

ALGORITHME COMBINATOIRE DE SELECTION DES PROFILS : MCAP+TOPSIS+HONGROIS

Par : Abdel YEZZA, Ph.D

Date : octobre 2025

Sommaire

🚀 Système de sélection de profil TOPSIS.....	2
🚀 Aperçu	3
🚀 Schéma fonctionnel général	3
Scénario 1 : Classement des profils pour chaque activité	4
Using: Profile Assignment System (MCAP) + TOPSIS.....	4
Scénario 2 : Affectation individuelle optimale.....	6
Utilisation : MCAP + TOPSIS + Méthode hongroise.....	6
Using: MCAP + TOPSIS + Hungarian Method	6
⌚ Principales différences Entre les scénarios.....	7
✳ Comment les algorithmes se complètent.....	7
⌚ Fonctionnalités.....	7
🚀 Installation	8
Prérequis	8
Installer les dépendances	8
Démarrage rapide	8
1. Utilisation de base (configuration par défaut)	8
2. Seuil personnalisé	8
3. Traitement d'activité unique	8
4. Mode verbeux avec visualisations.....	8
5. Stratégie de poids personnalisée	8
6. Fichiers d'entrée personnalisés	8
Configuration.....	9
Format des données d'entrée	10
Profils CSV (data/input/profiles.csv)	10
Activités CSV (data/input/activities.csv).....	10
🚀 Comment ça marche	11
1. Transformation des compétences	11
2. Algorithme TOPSIS	11



3. Classement	11
Exemple de sortie	11
4. Options de ligne de commande	13
5. Stratégies des poids	14
1. Poids uniformes (par défaut)	14
2. Pondérations basées sur les besoins	14
3. Poids personnalisés	14
6. Formules TOPSIS.....	14
Formule standard (par défaut).....	14
Formule variante	14
7. Exemples d'utilisation.....	15
Exemple 1 : Trouver le meilleur développeur back-end	15
Exemple 2 : Ajuster le seuil pour différentes normes	15
Exemple 3 : Générer un rapport complet avec des visualisations	15
Utilisation de l'API (Python).....	15
Exemples réels	16
Exemple 1 : TOPSIS standard	16
Exemple 2 : variante TOPSIS	18
Exemple 3 : contrainte d'affectation HONGROISE (affectation 1 à 1) et formule de proximité = variante	19
Références.....	21
Algorithme TOPSIS.....	21
Multicritères Décision Analyse	21
Licence	21
Contribuer.....	21
Auteur.....	21

Système de sélection de profil TOPSIS

Un système complet d'évaluation et de classement des profils, mais pas seulement, combinant 3 algorithmes :

1. La méthode **de correspondance profil-activité**
2. L'algorithme **TOPSIS** (Technique de préférence d'ordre par similarité à la solution idéale)
3. En option, l' **algorithme HONGROIS**

avec des seuils de niveau de compétence configurables et de nombreuses autres options et fonctionnalités comme vous le découvrirez ci-dessous.

Auteur : Abdel YEZZA, Ph.D , 2025

Aperçu

Ce projet combine deux concepts puissants et éventuellement hongrois algorithme:

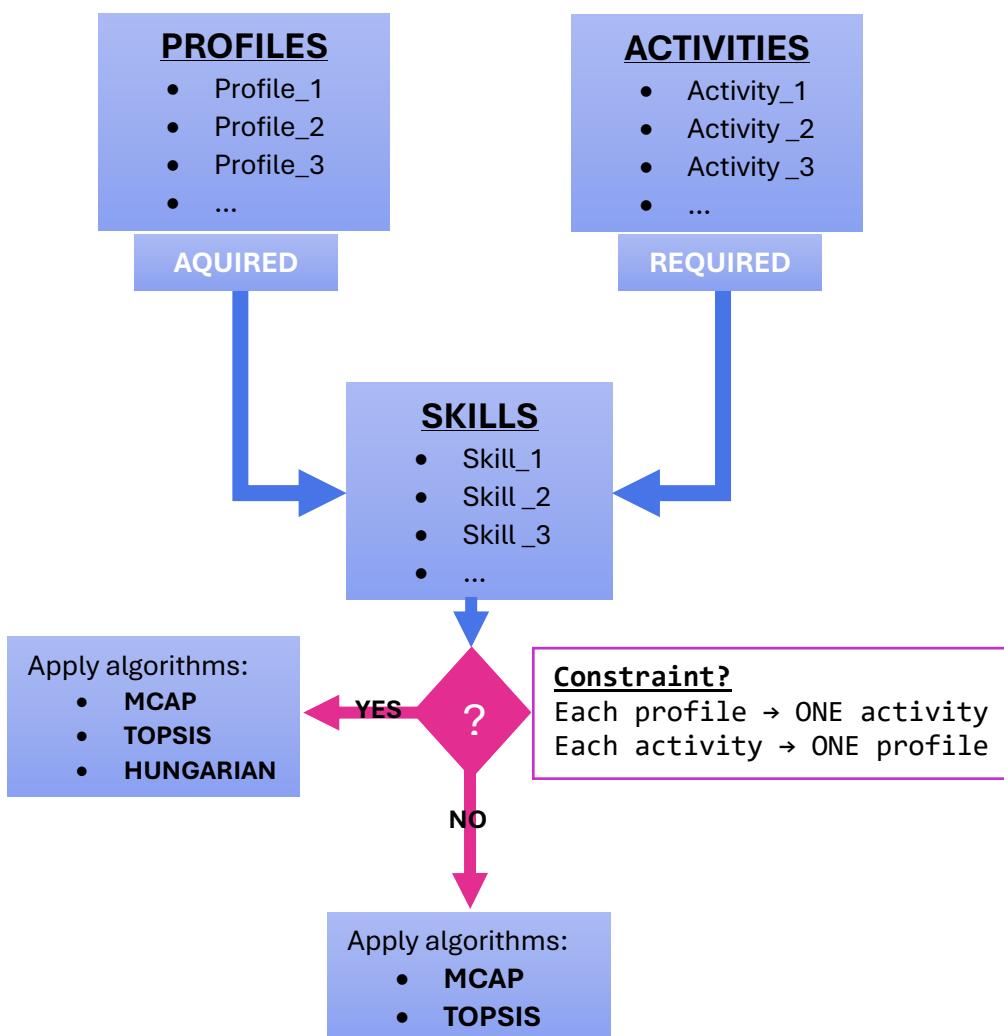
1. **Appariement Profil-Activité** : Appariement des profils aux activités en fonction des compétences - voir mon article : [Système d'attribution de profils \(MCAP\)](#) - Abdel YEZZA (Ph.D), 2025
2. **Algorithme TOPSIS** - Analyse de décision multicritère pour le classement - voir mon article : [Implémentation de l'algorithme TOPSIS](#) - Abdel YEZZA (Ph.D), 2025

Le système utilise un **seuil configurable** pour déterminer comment les compétences sont évaluées :

- **Compétences \geq Seuil** : Considérées comme des critères **bénéfiques** (plus le seuil est élevé, mieux c'est)
- **Compétences $<$ Seuil** : considérées comme des critères **non bénéfiques** (un seuil inférieur est acceptable)

Cette approche permet une évaluation flexible où certaines compétences sont essentielles (doivent être maximisées) tandis que d'autres sont facultatives (ne pénalisent pas le manque de maîtrise).

Schéma fonctionnel général

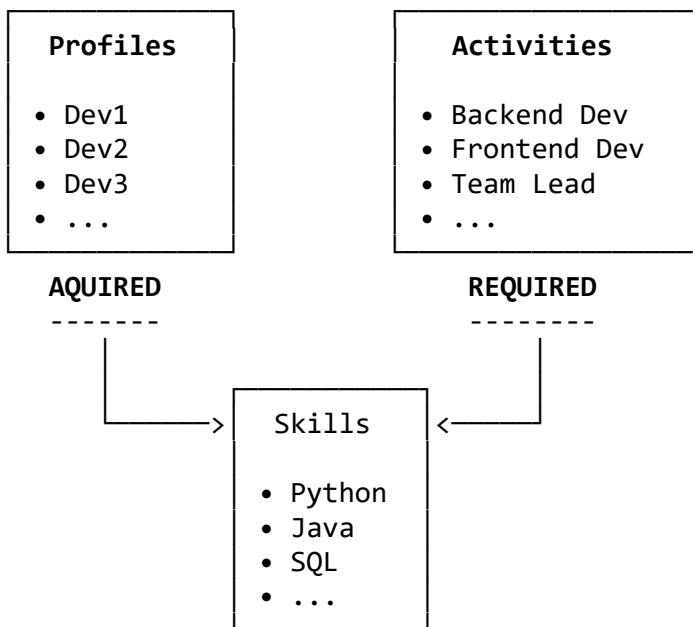


Illustrons ce schéma général par quelques scénarios :

Scénario 1 : Classement des profils pour chaque activité

Using: Profile Assignment System (MCAP) + TOPSIS

INPUT:



STEP 1: Profile Assignment System (MCAP)

Skill Transformation Based on Threshold

For each activity requirement:

- If skill level \geq Threshold (e.g., 3.0)
 - Mark as BENEFICIAL (higher is better)
- If skill level $<$ Threshold
 - Mark as NON-BENEFICIAL (lower is acceptable)

Example: Backend Dev needs Python=5, Leadership=2

With threshold=3.0:

- Python (5 \geq 3): BENEFICIAL → maximize
- Leadership (2<3): NON-BENEFICIAL → minimize



STEP 2: TOPSIS Algorithm

Multi-Criteria Decision Analysis

1. Normalize profile skills (make comparable)
2. Apply weights (importance of each skill)
3. Find IDEAL profile (best in all skills)
4. Find WORST profile (worst in all skills)
5. Calculate how close each profile is to IDEAL

Result: Proximity Score (0 to 1)

- 1.0 = Perfect match
- 0.0 = Worst match

**OUTPUT: Ranked List for Each Activity**

Backend Development:

1. Dev10 (Score: 0.95) ★ Best match
2. Dev7 (Score: 0.87)
3. Dev1 (Score: 0.82)
- ...

Frontend Development:

1. Dev2 (Score: 0.91) ★ Best match
2. Dev5 (Score: 0.85)
- ...

Cas d'utilisation :

- Vous avez de multiples activités et souhaitez connaître les meilleurs candidats pour chacune
- Vous devez voir tous les profils qualifiés classés par adéquation
- Aide à la prise de décision lorsque vous avez de la flexibilité dans les tâches

Scénario 2 : Affectation individuelle optimale

Utilisation : MCAP + TOPSIS + Méthode hongroise

Using: MCAP + TOPSIS + Hungarian Method

INPUT: Same as Scenario 1



STEPS 1-2: Same as Scenario 1 (MCAP + TOPSIS)

Create Compatibility Matrix

	Backend	Frontend	TeamLead
Dev1	0.82	0.91	0.65
Dev2	0.75	0.85	0.70
Dev3	0.90	0.60	0.88

Each cell = TOPSIS proximity score



STEP 3: Hungarian Algorithm (Optimal Assignment)

Find the BEST overall assignment

Constraint: Each profile → ONE activity
Each activity → ONE profile

Goal: Maximize total satisfaction

Algorithm finds optimal pairing considering all possibilities simultaneously



OUTPUT: Optimal Assignments

Optimal Assignments:

Dev1 → Frontend Development (0.91)
Dev2 → Team Lead (0.70)
Dev3 → Backend Development (0.90)

Total Score: 2.51 (best possible combination)

Cas d'utilisation :

- Vous devez attribuer exactement UN profil à chaque activité
- Aucun profil ne peut être attribué à plusieurs activités
- Vous souhaitez la solution optimale à l'échelle mondiale, pas seulement les meilleures solutions individuelles
- Exemple : Affecter des membres de l'équipe à des rôles de projet

Principales différences Entre les scénarios

Aspect	Scénario 1 (Classement)	Scénario 2 (affectation optimale)
Sortir	Liste classée par activité	Devoirs individuels
Flexibilité	Plusieurs candidats par activité	Un profil par activité
Optimisation algorithmes	Niveau d'activité individuel MCAP + TOPSIS	Niveau mondial MCAP + TOPSIS + Hongrois
Quand l'utiliser	Explorer les options de flexibilité	Travaux finaux

Comment les algorithmes se complètent

MCAP (Profile Assignment System)

- Transforms requirements based on threshold
- Identifies what matters most for each activity
- Provides context for evaluation



TOPSIS (Multi-Criteria Decision)

- Evaluates profiles against ideal solution
- Generates compatibility scores
- Ranks profiles for each activity



HUNGARIAN (Optional - For Optimal Assignment)

- Uses TOPSIS scores as input
- Finds globally optimal assignments
- Ensures no conflicts (1 profile = 1 activity)

Fonctionnalités

- **Système de seuil configurable :** (*min_threshold* à *max_threshold* avec valeurs de seuil) - voir **le fichier config.json** à la clé **threshold_settings**
- **Stratégies de pondération multiples :** (uniforme, basée sur les exigences, personnalisée)
- **Deux formules de proximité TOPSIS :** (*standard* et *variante*) - le cas *standard* utilise les formules standard TOPSIS, tandis que la *variante* utilise une fonction différente déjà normalisée et plus disparate que la norme
- **Visualisations complètes :** (cartes thermiques , graphiques à barres , tracés radar et analyse de distance)
- Le graphique **de la carte thermique** fournit des valeurs de classement et de proximité claires et complètes permettant de choisir parmi les profils les mieux classés.
- **Formats d'entrée flexibles :** (fichiers CSV avec profils et activités)
- **Interface de ligne de commande :** avec des options simples et étendues
- **Export des résultats détaillés :** (classements, matrices, rapports d'analyse)

Installation

Prérequis

- Python 3.8 ou supérieur
- gestionnaire de paquets pip

Installer les dépendances

```
cd . \ topsis_profiles_selection
pip install -r exigences.txt
```

Démarrage rapide

1. Utilisation de base (configuration par défaut)

```
python main.py
```

Cela va:

1. Charger des profils et des activités à partir de **données/entrées/**
2. Utiliser le seuil = 3,0 (configurable entre 0 et 5) par défaut ou des valeurs personnalisées
3. Appliquer des pondérations uniformes à toutes les compétences par défaut
4. Générer des classements pour toutes les activités
5. Enregistrer les résultats dans **les données/sortie/**

2. Seuil personnalisé

```
python main.py --seuil 3.5
```

3. Traitement d'activité unique

```
python main.py --activity " Développement_Backend "
```

4. Mode verbeux avec visualisations

```
python main.py -v --viz
```

5. Stratégie de poids personnalisée

```
python main.py --weight-strategy basé sur les exigences
```

6. Fichiers d'entrée personnalisés

```
python main.py --profiles données/entrée/profils.csv --activités données/entrée/activités.csv
```

Configuration

Cette option permet de personnaliser tous les paramètres d'entrée dans un fichier de configuration JSON. Vous n'avez pas besoin de mémoriser toutes les options de ligne de commande.

Modifiez **config.json** pour personnaliser :

```
{
  "projet_name" : "Système de sélection de profils TOPSIS" ,
  "description" : "Sélection de profil à l'aide de l'algorithme TOPSIS avec seuil de compétence configurable",
  "version" : "1.0.0" ,
  "auteur" : "Abdel YEZZA ( Ph.D )" ,

  "données" : {
    "profils_file" : "data/input/profiles.csv" ,
    "activités_file" : "data/input/activities.csv" ,
    "output_dir" : "données/sortie"
  },
  "topsis_settings" : {
    "options_formule_de_proximité" : [ "standard" , "variante" ],
    "formule de proximité" : "variante" ,
    "description_formule_de_proximité" : {
      "standard" : "S*[ i ] = E-[ i ] / (E+[ i ] + E-[ i ]) - Valeurs entre 0 et 1" ,
      "variante" : "S*[ i ] = E-[ i ] / E+[ i ] - Meilleure discrimination, normalisée à [0,1]"
    }
  },
  "paramètres_de_seuil" : {
    "seuil" : 3.0 ,
    "min_seuil" : 0.0 ,
    "seuil_max" : 5.0 ,
    « description » : « Les niveaux de compétence >= seuil sont considérés comme bénéfiques (maximiser), les niveaux < seuil comme non bénéfiques (minimiser) »
  },
  "paramètres_de_poids" : {
    "stratégie" : "uniforme" ,
    "options_stratégie" : [ "uniforme" , "basé_sur_les_exigences" , "personnalisé" ],
    "description_de_la_stratégie" : {
      "uniforme" : "Toutes les compétences ont le même poids" ,
      "requirement_based" : "Poids proportionnels aux niveaux de compétences requis" ,
      « personnalisé » : « Poids définis par l'utilisateur pour chaque compétence »
    }
  },
  "poids_personnalisés" : null
},
  "paramètres_de_sortie" : {
    "top_n_profiles" : 3 ,
    "enregistrer_les_résultats_détaillés" : vrai ,
    "générer_des_visualisations" : vrai ,
    "verbose" : faux
  },
  "paramètres_de_visualisation" : {
    "types_de_graphiques" : [ "barre" , "radar" , "carte_thermique" ],
    "figure_format" : "png" ,
    "chiffre_dpi" : 300
  },
  "affectation_paramètres" : {
    "activer_optimal_assignment" : faux ,
    "méthode_d'affectation" : "auto" ,
    "assignment_method_options" : [ "auto" , "hongrois" , "gourmand" ],
    "descriptions_méthodes_d'affectation" : {
      "hongrois" : "Méthode Hongrois : une méthode de recherche exhaustive qui détermine la meilleure affectation en examinant toutes les combinaisons possibles de profils et de compétences. Elle est efficace pour un petit nombre de profils mais devient rapidement intractable pour un grand nombre de profils ou de compétences.",
      "gourmand" : "Méthode Gourmand : une approche basée sur l'algorithme de recherche gloutonne. Elle commence par assigner le profil avec la plus haute compétence à la première compétence et continue jusqu'à ce que toutes les compétences soient assignées ou que le seuil soit atteint. C'est une méthode rapide mais pas nécessairement optimale."
    }
  }
}
```

```

"auto" : "Sélectionner automatiquement l'approche hongroise (si les dimensions correspondent) ou gourmande" ,
" hongrois " : "Utiliser l'algorithme hongrois pour une affectation optimale 1 à 1 (nécessite un nombre égal de profils et d'activités)" ,
« greedy » : « Utilisez l'approche Greedy pour une affectation optimale (fonctionne avec toutes les dimensions) »
},
" générer _assignment_heatmap " : vrai
}
}

```

La méthode d'affectation : les options sont « **auto** », « **hungarian** » ou « **greedy** »

- **auto** : sélectionne automatiquement **le hongrois** si les dimensions correspondent ; sinon, utilise **Greedy**. Cette option devrait être la méthode d'affectation par défaut en général.
- **hongrois** : Algorithme hongrois des forces (nécessite un nombre égal de profils et d'activités). Cet algorithme est d'ordre de complexité **O(n³)**.
- **Gourmand** : Utilise l'affectation gourmande au mieux (fonctionne avec toutes les dimensions). Cet algorithme rapide utilise l'optimisation locale par itération sur les activités pour les affecter toutes. Il est rapide et d'ordre de complexité **O(n² log n)**.

Le tableau ci-dessous résume les 3 options :

Fonctionnalité	Auto	Cupide algorithme	hongrois algorithme
Optimalité	Ça dépend	Sous-optimal	Globalement optimal
Exigence dimensionnelle	N'importe lequel	N'importe lequel	Doit être égal (n × n)
Complexité	Ça dépend	O(n² log n)	O(n³)
Cas d'utilisation	Défaut	Dimensions inégales	Dimensions égales
Score total	Varie	Peut être inférieur	Maximum possible

Format des données d'entrée

Assurez-vous que les valeurs csv d'entrée sont dans l' intervalle [min_threshold , max_threshold].

Profils CSV (data/input/profiles.csv)

```

Profil, Python , Java, SQL , Communication, Leadership
Dev1 , 5, 3, 4, 4, 2Dev2, 4, 5, 3, 5, 4Dev3, 3, 2, 5, 3, 3

```

- **Première colonne** : Noms/ID de profil
- **Autres colonnes** : Niveaux de compétence (échelle de 0 à 5 ou toute autre échelle)

Activités CSV (data/input/activities.csv)

```

Activité, Python , Java, SQL , Communication, Leadership
Développement back-end, 5, 4, 5, 3, 2 Développement front-end, 3, 5, 2, 4, 2 Chef d'équipe, 3, 3, 3, 5, 5

```

- **Première colonne** : Noms des activités
- **Autres colonnes** : Niveaux de compétences requis (mêmes compétences que les profils)
- **Important** : les colonnes de compétences doivent correspondre entre les profils et les activités

Comment ça marche

1. Transformation des compétences

Pour chaque activité, les niveaux de compétences requis sont analysés :

Exemple :

- Le développement backend nécessite Python=5, Leadership=2
- Python (5 >= 3) : Critère avantageux → Compétences Python supérieures préférées
- Leadership (2 < 3) : Critère non bénéfique → Leadership inférieur acceptable

2. Algorithme TOPSIS

Le système applique TOPSIS en 5 étapes :

1. **Normaliser** la matrice de décision (profils × compétences)
2. **Appliquer des pondérations** aux valeurs normalisées
3. **Déterminer les solutions idéales** (la meilleure et la pire)
4. **Calculer les distances** de chaque profil aux solutions idéales
5. **Calculer les coefficients de proximité** (plus élevé = meilleure correspondance)

3. Classement

Les profils sont classés par coefficient de proximité (0 à 1) : - **1,0** = Correspondance parfaite avec la solution idéale - **0,0** = La plus proche de la pire solution

REMARQUE : si vous utilisez la formule de proximité variante, la correspondance parfaite est toujours 1, ce qui facilite la détection du meilleur choix.

Exemple de sortie

Cas: standard proximity formula

```
=====
Best Profile for Each Activity
=====
Activity           Best Profile          Coefficient
-----
Backend_Development Profile_10          0.750578
Frontend_Development Profile_1           0.672916
Team_Lead            Profile_14          0.659264
Data_Engineer        Profile_1           0.703770
DevOps_Engineer     Profile_4           0.718717
...
=====

=====
RANKING MATRIX - Top 3 Profiles per Activity
=====

Activity      Rank 1          Rank 2          Rank 3
Backend_Development Profile_10 (0.7506) Profile_7 (0.7153) Profile_1 (0.6702)
Frontend_Development Profile_1 (0.6729) Profile_2 (0.6305) Profile_7 (0.6281)
Team_Lead       Profile_14 (0.6593) Profile_9 (0.6236) Profile_12 (0.6116)
Data_Engineer   Profile_1 (0.7038) Profile_10 (0.6652) Profile_7 (0.6460)
DevOps_Engineer Profile_4 (0.7187) Profile_10 (0.6542) Profile_14 (0.6458)
...
```

Cas: variant proximity formula

```
=====
Best Profile for Each Activity
=====
Activity           Best Profile          Coefficient
-----
Backend_Development    Profile_10        1.000000
Frontend_Development   Profile_1         1.000000
Team_Lead              Profile_14        1.000000
Data_Engineer          Profile_1         1.000000
DevOps_Engineer        Profile_4         1.000000
...
=====

=====
RANKING MATRIX - Top 3 Profiles per Activity
=====

```

Activity	Rank 1	Rank 2	Rank 3
Backend_Development	Profile_10 (1.0000)	Profile_7 (0.8348)	Profile_1 (0.6754)
Frontend_Development	Profile_1 (1.0000)	Profile_2 (0.8295)	Profile_7 (0.8207)
Team_Lead	Profile_14 (1.0000)	Profile_9 (0.8564)	Profile_12 (0.8140)
Data_Engineer	Profile_1 (1.0000)	Profile_10 (0.8361)	Profile_7 (0.7681)
DevOps_Engineer	Profile_4 (1.0000)	Profile_10 (0.7404)	Profile_14 (0.7137)

...

4. Options de ligne de commande

Toutes les options sont facultatives. Les paramètres du fichier config.json sont chargés en premier et remplacés si un paramètre est transmis sur la ligne de commande.

```
utilisation : main.py [-h] [-c CONFIG] [--profiles PROFILS] [-- activities ACTIVITÉS]
                     [--threshold SEUIL] [--min-threshold SEUIL_MIN]
                     [--max-threshold SEUIL_MAX] [--activity ACTIVITÉ]
                     [--weight-strategy { uniforme, basé sur les exigences }]
                     [--proximity-formula { standard,variante }]
                     [-v] [--viz] [-o SORTIE]
                     [--assignment] [--assignment-method { auto,hongrois ,gourmand }]
```

Options:

```
-h, --help Afficher le message d'aide
-c, --config Chemin du fichier de configuration
--profiles Chemin du fichier CSV
des profils --activités Chemin du fichier CSV
des activités --threshold Seuil de niveau de compétence ( par défaut : 3,0 )
--min-threshold Niveau de compétence minimum ( par défaut : 0,0 )
--max-threshold Niveau de compétence maximal ( par défaut : 5,0 )
--activité Traiter uniquement une activité spécifique
--weight-strategy Stratégie de génération de poids
--proximity-formula Formule de proximité TOPSIS
-v, --verbose Activer la sortie détaillée
--à savoir, --visualiser Générer des visualisations
-o, --sortir Répertoire de sortie
--assignment Activer l'affectation optimale (hongrois/gourmand)
--assignment-method { auto,hongrois ,gourmand }
Méthode d'affectation (remplace la configuration)
```

Exemples:

```
# Utiliser la configuration par défaut, la plus simple
python main.py
```

```
# Utiliser une configuration personnalisée
python main.py -c mon_config.json
```

```
# Seuil de dépassement
python main.py --seuil 3.5
```

```
# Traiter une activité unique
python main.py --activity " Développement_Backend "
```

```
# Utiliser des fichiers d'entrée personnalisés
python main.py --profiles data/my_profiles.csv --activités data/my_activities.csv
```

```
# Mode verbeux avec visualisations
python main.py -v --viz
```

```
# Utiliser une stratégie de poids différente
python main.py --weight-strategy basé sur les exigences
```

```
# Activer l'affectation optimale avec l'algorithme hongrois/gourmand
python main.py --assignment
```

```
# Forcer l'algorithme hongrois (nécessite des activités/profil CSV de dimensions égales)
python main.py --assignment --assignment-method hongrois
```

```
# Utiliser l'approche d'affectation gloutonne
python main.py --assignment --assignment-method gourmand
```

5. Stratégies des poids

1. Poids uniformes (par défaut)

Toutes les compétences ont la même importance, comme par exemple :

```
poids = [0,1, 0,1, 0,1, ..., 0,1]
```

2. Pondérations basées sur les besoins

Les compétences avec des niveaux requis plus élevés obtiennent des pondérations plus élevées :

poids proportionnels aux niveaux requis

3. Poids personnalisés

Définir des poids spécifiques dans **config.json** :

```
{
  "paramètres_poids" : {
    "stratégie" : "coutume",
    "poids_personnalisés" : [ 0 . 3 , 0 . 2 , 0 . 2 , 0 . 1 , 0 . 1 , 0 . 1 ]
  }
}
```

Dans cet exemple, les poids sont tous normalisés puisque leur somme est 1.

6. Formules TOPSIS

Formule standard (par défaut)

$$S^* = \frac{E^-}{(E^+ + E^-)}$$

- Valeurs entre 0 et 1
- Plus facile à interpréter
- De plus petites différences entre les alternatives

Formule variante

$$S^* = \begin{cases} \frac{E^-}{E^+} & \text{for } E^+ \neq 0 \\ \frac{E^-}{\max_i(E_i^+)} & \text{for } E^+ = 0 \end{cases}$$

Où:

- **E-** = Distance à la pire solution idéale
- **E+** = Distance à la meilleure solution idéale
- **max(E+)** = Distance maximale par rapport à la meilleure parmi toutes les alternatives

Caractéristiques:

- Une meilleure discrimination entre les alternatives
- Gère les cas limites lorsque la distance au meilleur est nulle
- Aucune normalisation requise

- Des valeurs plus élevées indiquent une meilleure correspondance
- Les résultats du classement sont les mêmes que dans le cas de la formule standard

7. Exemples d'utilisation

Exemple 1 : Trouver le meilleur développeur back-end

```
python main.py -- activité " Développement_Backend " -v
```

Cela montrera une analyse détaillée des profils qui correspondent le mieux aux exigences de développement du backend.

Exemple 2 : Ajuster le seuil pour différentes normes

```
# Évaluation stricte (seuil = 4.0)
python main.py --threshold 4.0

# Évaluation indulgente (seuil = 2.5)
python main.py --threshold 2.5
```

Des seuils plus élevés signifient que davantage de compétences sont considérées comme « indispensables » (bénéfiques).

Exemple 3 : Générer un rapport complet avec des visualisations

```
python main.py -v --à savoir --rapports de sortie/analyse_2025
```

Utilisation de l'API (Python)

```
depuis pathlib importer Chemin
importer sys
sys.path.insert ( 0 , ' src ' )

de core.profile_processor importer ProfileProcessor , load_profiles_from_csv , load_activities_from_csv

# Charger les données
profiles_df = load_profiles_from_csv ( 'data/input/profiles.csv' )
activités_df = load_activities_from_csv ( 'data/input/activities.csv' )

# Créer
un processeur processeur = Processeur de profil (
    profils_df = profils_df ,
    activités_df = activités_df ,
    seuil = 3.0 ,
    min_threshold = 0,0 ,
    seuil_max = 5,0 ,
    proximity_formula = 'standard'
)

# Traiter tous
les résultats des activités = processeur.process_all_activities (
    weight_strategy = 'uniform' ,
    verbose = True
)

# Obtenir des résultats d'activité spécifiques
backend_results = processeur.résultats [ ' Backend_Development ' ]
meilleur_profil = backend_results [ ' best_alternative ' ]
print ( f"Meilleur profil pour le développement backend : { best_profile } " )

# Enregistrer les résultats
processor.save_results (Path( 'data/output' ))
```

 **Exemples réels**
Exemple 1 : TOPSIS standard
Activités.csv

Profil	Python	Java	SQL	Communication	Direction	Résolution de problèmes	Travail d'équipe	Gestion de projet	Analyse des données	Cloud Computing
Profil_1	5	3	4		4	2	5	4	2	3
Profil_2	4	5	3		5	4	4	5	3	2
Profil_3	3	2	5		3	3	4	3	2	5
Profil_4	5	4	5		4	4	5	4	4	5
Profil_5	2	3	2		5	5	3	5	5	2
Profil_6	4	4	4		3	3	4	4	3	4
Profil_7	5	5	4		2	2	5	3	1	5
Profil_8	3	4	5		4	3	4	4	3	4
Profil_9	4	3	3		5	5	3	5	4	3
Profil_10	5	5	5		3	2	5	3	2	5
Profil_11	2	2	3		4	4	3	4	4	3
Profil_12	3	4	4		5	4	4	5	4	3
Profil_13	4	3	5		3	3	5	3	3	5
Profil_14	5	4	4		4	5	4	4	5	4
Profil_15	3	5	3		4	3	3	4	3	4

Profils.csv

Activité	Python	Java	SQL	Communication	Direction	Résolution de problèmes	Travail d'équipe	Gestion de projet	Analyse des données	Cloud Computing
Développement backend	5	4	5		3	2	5	3	2	4
Développement Frontend	3	5	2		4	2	4	4	2	2
Chef d'équipe	3	3	3		5	5	4	5	5	2
Ingénieur de données	5	2	5		3	2	5	3	2	5
Ingénieur DevOps	4	3	3		3	2	4	3	3	5
Chef de projet	2	2	2		5	5	4	5	5	2
Développeur FullStack	4	4	4		4	3	4	4	3	4
Administrateur de base de données	3	2	5		3	2	4	3	2	4
Cloud_Architect	4	3	3		3	2	5	3	3	5
Responsable technique	4	4	4		5	5	5	5	5	4

Sortir:

```

Configuration Summary:

Profiles File: data/input/profiles.csv
Activities File: data/input/activities.csv
Output Directory: data/output

Threshold Settings:
Threshold: 3.0
Range: [0.0, 5.0]
Description: Skill levels >= threshold are treated as beneficial (maximize), < threshold as non-beneficial (minimize)

TOPSIS Settings:
Proximity Formula: standard

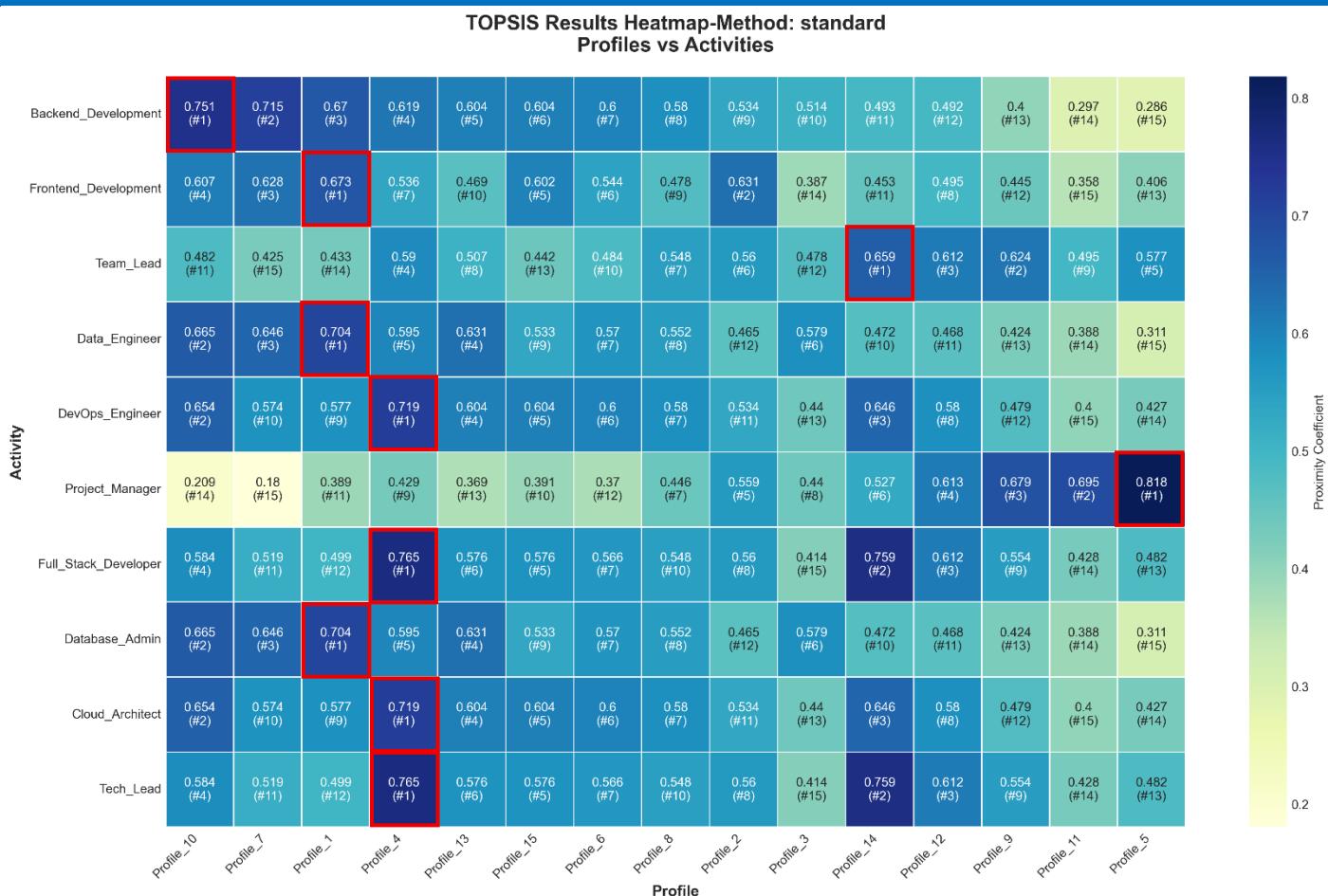
Weight Strategy:
Strategy: uniform
Description: All skills have equal weight

Loading data...
Loaded 15 profiles
Loaded 10 activities
Skills: 10
...

```

Best Profile for Each Activity		
Activity	Best Profile	Coefficient
Backend_Development	Profile_10	0.750578
Frontend_Development	Profile_1	0.672916
Team_Lead	Profile_14	0.659264
Data_Engineer	Profile_1	0.703770
DevOps_Engineer	Profile_4	0.718717
Project_Manager	Profile_5	0.818082
Full_Stack_Developer	Profile_4	0.765258
Database_Admin	Profile_1	0.703770
Cloud_Architect	Profile_4	0.718717
Tech_Lead	Profile_4	0.765258

PROCESSUS TERMINÉ AVEC SUCCÈS

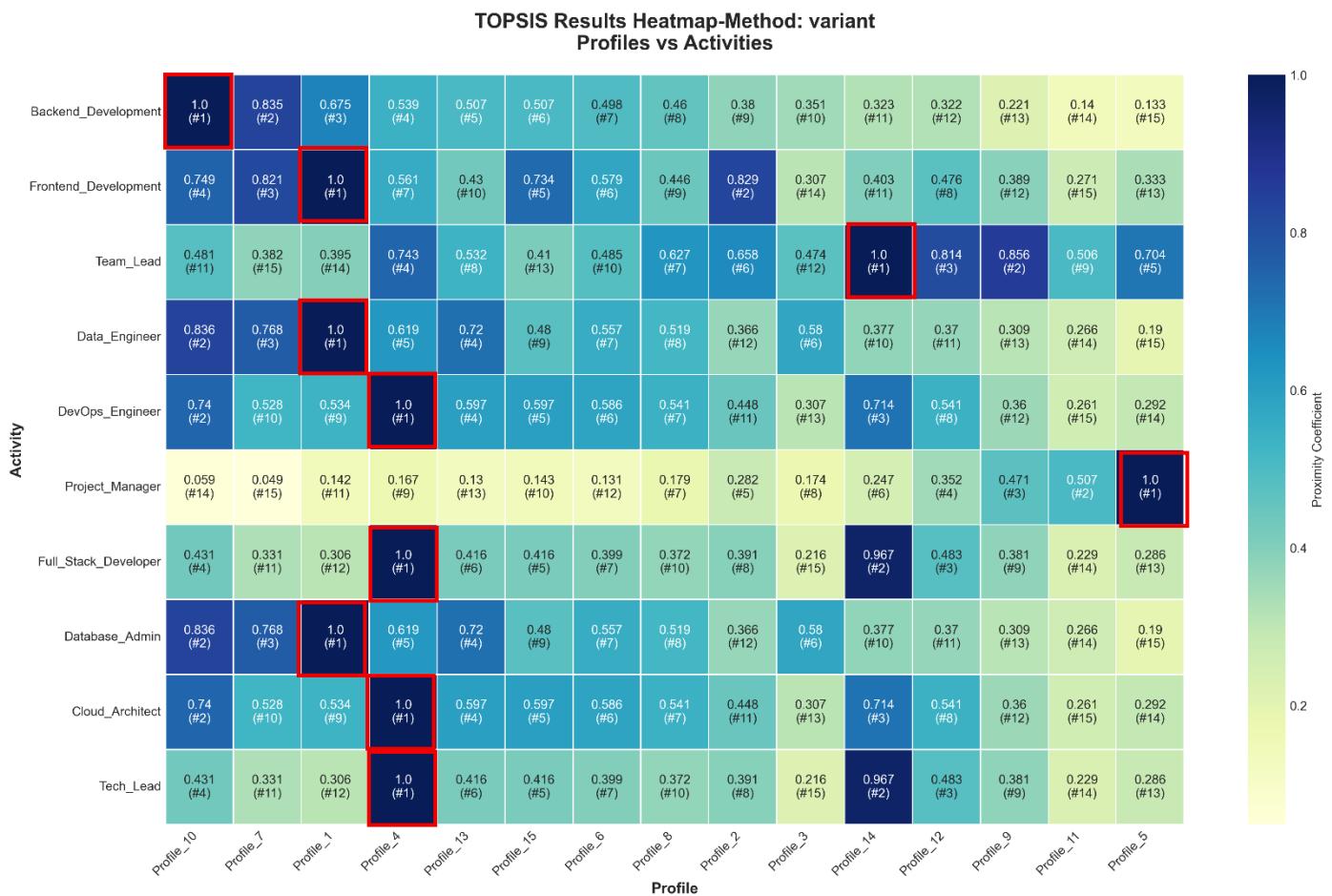


En changeant la formule de proximité = '**standard**' en proximity_formula = '**variant**' on obtient la même matrice d'affectation plus facile à identifier les affectations optimales en repérant les 1 :

Exemple 2 : variante TOPSIS

Identique à l'**exemple 1** en changeant la **formule de proximité** en **variante** :

```
" topsis_settings " : {
  " options_de_formule_de_proximité " : [ "standard", "variante" ] ,
  " formule de proximité " : "variante",
  ...
},
...
}
```



Dans les deux exemples 1 et 2, un profil peut être affecté à plusieurs activités, même si le nombre de profils est supérieur au nombre d'activités. Néanmoins, cela offre aux recruteurs et aux employeurs une plus grande liberté de choix parmi tous les profils, et pas seulement parmi les meilleurs !

Exemple 3 : contrainte d'affectation HONGROISE (affectation 1 à 1) et formule de proximité = variante

Dans ce cas, le nombre d'activités doit être le même que le nombre de profils (=10) :

Activités_2.csv

Profil	Python	Java	SQL	Communication	Direction	Résolution de problèmes	Travail d'équipe	Gestion de projet	Analyse des données	Cloud Computing
Profil_1	5	4	5	3	2	5	3	2	4	5
Profil_2	3	5	2	4	2	4	4	2	2	3
Profil_3	3	3	3	5	5	4	5	5	3	2
Profil_4	5	2	5	3	2	5	3	2	5	4
Profil_5	4	3	3	3	2	4	3	3	3	5
Profil_6	2	2	2	5	5	4	5	5	2	2
Profil_7	4	4	4	4	3	4	4	3	3	4
Profil_8	3	2	5	3	2	4	3	2	4	3
Profil_9	4	3	3	3	2	5	3	3	4	5
Profil_10	4	4	4	5	5	5	5	5	4	4

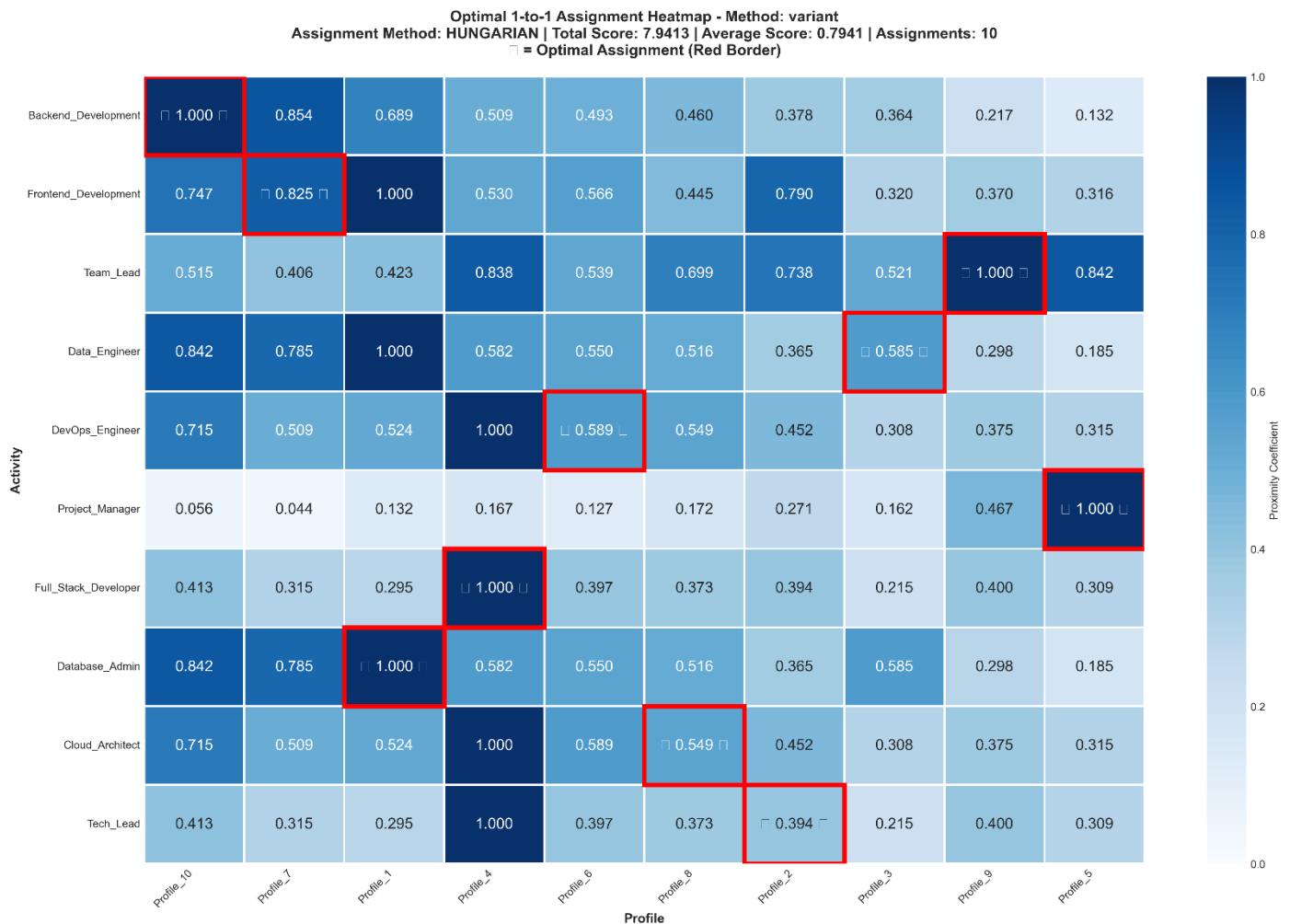
Profils_2.csv

Activité	Python	Java	SQL	Communication	Direction	Résolution de problèmes	Travail d'équipe	Gestion de projet	Analyse des données	Cloud Computing
Développement backend	5	3	4	4	2	5	4	2	3	4
Développement Frontend	4	5	3	5	4	4	5	3	2	3
Chef d'équipe	3	2	5	3	3	4	3	2	5	2
Ingénieur de données	5	4	5	4	4	5	4	4	4	5
Ingénieur DevOps	2	3	2	5	5	3	5	5	2	1
Chef de projet	4	4	4	3	3	4	4	3	4	4
Développeur FullStack	5	5	4	2	2	5	3	1	5	5
Administrateur de base de données	3	4	5	4	3	4	4	3	4	3
Cloud_Architect	4	3	3	5	5	3	5	4	3	2
Responsable technique	5	5	5	3	2	5	3	2	5	5

Ligne de commande :

```
python main.py --assignment --assignment-method hungarian
```

Nous obtenons l'affectation suivante représentée par un graphique de carte thermique :



Nous constatons que même un profil possède 1 comme facteur de proximité, il n'est pas automatiquement affecté à l'activité correspondante comme par exemple : **Profile_7** est affecté à l'activité **Frontend_Development** avec un facteur de proximité de .825<1.0 même si **profile_1** possède 1 comme facteur de proximité pour la même activité.

REMARQUE : La matrice d'affectation HONGROISE n'est pas nécessairement la même dans la **norme Formule de proximité** comme dans le cas de la variante . Application de la formule standard et de la variante **proximity_formulas** donne les conclusions suivantes :

1. Différentes plages de scores :

- **Standard** : [0,173, 0,825] – plage plus compressée
- **Variante** : [0,044, 1,000] – plage plus large avec une meilleure discrimination

2. Optimisation totale différente Notes:

- **Norme** : 6,633 (**moyenne** : 0,663)
- **Variante** : 7,941 (**moyenne** : 0,794)
- **Déférence** : 1,309 points !

Références

Algorithme TOPSIS

- Hwang, CL et Yoon, K. (1981). Prise de décision à attributs multiples : méthodes et applications. Springer-Verlag.
- Yoon, KP et Hwang, CL (1995). Prise de décision à attributs multiples : Introduction. SAGE Publications.

Multicritères Décision Analyse

- Triantaphyllou , E. (2000). Méthodes de prise de décision multicritère : une étude comparative. Kluwer Academic Publishers.

Licence

Ce projet combine des concepts de : - [Système d'attribution de profils \(MCAP\)](#) - Abdel YEZZA (Ph.D), 2025 - [Implémentation de l'algorithme TOPSIS](#) - Abdel YEZZA (Ph.D), 2025

Contribuer

Pour tout problème, suggestion ou contribution, veuillez contacter l'auteur.

Auteur

Abdel YEZZA, Ph.D

Version : 1.0.0 Dernière mise à jour : 2025