

Anatomie d'une salade

LALEU Maxime 19276

TIPE - Thème: Transition, transformation, conversion.

2024 - 2025

Introduction

- Étude de la structure des salades
- Analyse des ingrédients
- Production de résultats réalistes

1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

5 Résultats

6 Annexe

Exploration initiale des données

- Base principale : RecipeNLG_dataset
- Contient recettes et annotations NER (entités nommées)
- Volume important : 2 231 142 entrées, diversité élevée des ingrédients
- Premières observations :
 - Variété des recettes sucrées et salées
 - Présence d'ingrédients communs et spécifiques : "semi-sweet chocolate chips"

Premier défi — Réduire la taille de la base

- Objectif : rendre calculable en temps raisonnable
`make_Vgs_grph`
- Filtrage des recettes :
 - Requête SQL : `LOWER(title) LIKE '%salad%'`
 - 2 231 142 → 153 822 recettes
 - Suppression des recettes sucrées → 66 097 recettes restantes
- Nettoyage des ingrédients :
 - Normalisation des noms (singulier, orthographe)
 - Suppression des ingrédients trop rares
- Résultat : base plus compacte et exploitable

1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

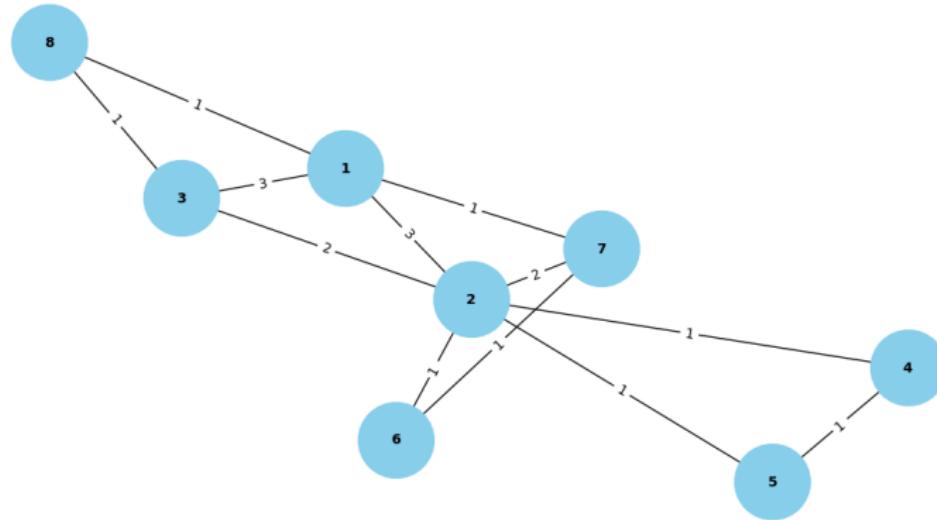
5 Résultats

6 Annexe

Construction du graphe de co-occurrences

- Objectif :
- Noeuds : ingrédients
- Arêtes : co-occurrence dans une même recette
- Poids des arêtes : nombre de co-occurrences (`cooc_count`)
- Pourquoi un graphe ?
 - Structure naturelle pour modéliser des dépendances
 - Outils algorithmiques puissants pour l'analyse

Exemple de graphe



1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

PageRank

k-MST

R-edge-MWCS

PMI

5 Résultats

6 Annexe

1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

PageRank

k-MST

R-edge-MWCS

PMI

5 Résultats

6 Annexe

PageRank : principe

- Algorithme de Google pour classer des pages web
- Idée centrale : un sommet est important s'il est lié à d'autres sommets importants
- Application ici : classer les ingrédients selon leur importance dans les salades
- Calcul sur un graphe pondéré :
 - Arêtes : co-occurrences d'ingrédients
 - Poids : fréquence de co-occurrence (`cooc_count`)

Résultats du PageRank

- Top 10 ingrédients par importance (selon le score PageRank)
- Analyse :
 - Ingrédients « pivots » dans les salades
 - Centralité \neq popularité brute (structure prise en compte)
- Permet d'identifier des ingrédients de base ou « ponts »

Graphe PageRank (top 20 ingrédients)



1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

PageRank

k-MST

R-edge-MWCS

PMI

5 Résultats

6 Annexe

Énoncé du problème de décision

Problème ou k-MST

Entrée : un graphe non orienté $G = (S, A)$,
une fonction de poids $\omega : E \rightarrow \mathbb{R}$,
un entier k tel que $1 < k \leq |V|$,
une valeur seuil $\Omega \in \mathbb{R}$.

Question : Existe-t-il un sous-graphe connexe et acyclique de G qui contient exactement k sommets et dont le poids total est au moins Ω ?

Pourquoi utiliser k-MST ?

- Extraire des sous-ensembles cohérents d'ingrédients
- acycliques : \emptyset redondances
- Permet de mettre en avant des « familles » d'ingrédients fortement liées

Étude de k-MST

Théorème 1.1

$k\text{-MST} \in \text{NP}$. Démonstration en annexe.

Théorème 1.2

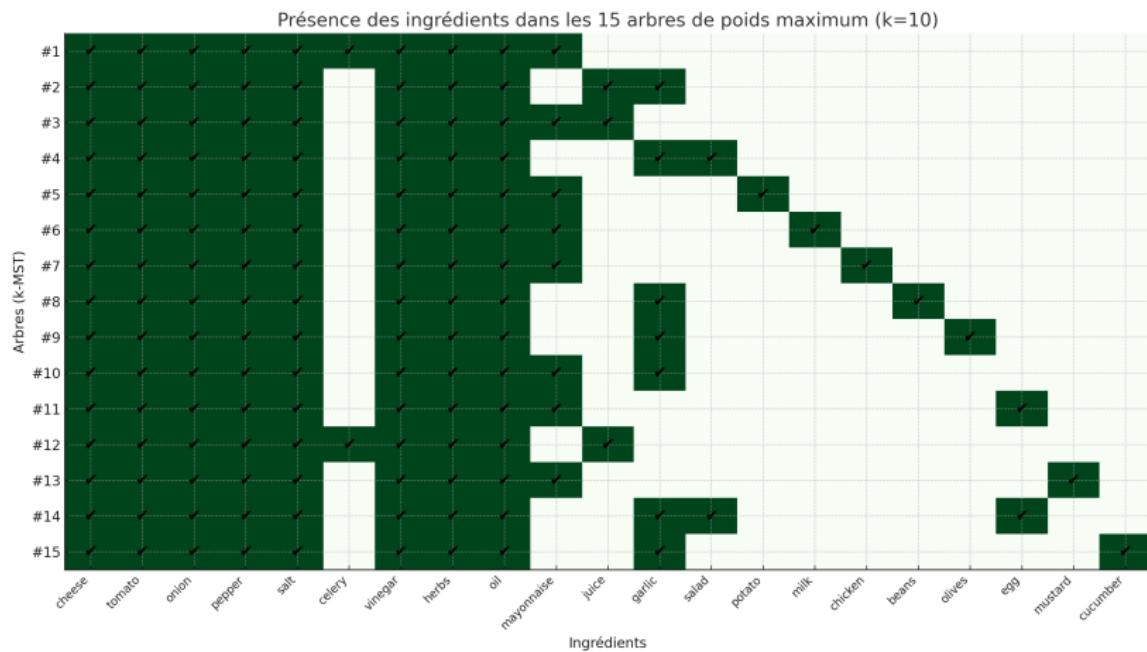
$k\text{-MST}$ est NP difficile. Lien vers démonstration en annexe.

Exemple de k-MST trouvé

Description :

- Sous-graphe avec $k = 15$ ingrédients
- Poids total élevé (forte co-occurrence)
- Ingrédients complémentaires dans une salade type

Exemple de k-MST trouvé



1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

PageRank

k-MST

R-edge-MWCS

PMI

5 Résultats

6 Annexe

Énoncé de R-edge-MWCS

Problème R-edge-MWCS

Entrée : un graphe non orienté $G = (S, A)$,
une fonction de poids $\omega : E \rightarrow \mathbb{R}$,
un ensemble de sommets imposés $R \subseteq S$,
une valeur seuil $\Omega \in \mathbb{R}$.

Question : Existe-t-il un sous-graphe connexe $G' = (S', A') \subseteq G$
tel que $R \subseteq S'$ et $\sum_{(u,v) \in A'} \omega(u, v) \geq \Omega$?

R-edge-MWCS : Intuition et objectifs

- Extension du k-MST permettant des sous-graphes cycliques
- Inclusion obligatoire de certains sommets (ex : ingrédients du placard)
- Maximiser le poids total tout en respectant la contrainte R
- Application : trouver des combinaisons d'ingrédients réalistes pour des recettes ciblées (ex : harissa + carotte)
- Problème NP-complet : solutions approximatives ou heuristiques nécessaires

Étude de ce problème

Théorème 2.1

R-edge-MWCS \in NP. Démonstration en annexe.

Théorème 2.2

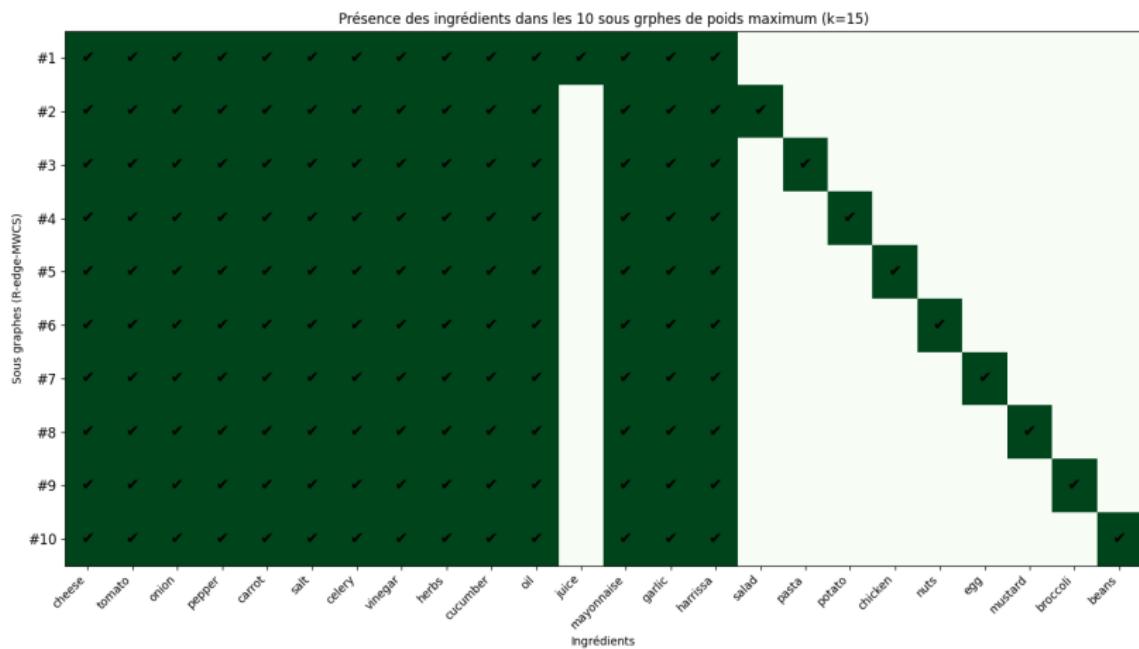
R-edge-MWCS est NP difficile. Lien vers démonstration en annexe.

Top 10 R-edge-MWCS : exemples d'arbres sélectionnés

- Recherche de sous-graphes connectés incluant les ingrédients imposés (ex : harissa, carotte)
- Résultats basés sur les poids de co-occurrence dans le graphe

Top 10 R-edge-MWCS : exemples d'arbres sélectionnés

Exemples de combinaisons d'ingrédients :



1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

PageRank

k-MST

R-edge-MWCS

PMI

5 Résultats

6 Annexe

PMI ?

- Recherche d'outils alternatifs à la simple co-occurrence
- Essai PMI (Pointwise Mutual Information) :

•

$$\text{PMI}(i,j) = \log \left(\frac{P(i,j)}{P(i) \cdot P(j)} \right)$$

- Exemple : "salade spirituelle"

1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

5 Résultats

6 Annexe

Validation pratique : réalisation d'une salade

- Réalisation concrète de la 9^e salade issue des résultats
- Photo de la salade (à insérer)
- Retour d'expérience :
- Conclusion ?

Photo des ingrédients de la 9^e salade réalisée



Photo de la 9^e salade réalisée



Synthèse et bilan

- Méthodes explorées : co-occurrences, PageRank, k-MST, R-edge-MWCS
- Résultats pratiques :
- Limites :
- Perspectives : amélioration des graphes, autres critères culinaires

Perspectives et travaux futurs

- filtrage plus fin, prise en compte de la quantité et du contexte
- Intégrer saveurs, textures, saisonnalité
- Étendre l'analyse à d'autres types de plats ou ingrédients

1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

5 Résultats

6 Annexe

Théorèmes de calculabilité

k-MST

R-edge-MWCS

1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

5 Résultats

6 Annexe

Théorèmes de calculabilité

k-MST

R-edge-MWCS

Appartenance à NP

Annexe th.1

Soient :

- un problème de décision f et E l'ensemble de ses instances :
 $f : E \rightarrow \mathbb{B}$;
- un ensemble \mathfrak{C} de certificats ;
- une fonction : $f_{\text{verif}} : E \times \mathfrak{C} \rightarrow \mathbb{B}$

Si $\forall e \in E, f(e) = \text{Vrai} \Leftrightarrow \exists c \in \mathfrak{C}, |c|$ est de taille polynomiale en $|e|$ tel que $f_{\text{verif}}(e, c) = \text{Vrai}$ et $f_{\text{verif}} \in P$

Alors $f \in NP$

Réduction polynomiale

Annexe def.1

Soient deux problèmes de décision :

- $f_1 : E_1 \rightarrow \mathbb{B}$ et $f_2 : E_2 \rightarrow \mathbb{B}$

Si $\exists g : E_1 \rightarrow E_2$ de complexité polynomiale en la taille de son entrée et $\forall e \in E_1$ telle que $f_1(e) = f_2(g(e))$

Alors f_1 se réduit polynomiallement à f_2 et on note $f_1 \leq_P f_2$

NP-difficile

Annexe th.2

Soit f un problème de décision ;

Si $\forall g \in NP, g \leq_P f$, alors $f \in NP$.

Théorème Annexe.3

Si $f_1 \leq_P f_2$ et si f_1 est NP-difficile, alors f_2 est NP-difficile.

1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

5 Résultats

6 Annexe

Théorèmes de calculabilité

k-MST

R-edge-MWCS

k-MST est dans NP. Théorème 1.1

Démonstration :

Ensemble des certificats :

$\mathfrak{C} : \{\text{Ensemble des listes chainées de couple d'entier}\}$

Soit un certificat $c \in \mathfrak{C}$ qui correspond à une instance positive de k-MST

Alors c'est au pire une liste chainée de $|A|$ éléments.

k-MST est dans NP. Théorème 1.1

Démonstration :

Structures :

- Couple et Couplage : liste chainée de couples.
- Graphe : graphe avec matrice d'adjacence.
- Méthodes standard associées.

k-MST est dans NP. Théorème 1.1

Démonstration :

Avec n la taille de l'entrée.

- dfs en $O(n^2)$
- vérification connexe acyclique en $O(n^2)$
- compte sommets en $O(n)$
- vérification du poids en $O(n)$

k-MST est dans NP. Théorème 1.1

Démonstration :

Fonction de vérification totale en $O(n^2)$

Donc la vérification du certificat se fait en temps polynomial.

Conclusion

Par Annexe th.1, k-MST est dans NP.

k-MST est NP difficile. Théorème 1.2

Giulia Galbiati, Angelo Morzenti, Francesco Maffioli, On the approximability of some maximum spanning tree problems, Theoretical Computer Science, Volume 181, Issue 1, 1997, Pages 107-118, ISSN 0304-3975,
[https://doi.org/10.1016/S0304-3975\(96\)00265-4](https://doi.org/10.1016/S0304-3975(96)00265-4).

1 Introduction

2 Début de l'analyse

3 Graphe de co-occurrences

4 Analyse de données

5 Résultats

6 Annexe

Théorèmes de calculabilité

k-MST

R-edge-MWCS

R-edge-MWCS est dans NP. Théorème 2.1

Structures et fonctions :

Programme `fverifrmwcs.c` simulant une vérification polynomiale :

- **Structures :**
 - graphe : matrice d'adjacence et des poids.
 - couple : deux sommets représentant une arête.
 - couplage : liste chaînée d'arêtes (le certificat).
- **Fonctions :**
 - `est_coupleage_du_graphe` : vérifie que les arêtes sont valides.
 - `est_connexe` : test DFS de connexité.
 - `contient_terminaux` : tous les sommets imposés sont présents.
 - `verif_poids` : poids total supérieur à Ω .

R-edge-MWCS est dans NP. Théorème 2.1

Complexité des fonctions :

- DFS pour connexité : $O(n^2)$
- Vérification des arêtes valides : $O(n^2)$
- Vérification des sommets imposés : $O(n)$
- Somme des poids : $O(n)$

Conclusion :

Certificat de taille polynomiale.

Vérification en temps polynomial \Rightarrow R-edge-MWCS \in NP.

R-edge-MWCS est NP difficile. Théorème 2.2

adatable facilement à partir de : (généralisation) El-Kebir,
Mohammed, and Gunnar W. Klau.

"Solving the Maximum-Weight Connected Subgraph Problem to
Optimality."

DIMACS Workshop on Algorithmic Challenges in Computational
Biology, 2012.

<https://dimacs11.zib.de/workshop/ElKebirKlau.pdf>.

R-edge-MWCS

code

```
import networkx as nx
import matplotlib.pyplot as plt
import sqlite3 as sql
import winsound
import time
import threading
import keyboard
import pickle
import base
import json

class Graphe:
    def __init__(self):
        self.graph = nx.Graph()

    def ajouter_sommet(self, sommet):
        """Ajoute un sommet au graphe."""
        self.graph.add_node(sommet)

    def ajouter_arrete(self, sommet1, sommet2, poids=None):
        """Ajoute une arête entre deux sommets. Un poids peut être ajouté."""
        self.graph.add_edge(sommet1, sommet2, weight=poids)

    def existe_arrete(self, sommet1, sommet2):
        """Vérifie si une arête existe entre deux sommets."""
        return self.graph.has_edge(sommet1, sommet2)

    def poids_arrete(self, sommet1, sommet2):
        """Recupère le poids d'une arête"""
        return self.graph.get_edge_data(sommet1, sommet2)['weight']
```

R-edge-MWCS

code

```
def incrementer_poids_arrete(self, sommet1, sommet2, pas = 1):
    if self.graph.has_edge(sommet1, sommet2):
        self.graph[sommet1][sommet2]['weight'] = self.graph[sommet1][sommet2]['weight'] + pas

def afficher_graphe(self):
    """Affiche le graphe en utilisant matplotlib."""
    pos = nx.spring_layout(self.graph) # Positionnement automatique des sommets
    weights = nx.get_edge_attributes(self.graph, 'weight')

    nx.draw(self.graph, pos, with_labels=True, node_color='skyblue', node_size=3000, font_size=10, font_weight='bold')
    if weights:
        nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=weights)

    plt.title("Représentation du Graphe")
    plt.show()

def beep():
    """Émet un bip en boucle jusqu'à ce que l'utilisateur appuie sur une touche."""
    while not keyboard.is_pressed("enter"): # Arrêter en appuyant sur "Enter"
        winsound.Beep(1000, 500) # Bip à 1000 Hz pendant 500 ms
        time.sleep(0.5)
```

R-edge-MWCS

code

```
db = str

def make_Vgs_grph(d:db, test=False, show = True) ->Graphe:
    cnect = sql.connect(d)
    cur = cnect.cursor()
    if test:
        txttest = "test"
    else:
        txttest = ""

    cur.execute("Select max(id) from ingred{}".format(txttest))
    n_ingred = cur.fetchall()[0][0]
    l_ingred = [i for i in range(1, n_ingred+1)]
    g = Graphe()
    for i in l_ingred:
        g.ajouter_sommet(i)

    print("Sommet ok")
    cur.execute("select * from jointab{}".format(txttest))
    print("Select all ok")
    data = cur.fetchall()
    n_data = len(data)
    ci = 0
    start = time.time()
    acttime = start
    print("c'est parti")
    print("\n data:", n_data)
    for i in range(n_data):
        if (ci == 100):
            if (ci == 100):
                ci = 0
                print(i, time.time()-acttime)
```

R-edge-MWCS

code

```
acttime = time.time()
ci+=1
for j in range(i+1, n_data):
    r1, i1 = data[i]
    r2, i2 = data[j]
    if r1 == r2:
        if g.existe_arrete(i1, i2):
            g.incrementer_poids_arrete(i1, i2)
        else:
            g.ajouter_arrete(i1, i2, 1)
print("arrete ok")
print("Final time:", time.time()-start)
bip_thread = threading.Thread(target=beep, daemon=True)
bip_thread.start()
keyboard.wait("enter")
if (show):
    print("graphe:")
    g.afficher_graphe()
with open("graphe{}.bin".format(txttest), "wb") as f:
    pickle.dump(g, f)
print("ok")
cnect.close()
```

R-edge-MWCS

code

```
# === Charger le graphe depuis le fichier .bin ===
def charger_graphe(filepath: str):
    with open(filepath, "rb") as f:
        |   graphe = pickle.load(f)
    return graphe

# === Calculer le PageRank pondéré ===
def calculer_pagerank(graphe):
    |   return nx.pagerank(graphe.graph, weight='weight')

# === Charger la table ingred pour avoir les noms ===
def get_ingredient_names(db_path):
    conn = sqlite3.connect(db_path)
    cur = conn.cursor()
    cur.execute("SELECT id, name FROM ingred")
    id_to_name = {row[0]: row[1] for row in cur.fetchall()}
    conn.close()
    return id_to_name

# === Affichage du classement PageRank des ingrédients ===
def afficher_pagerank_top(graphe, db_path, top_n=20):
    pagerank_scores = calculer_pagerank(graphe)
    id_to_name = get_ingredient_names(db_path)

    sorted_pagerank = sorted(pagerank_scores.items(), key=lambda x: x[1], reverse=True)

    print(f"Top {top_n} ingrédients selon PageRank :\n")
    for ing_id, score in sorted_pagerank[:top_n]:
        name = id_to_name.get(ing_id, "UNKNOWN")
        print(f"\t{name:<20} : {score:.6f}")
```

R-edge-MWCS

code

```
class KMSTMaxTopM:  
    def __init__(self, graph: nx.Graph, k: int, m: int):  
        self.k = k  
        self.m = m  
        self.top_trees = [] # Min-heap: (poids, frozenset(somets))  
        self.tree_set = set() # Pour éviter les doublons  
        self.call_count = 0  
        self.last_log = time.time()  
  
    def add_solution(self, weight, tree):  
        tree_frozen = frozenset(tree)  
        if tree_frozen in self.tree_set:  
            return  
        self.tree_set.add(tree_frozen)  
        heapq.heappush(self.top_trees, (weight, tree_frozen))  
        if len(self.top_trees) > self.m:  
            _, removed = heapq.heappop(self.top_trees)  
            self.tree_set.remove(removed)  
  
    def dfs(self, current, visited, edge_sum, depth):  
        self.call_count += 1  
        if self.call_count % 100000 == 0:  
            now = time.time()  
            best = max(self.top_trees, default=(0, None))[0]  
            print(f"[{self.call_count} appels] Niveau {depth} | Sommet: {current} | Poids: {edge_sum} | BestMax: {best} | Δt: {now - self.last_log}s")  
            self.last_log = now  
  
        if len(visited) == self.k:  
            self.add_solution(edge_sum, visited.copy())  
            return  

```

R-edge-MWCS

code

```
        for neighbor, data in sorted(self.graph[current].items(), key=lambda x: -x[1]['weight']):
            if neighbor in visited:
                continue
            max_edge = data['weight']
            projected = edge_sum + max_edge + (self.k - len(visited) - 1) * max_edge
            if self.top_trees and projected <= self.top_trees[0][0]:
                continue
            visited.add(neighbor)
            self.dfs(neighbor, visited, edge_sum + data['weight'], depth + 1)
            visited.remove(neighbor)

    def run(self):
        for node in self.graph.nodes:
            self.dfs(node, {node}, 0, 1)
        return sorted(self.top_trees, reverse=True)
```

R-edge-MWCS

code

```
class EdgeRMWCSStopN:
    def __init__(self, graph: nx.Graph, k: int, m: int, required_nodes: set):
        self.graph = graph
        self.k = k
        self.m = m
        self.required_nodes = required_nodes
        self.top_solutions = [] # Min-heap: (weight_sum, frozenset(nodes))
        self.solution_set = set()
        self.call_count = 0
        self.last_log = time.time()

    def add_solution(self, weight, nodes):
        frozen = frozenset(nodes)
        if frozen in self.solution_set:
            return
        self.solution_set.add(frozen)
        heapq.heappush(self.top_solutions, (weight, frozen))
        if len(self.top_solutions) > self.m:
            _, removed = heapq.heappop(self.top_solutions)
            self.solution_set.remove(removed)

    def dfs(self, current, visited, weight_sum, depth):
        self.call_count += 1
        if self.call_count % 100_000 == 0:
            now = time.time()
            best = max(self.top_solutions, default=(0, None))[0]
            print(f"[{self.call_count} calls] Depth: {depth} | Node: {current} | Weight: {weight_sum:.2f} | BestMax: {best:.2f} | Δt: {(now - self.last_log):.2f}")
            self.last_log = now

        for neighbor in self.graph.neighbors(current):
            if neighbor not in visited:
                visited.add(neighbor)
                self.dfs(neighbor, visited, weight_sum + 1, depth + 1)
```

R-edge-MWCS

code

```
if len(visited) == self.k:
    if self.required_nodes.issubset(visited):
        self.add_solution(weight_sum, visited.copy())
    return

for neighbor in sorted(self.graph.neighbors(current), key=lambda n: -self.graph[current][n]['weight']):
    if neighbor in visited:
        continue

    edge_weight = self.graph[current][neighbor]['weight']
    projected_weight = weight_sum + edge_weight + (self.k - len(visited) - 1) * edge_weight

    if self.top_solutions and projected_weight <= self.top_solutions[0][0]:
        continue

    visited.add(neighbor)
    self.dfs(neighbor, visited, weight_sum + edge_weight, depth + 1)
    visited.remove(neighbor)

def run(self):
    start_nodes = self.required_nodes if self.required_nodes else self.graph.nodes
    for start in start_nodes:
        self.dfs(start, {start}, 0, 1)
    return sorted(self.top_solutions, reverse=True)
```

R-edge-MWCS

code

```
import sqlite3 as sql
import json
import math
import ast # Import the ast module

db = str

def list_ingred(d:db)->list[str]:
    cnct = sql.connect(d)
    cur = cnct.cursor()
    cur.execute("select NER from test order by field1")
    data = cur.fetchall()
    cnct.close()
    # Changed json.loads to ast.literal_eval
    return list(dict.fromkeys(ing for tup in data for ing in ast.literal_eval(tup[0])))
```

R-edge-MWCS

code

```
# Récupérer toutes les recettes et leurs ingrédients NER
cursor.execute("SELECT field1, NER FROM test")
recettes = cursor.fetchall()

# Créer des listes pour les insertions en masse
ingredients_to_insert = []
jointab_to_insert = set() # Utiliser un set pour éviter les doublons

# Traitement des recettes sans calculs de progression
for i, recette in enumerate(recettes):
    field1 = recette[0]
    # Changed json.loads to ast.literal_eval
    ner_ingredients = ast.literal_eval(recette[1])

    for ingredient in ner_ingredients:
        # Ajouter l'ingrédient à la liste pour insertion en masse
        ingredients_to_insert.append((ingredient,))

        # Ajouter l'association (recipe_id, ingredient) dans un set pour éviter les doublons
        jointab_to_insert.add((field1, ingredient))

# Insertion des ingrédients en masse
cursor.executemany("INSERT OR IGNORE INTO ingredtest (name) VALUES (?)", ingredients_to_insert)

# Récupérer les IDs des ingrédients nouvellement insérés
cursor.execute("SELECT id, name FROM ingredtest")
ingredient_ids = {name: id for id, name in cursor.fetchall()}

# Construire la liste des insertions pour jointab, avec les IDs des ingrédients
jointab_to_insert = [(field1, ingredient_ids[ingredient]) for field1, ingredient in jointab_to_insert]
```

R-edge-MWCS

code

```
# Insertion des associations (jointab) en masse
cursor.executemany("INSERT OR IGNORE INTO jointabtest (recipe_id, ingredient_id) VALUES (?, ?)", jointab_to_insert)

# Valider les changements et fermer la connexion
conn.commit()
conn.close()

print("Traitement terminé.")

def buildPMI(d: str):
    conn = sql.connect(d)
    cursor = conn.cursor()

    cursor.execute('''
        CREATE TABLE IF NOT EXISTS PMI(
            id INTEGER,
            ext_id INTEGER,
            pmi REAL
        );
    ''')
    ...

    # Nombre total de recettes (DISTINCT est préférable)
    cursor.execute("SELECT COUNT(DISTINCT recipe_id) FROM jointabtest")
    n_recipes = cursor.fetchone()[0]

    # Probabilités individuelles P(i)
    cursor.execute(f"""
        SELECT ingredient_id, COUNT(ingredient_id) * 1.0 / {n_recipes}
        FROM jointab
    """)
```

R-edge-MWCS

code

```
        GROUP BY ingredient_id
    """
)
# Transforme en dict pour accès rapide
self_probas = dict(cursor.fetchall())

# Probabilités conjointes  $P(i, j)$ 
cursor.execute(f"""
    SELECT
        j1.ingredient_id AS i,
        j2.ingredient_id AS j,
        COUNT(DISTINCT j1.recipe_id) * 1.0 / {n_recipes} AS pij
    FROM jointab j1
    JOIN jointab j2
        ON j1.recipe_id = j2.recipe_id AND j1.ingredient_id < j2.ingredient_id
    GROUP BY i, j
""")
ext_probas = cursor.fetchall()

for i, j, pij in ext_probas:
    pi = self_probas.get(i)
    pj = self_probas.get(j)

    if pi and pj and pij > 0:
        pmi = math.log(pij / (pi * pj))
        cursor.execute(
            "INSERT INTO PMI (id, ext_id, pmi) VALUES (?, ?, ?)",
            (i, j, pmi)
        )
```

R-edge-MWCS

code

```
conn.commit()
conn.close()

d = "C:/Users/xavie/Documents/Etudes/TIPE/Salad/main.db"

def get_ingred(id):
    conn = sql.connect(d)
    cur = conn.cursor()

    cur.execute("SELECT name FROM ingred WHERE id={}".format(id))
    ingred = cur.fetchone()[0]

    conn.close()
    return ingred

import sqlite3
import itertools

def count_cooccurrences(db_path: str):
    """
    Counts the co-occurrences of each ingredient pair across all recipes
    and stores them in a new table named 'cooc'.

    Args:
        db_path (str): The path to the SQLite database file (e.g., 'main.db').
    """
    conn = None
    try:
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
```

R-edge-MWCS

code

```
# 1. Create the 'cooc' table if it doesn't exist
cursor.execute("""
    CREATE TABLE IF NOT EXISTS cooc (
        id INTEGER NOT NULL,
        ext_id INTEGER NOT NULL,
        cooc_count INTEGER NOT NULL,
        PRIMARY KEY (id, ext_id),
        FOREIGN KEY (id) REFERENCES ingred(id),
        FOREIGN KEY (ext_id) REFERENCES ingred(id)
    );
""")
# Add indexes for efficient lookups
cursor.execute("CREATE INDEX IF NOT EXISTS idx_cooc_id ON cooc (id);")
cursor.execute("CREATE INDEX IF NOT EXISTS idx_cooc_ext_id ON cooc (ext_id);")
conn.commit()
print("Table 'cooc' created or already exists.")

# 2. Fetch all recipe-ingredient relationships
# This query gets all (recipe_id, ingredient_id) pairs, ordered by recipe_id
cursor.execute("SELECT recipe_id, ingredient_id FROM jointab ORDER BY recipe_id;")
recipe_ingredient_data = cursor.fetchall()

# 3. Group ingredients by recipe
recipes_ingredients = {}
for recipe_id, ingred_id in recipe_ingredient_data:
    if recipe_id not in recipes_ingredients:
        recipes_ingredients[recipe_id] = []
    recipes_ingredients[recipe_id].append(ingred_id)

# 4. Calculate co-occurrences
cooccurrence_counts = {}
```

R-edge-MWCS

code

```
for recipe_id, ingredient_list in recipes_ingredients.items():
    # Get all unique pairs of ingredients within the current recipe
    # A recipe must have at least 2 ingredients to form a pair
    if len(ingredient_list) >= 2:
        for ing1, ing2 in itertools.combinations(sorted(ingredient_list), 2): # Sort to ensure canonical order of pairs
            # Ensure canonical order for the pair (smaller ID first)
            pair = (ing1, ing2) # Already sorted by itertools.combinations(sorted(List), 2)

            if pair not in cooccurrence_counts:
                cooccurrence_counts[pair] = 0
                cooccurrence_counts[pair] += 1

    # 5. Prepare data for batch insertion
    data_to_insert = []
    for (id1, id2), count in cooccurrence_counts.items():
        data_to_insert.append((id1, id2, count))

    # 6. Insert/Update into 'cooc' table
    # Clear existing data in 'cooc' table before inserting new counts
    cursor.execute("DELETE FROM cooc;")
    conn.commit()
    print("Existing data in 'cooc' table cleared.")

    # Insert new co-occurrence counts
    cursor.executemany("INSERT INTO cooc (id, ext_id, cooc_count) VALUES (?, ?, ?)", data_to_insert)
    conn.commit()
    print(f"Successfully counted and inserted {len(data_to_insert)} co-occurrence pairs into 'cooc' table.")
```

R-edge-MWCS

code

```
1 ˜ #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  int sup_int = 1000000;
6
7 ˜ struct couple
8  {
9      int a;
10     int b;
11 };
12 typedef struct couple couple;
13
14 ˜ struct couplage
15  {
16     couple val;
17     struct couplage* next;
18 };
19 typedef struct couplage couplage;
20
21 ˜ struct graphe
22  {
23     int n;
24     int** adj;
25     int** W;
26 };
27 typedef struct graphe graphe;
28
29 ˜ graphe* init_graphe(int n){
30     graphe* res = malloc(sizeof(graphe));
```

R-edge-MWCS

code

```
33     res->W = malloc(n*sizeof(int));
34     for (int i = 0; i < n; i++)
35     {
36         res->adj[i] = malloc(n*sizeof(int));
37         res->W[i] = malloc(n*sizeof(int));
38         for (int j = 0; j < n; j++)
39         {
40             res->adj[i][j] = 0;
41             res->W[i][j] = sup_int;
42         }
43     }
44     return res;
45 }
46
47 void free_graph(graphe* g){
48     for (int i = 0; i < g->n; i++)
49     {
50         free(g->adj[i]);
51     }
52     free(g->adj);
53     free(g);
54 }
55
56 void add_edge(graphe* g, int u, int v, int w){
57     g->adj[u][v] = 1;
58     g->adj[v][u] = 1;
59     g->W[u][v] = w;
60     g->W[v][u] = w;
61 }
```

R-edge-MWCS

code

```
63 void del_edge(graphe* g, int u, int v){  
64     g->adj[u][v] = 0;  
65     g->adj[v][u] = 0;  
66 }  
67  
68 couple init_couple(int a, int b){  
69     couple res;  
70     res.a = a;  
71     res.b = b;  
72     return res;  
73 }  
74  
75 void free_couplage(couplage* c){  
76     if (c == NULL) return;  
77     free_couplage(c->next);  
78     free(c);  
79 }  
80  
81  
82 couplage* append_couplage(couplage* c, couple a){  
83     couplage* new = malloc(sizeof(couplage));  
84     new->val = a;  
85     new->next = c;  
86     return new;  
87 }
```

R-edge-MWCS

code

```
89 bool est_coupleage_du_graphe(graphe* g, coupleage* c){
90     for (coupleage* actc = c; actc != NULL; actc = actc->next)
91     {
92         if (g->adj[actc->val.a][actc->val.b] != 1){
93             return false;
94         }
95     }
96     return true;
97 }
98
99 void dfs_connexe_acyclique(int node, int parent, bool* visited, int** adj, int max, bool* cycle) {
100    visited[node] = true;
101    for (int v = 0; v < max; v++) {
102        if (adj[node][v]) {
103            if (!visited[v]) {
104                dfs_connexe_acyclique(v, node, visited, adj, max, cycle);
105            } else if (v != parent) {
106                *cycle = true;
107            }
108        }
109    }
110 }
```

R-edge-MWCS

code

```
112 bool est_connexe_et_acyclique(couplage* c) {
113     if (c == NULL) return true;
114     int max = 0;
115     for (couplage* actc = c; actc != NULL; actc = actc->next) {
116         if (actc->val.a > max) max = actc->val.a;
117         if (actc->val.b > max) max = actc->val.b;
118     }
119     max += 1;
120     int** adj = malloc(max * sizeof(int*));
121     for (int i = 0; i < max; i++) {
122         adj[i] = calloc(max, sizeof(int));
123     }
124     bool* used = calloc(max, sizeof(bool));
125     for (couplage* actc = c; actc != NULL; actc = actc->next) {
126         int a = actc->val.a;
127         int b = actc->val.b;
128         adj[a][b] = 1;
129         adj[b][a] = 1;
130         used[a] = true;
131         used[b] = true;
132     }
133     bool* visited = calloc(max, sizeof(bool));
134     bool cycle = false;
135     int start = -1;
136     for (int i = 0; i < max; i++) {
137         if (used[i]) {
138             start = i;
139             break;
140         }
141     }
```

R-edge-MWCS

code

```
142     dfs_connexe_acyclique(start, -1, visited, adj, max, &cycle);
143     for (int i = 0; i < max; i++) {
144         if (used[i] && !visited[i]) {
145             cycle = true;
146             break;
147         }
148     }
149     for (int i = 0; i < max; i++) {
150         free(adj[i]);
151     }
152     free(adj);
153     free(used);
154     free(visited);
155     return !cycle;
156 }
157
158 int compte_sommet(couplage* c, int n){
159     int res = 0;
160     int* deja_vu = malloc(n*sizeof(int));
161     for (int i = 0; i < n; i++)
162     {
163         deja_vu[i] = 0;
164     }
165
166 }
```

code

```

167 ~     for (couplage* actc = c; actc != NULL; actc = actc->next)
168 ~     {
169 ~         if (deja_vu[actc->val.a] == 0){
170 ~             res++;
171 ~             deja_vu[actc->val.a] = 1;
172 ~         }
173 ~         if (deja_vu[actc->val.b] == 0){
174 ~             res++;
175 ~             deja_vu[actc->val.b] = 1;
176 ~         }
177 ~     }
178 ~     free(deja_vu);
179 ~     return res;
180 ~ }
181
182 ~ bool verif_poids(int** W, couplage* c, int w){
183 ~     int cmt = 0;
184 ~     for (couplage* actc = c; actc != NULL; actc = actc->next)
185 ~     {
186 ~         cmt += W[actc->val.a][actc->val.b];
187 ~     }
188 ~     return cmt >= w;
189 ~ }
190
191 ~ bool verif(graphe* g, couplage*c, int k, int w){
192 ~     return est_couplege_du_graphe(g,c) && est_connexe_et_acyclique(c) && compte_sommet(c, g->n) == k && verif_poids(g->W, c, w);
193 ~ }
```

R-edge-MWCS

code

```
195 bool contient_terminaux(couplage* c, int* terminaux, int nb_term, int n) {
196     int* vus = calloc(n, sizeof(int));
197     for (couplage* actc = c; actc != NULL; actc = actc->next) {
198         vus[actc->val.a] = 1;
199         vus[actc->val.b] = 1;
200     }
201     for (int i = 0; i < nb_term; i++) {
202         if (!vus[terminaux[i]]) {
203             free(vus);
204             return false;
205         }
206     }
207     free(vus);
208     return true;
209 }
210
211 void dfs_connexe(int node, bool* visited, int** adj, int max) {
212     visited[node] = true;
213     for (int v = 0; v < max; v++) {
214         if (adj[node][v] && !visited[v]) {
215             dfs_connexe(v, visited, adj, max);
216         }
217     }
218 }
```

R-edge-MWCS

code

```

253     dfs_connexe(start, visited, adj, max);
254
255     for (int i = 0; i < max; i++) {
256         if (used[i] && !visited[i]) {
257             for (int j = 0; j < max; j++) free(adj[j]);
258             free(adj);
259             free(used);
260             free(visited);
261             return false;
262         }
263     }
264
265     for (int i = 0; i < max; i++) free(adj[i]);
266     free(adj);
267     free(used);
268     free(visited);
269     return true;
270 }
271
272 bool verif_edge_rmwcs(graphe* g, couplage* c, int poids_min, int* terminaux, int nb_term) {
273     return est_coupleage_du_graphe(g, c)
274     && est_connexe(c)
275     && contient_terminaux(c, terminaux, nb_term, g->n)
276     && verif_poids(g->W, c, poids_min);
277 }
278
279

```

R-edge-MWCS

code

```
280 int main(void){  
281     int n = 6;  
282     graphe* g = init_graphe(n);  
283     add_edge(g, 1, 2, 2);  
284     add_edge(g, 3, 2, 3);  
285     add_edge(g, 1, 3, 1);  
286     add_edge(g, 3, 5, 2);  
287     add_edge(g, 1, 4, 3);  
288     add_edge(g, 5, 4, 3);  
289     coupleage* c = NULL;  
290     c = append_coupleage(c, init_couple(1, 2));  
291     c = append_coupleage(c, init_couple(3, 2));  
292     c = append_coupleage(c, init_couple(3, 5));  
293  
294     if (verif(g,c,4,6)) {  
295         printf("C'est un arbre à 5 sommets\n");  
296     } else {  
297         printf("Ce n'est pas un arbre à 5 sommets\n");  
298     }  
299  
300     c = NULL;  
301     c = append_coupleage(c, init_couple(1, 2));  
302     c = append_coupleage(c, init_couple(3, 2));  
303     c = append_coupleage(c, init_couple(3, 5));  
304  
305     int terminaux[2] = {1, 5}; // sommets imposés  
306     int seuil = 6;  
307  
308     if (verif_edge_rmwcs(g, c, seuil, terminaux, 2)) {  
309         printf("Certificat valide pour edge-r-MWCS\n");  
310     } else {
```

R-edge-MWCS

code

```
310 } else {
311     printf("Certificat invalide\n");
312 }
313
314 c = NULL;
315 c = append_couple(c, init_couple(1, 2));
316 c = append_couple(c, init_couple(3, 2));
317
318 if (verif_edge_rmwcs(g, c, seuil, terminaux, 2)) {
319     printf("Certificat valide pour edge-r-MWCS\n");
320 } else {
321     printf("Certificat invalide\n");
322 }
323
324 c = NULL;
325 c = append_couple(c, init_couple(1, 2));
326 c = append_couple(c, init_couple(4, 5)); // déconnecté de (1,2)
327 if (verif_edge_rmwcs(g, c, seuil, terminaux, 2)) {
328     printf("Certificat valide pour edge-r-MWCS\n");
329 } else {
330     printf("Certificat invalide\n");
331 }
332
333 free_couple(c);
334 free_graph(g);
335 return 0;
336 }
```