (pour $n=20$ notre pc de test a crash et le processus n'as donc jamais fini de tourner) illustre parfaitement le problème de l'explosion combinatoire. En voulant interdire tous les sous-tours possibles, cette méthode génère une quantité massive de contraintes (plus de 32 000 pour 15 villes) dont l'immense majorité est inutile pour trouver la solution optimale. Le solveur est alors "noyé" par la taille du problème avant même de commencer à chercher efficacement.

C'est ici que la méthode DFJ Itérative démontre sa supériorité. Nos chiffres montrent que sur les milliards de sous-tours possibles, seule une poignée (moins de 80, même pour $n=25$) a réellement besoin d'être interdite pour converger vers la solution. En n'ajoutant que ces contraintes "actives" au fur et à mesure, cette approche maintient le modèle extrêmement léger, ce qui explique sa rapidité d'exécution (toujours sous la seconde).

Enfin, concernant MTZ, bien que cette formulation évite l'explosion exponentielle grâce à sa croissance polynomiale (quadratique), elle reste une approche "statique". Elle oblige le solveur à gérer un nombre fixe de variables supplémentaires (u_i) et de contraintes (602 pour $n=25$) dès le départ. À l'inverse, l'approche itérative termine souvent avec moins de contraintes finales que MTZ (72 contre 602 pour $n=25$), ce qui la rend plus efficace en termes de gestion mémoire et de temps de calcul sur les instances testées. Un autre phénomène notable est l'inconstance de son temps d'exécution. Alors que DFJ Itératif reste stable, MTZ subit des pics de temps de calcul sur certaines instances (notamment *instance_14_line_1* qui prend plus de 21 secondes). Cela s'expliquerait par la nature des variables supplémentaires u_i . Dans MTZ, le solveur ne doit pas seulement décider « quelles routes prendre » (variables x_{ij}), il doit aussi réussir à attribuer un « numéro d'ordre » cohérent à chaque ville (variables u_i). Le problème est que les contraintes qui lient ces deux décisions sont de type « Big-M » :

$$u_i - u_j + n \cdot x_{ij} \leq n - 1$$

Quand un arc n'est pas pris ($x_{ij} = 0$), la contrainte devient $u_i - u_j \leq n - 1$. Cette inéquation est presque toujours vraie et ne donne quasiment aucune information au solveur pour éliminer des mauvaises solutions. Le solveur doit donc explorer un arbre de possibilités (*branch-and-bound*) beaucoup plus vaste pour trouver la bonne combinaison de u_i , surtout sur des instances structurées comme des lignes ou des cercles où l'ordre est très contraint géométriquement. À l'inverse, DFJ travaille uniquement sur la structure du graphe (les connexions directes). Il n'a pas à gérer cette couche artificielle de numérotation, ce qui rend sa performance beaucoup plus prévisible et moins sensible à la topologie de l'instance.

Nous concluons donc que la stratégie dynamique (Itératif) l'emporte sur les stratégies statiques (MTZ et Énumératif) car elle adapte la complexité du problème à l'instance précise que l'on cherche à résoudre, au lieu d'imposer une structure lourde dès le départ.

TABLE 1 – Comparaison expérimentale - Formulation MTZ

Instance	Vars	Contraintes	Temps (s)	Objectif
instance_10_circle_1	99	92	0.4108	298.57
instance_10_euclidean_1	99	92	0.0840	207.85
instance_10_euclidean_2	99	92	0.1517	323.92
instance_10_line_1	99	92	0.9693	156.14
instance_10_random_asym_1	99	92	0.0361	191.53
instance_10_random_sym_1	99	92	0.2186	280.99
instance_12_circle_1	143	134	0.9653	265.22
instance_12_line_1	143	134	3.8597	174.80
instance_14_circle_1	195	184	0.6830	304.48
instance_14_line_1	195	184	12.5698	188.02
instance_15_euclidean_1	224	212	3.5535	306.70
instance_15_euclidean_2	224	212	6.2765	345.91
instance_15_random_asym_1	224	212	0.1103	342.94
instance_15_random_sym_1	224	212	0.4907	377.50
instance_20_euclidean_1	399	382	0.4871	370.13
instance_20_euclidean_2	399	382	3.8792	396.75
instance_20_random_asym_1	399	382	0.3141	324.08
instance_20_random_sym_1	399	382	1.8501	371.10
instance_25_euclidean_1	624	602	2.5075	410.26
instance_25_euclidean_2	624	602	4.5200	418.56

Instance	DFJ Énumératif			DFJ Itératif			Objectif
	Vars	Constr	Temps (s)	Vars	Constr Finales	Temps (s)	
Instance	DFJ Énumératif			DFJ Itératif			Objectif
	Vars	Constr	Temps (s)	Vars	Constr Finales	Temps (s)	
instance_10_circle_1	90	1032	0.0623	90	28	0.0641	298.57
instance_10_euclidean_1	90	1032	0.0601	90	25	0.0437	207.85
instance_10_euclidean_2	90	1032	0.0622	90	28	0.0609	323.92
instance_10_line_1	90	1032	0.1128	90	33	0.1266	156.14
instance_10_random_asym_1	90	1032	0.0602	90	20	0.0208	191.53
instance_10_random_sym_1	90	1032	0.0719	90	28	0.0651	280.99
instance_12_circle_1	132	4106	0.2973	132	39	0.1718	265.22
instance_12_line_1	132	4106	0.4377	132	43	0.2170	174.80
instance_14_circle_1	182	16396	1.6157	182	34	0.0486	304.48
instance_14_line_1	182	16396	1.9242	182	53	0.2136	188.02
instance_15_euclidean_1	210	32781	3.2926	210	50	0.1571	306.70
instance_15_euclidean_2	210	32781	3.1823	210	45	0.1091	345.91
instance_15_random_asym_1	210	32781	3.0802	210	32	0.0553	342.94
instance_15_random_sym_1	210	32781	3.2744	210	37	0.0528	377.50
instance_20_euclidean_1	-	-	-	380	50	0.0588	370.13
instance_20_euclidean_2	-	-	-	380	56	0.1179	396.75
instance_20_random_asym_1	-	-	-	380	43	0.0574	324.08
instance_20_random_sym_1	-	-	-	380	59	0.2416	371.10
instance_25_euclidean_1	-	-	-	600	72	0.2763	410.26
instance_25_euclidean_2	-	-	-	600	74	0.2060	418.56

TABLE 2 – Comparaison expérimentale - Formulation DFJ (Énumérative vs Itérative)

4 Analyse de la relaxation continue

TABLE 3 – Comparaison des relaxations linéaires (MTZ vs DFJ)

Instance	Solution Entière	MTZ		DFJ	
		Relaxation	Gap	Relaxation	Gap
instance_10_circle_1	298.57	248.58	16.74%	298.57	0.00%
instance_10_euclidean_1	207.85	200.21	3.68%	207.85	0.00%
instance_10_euclidean_2	323.92	312.94	3.39%	323.92	0.00%
instance_10_line_1	156.14	81.49	47.81%	156.14	0.00%
instance_10_random_asym_1	191.53	191.53	0.00%	191.53	0.00%
instance_10_random_sym_1	280.99	262.29	6.65%	280.99	0.00%
instance_12_circle_1	265.22	176.08	33.61%	265.22	0.00%
instance_12_line_1	174.80	111.26	36.35%	174.80	0.00%
instance_14_circle_1	304.48	221.94	27.11%	304.48	0.00%
instance_14_line_1	188.02	71.59	61.93%	188.02	0.00%
instance_15_euclidean_1	306.70	228.74	25.42%	306.70	0.00%
instance_15_euclidean_2	345.91	243.46	29.62%	345.91	0.00%
instance_15_random_asym_1	342.94	342.64	0.09%	342.94	0.00%
instance_15_random_sym_1	377.50	332.28	11.98%	377.50	0.00%

4.1 Analyse des résultats

Après exécution des deux algorithmes avec relaxation linéaire, nous allons analyser nos observations.

Dans un premier temps, nous remarquons directement pour l'exécution de DFJ avec relaxation que pour toutes les implémentations, il atteint toujours la solution entière. Cela s'explique par le fait que la relaxation DFJ définit le "subtour polytope". C'est l'intersection de toutes les contraintes posées qui délimite le périmètre de résolution au solveur. Ce dernier va tenter de résoudre le problème avec un coût minimum dans la zone déterminée.

Toutefois, nous sommes dans la configuration la optimale pour ne pas avoir (du moins pas beaucoup de chances) un gap énorme entre la solution entière et la solution par relaxation DFJ car il fallait que n soit pas supérieur à 6 pour permettre aux contraintes de générer un périmètre simple à construire. Car plus n est grand, plus le polytope perd de sa précision au niveau des sommets fractionnaires, délimitant la zone de résolution. Mais DFJ reste assez robuste pour tout de même faire que le solveur converge vers la solution entière optimale, ce qui explique pourquoi nous n'avons pas de gap malgré la relaxation linéaire. Pour le cas de MTZ, nous voyons qu'il est très souvent éloigné de la solution entière, voire très souvent sauf à quelques instances près (ex : instance_10_random_asym_1 qui donne la solution entière) car il fonctionne complètement différemment.

Étant donné que nous sommes dans une relaxation linéaire, il nous est permis de ne pas utiliser des entiers. Comme le solveur cherche des valeurs minimales, si comme dans le cas du MTZ, les contraintes ne sont pas strictes comme celle de DFJ, on peut se retrouver avec des solutions beaucoup plus inférieures que les solutions entières.

Avec MTZ, la relaxation est vraiment appliquée et les contraintes sont tolérantes aux solutions de moindre coût car sur papier, les contraintes sont respectées.

Cependant, il est possible, comme dans le cas mentionné plus haut, que l'on tombe sur la solution entière car dans ce cas précis, pas possible de simplifier le chemin de retour car le chemin aller et retour sont asymétriques. Contrairement aux autres cas, où le chemin aller retour peut être raccourci.

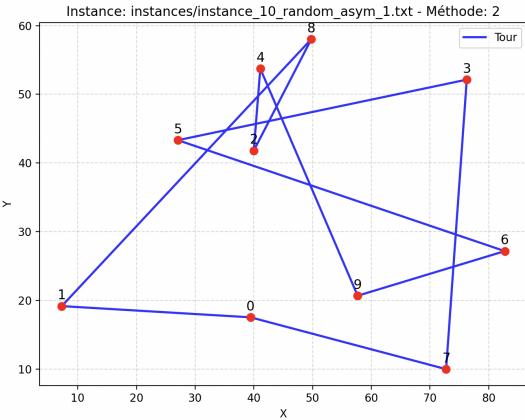


FIGURE 5

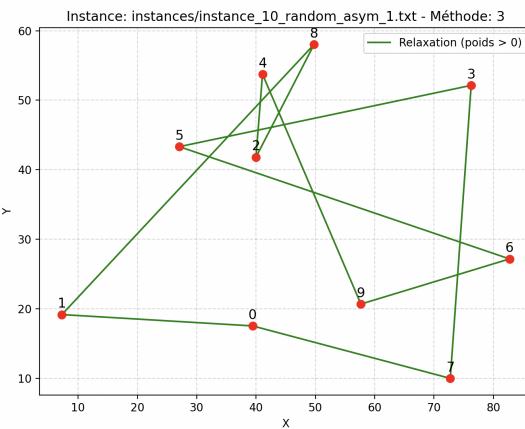


FIGURE 6

5 Questions de réflexion

5.1 2^n vs $(n - 1)!/2$

Nous allons maintenant comparer deux approches qui sont toutes deux théoriquement exponentielles :

- L'approche DFJ (que nous avons testée), qui contient potentiellement 2^n contraintes.
- L'approche "force brute", qui énumère tous les cycles hamiltoniens possibles, soit $(n-1)!/2$ cycles.

Premièrement, comme énoncé plus tôt, nous comparons deux approches exponentielles, mais cela ne veut pas dire que leur ordre de grandeur est comparable. La fonction factorielle croît beaucoup plus rapidement que la fonction 2^n . Pour $n=20$:

- $2^{20} \approx 10^6$ (un million). C'est grand, mais cela reste gérable par un ordinateur (c'est d'ailleurs pour ça que notre version DFJ Énumérative a fonctionné jusqu'à $n=15$).
- $19!/2 \approx 6 \times 10^{16}$ (60 millions de milliards). C'est physiquement impossible à énumérer.

Ensuite, il est logique que les performances de DFJ surpassent largement celles de l'approche par énumération de cycles. Contrairement à cette dernière qui avance "à l'aveugle" (en testant bêtement chaque route possible), DFJ se sert de la programmation linéaire pour se guider mathématiquement vers la solution.

C'est là que la structure de la méthode fait toute la différence. Théoriquement nous avons 2^n contraintes, mais comme expliqué plus haut dans le rapport, l'approche itérative couplée à la program-

mation linéaire permet de n'utiliser qu'une infime partie de celles-ci (les contraintes actives) pour trouver l'optimum, ce qui est impossible avec une simple énumération de cycles.

5.2 Paradoxe DFJ-MTZ

Ce comportement s'explique par la distinction entre nombre de contraintes et la qualité de la relaxation .

La qualité de la relaxation (Le Gap) :

Comme démontré dans la section 4, la relaxation linéaire de DFJ est extrêmement forte (Gap de 0%). Le polyèdre défini par DFJ colle presque parfaitement à l'enveloppe des solutions entières. Le solveur trouve donc souvent la solution entière optimale directement à la racine, sans avoir besoin de diviser le problème.

À l'inverse, MTZ définit un polyèdre « mou » (Gap élevé). Le solveur perd un temps considérable à explorer un arbre de recherche gigantesque pour compenser la faiblesse des bornes initiales.

La gestion dynamique des contraintes :

L'argument du « nombre exponentiel de contraintes » n'est vrai que pour la version Énumérative (qui est effectivement lente).

Dans la version Itérative (celle qui est performante), le solveur ne charge jamais les 2^n contraintes en mémoire. Il commence avec un modèle vide de contraintes de sous-tours. Nos observations ont montré qu'il ne rajoute au final que quelques dizaines de contraintes « actives » (ex : 50 contraintes pour $n = 20$).

On conclut que en pratique, le modèle résolu par DFJ Itératif est souvent plus petit (en nombre de contraintes actives) que le modèle MTZ, tout en étant mathématiquement plus fort. C'est cette combinaison qui explique sa supériorité temporelle.

6 Bonus

Lorsqu'une solution détecte 2 sous-tours, les contraintes d'éliminations respectives sont redondantes car l'une est le miroir de l'autre. Donc ajouter l'autre est redondante.

Au départ, les deux sous ensembles sont disjoints. Pour ne pas faire la même chose deux fois, on doit savoir que le nombre d'arrêts qui sortent de l'un vaut le nombre de ceux qui entrent dans l'autre sous-ensemble. Pour ne pas être redondant, comme les deux sous-tours sont mutuellement impliquées dans ces contraintes, on ajoute la contrainte seulement du premier cycle car forcer un chemin à sortir d'un ensemble équivaut à dire qu'il revienne dans le même. Donc on va juste utiliser une contrainte pour briser les 2 sous-tours