

ALGORITHME COMBINATOIRE DES AFFECTATIONS DES PROFILS : MCAP+TOPSIS+HONGROIS

Par : Abdel YEZZA, Ph.D

Date : octobre 2025

Sommaire

🚀 Système de sélection de profils	3
🚀 Aperçu	3
🚀 Schéma fonctionnel global	4
Généralisation de la méthode à d'autre domaines métiers.....	5
Scénario 1 : Classement des profils pour chaque activité	6
Utilisation: Profile Assignment System (MCAP) + TOPSIS.....	6
Scénario 2 : Affectation individuelle optimale.....	7
Utilisation : MCAP + TOPSIS + Méthode hongroise.....	7
FAQ Principales différences Entre les scénarios.....	8
FAQ Comment les algorithmes se complètent.....	8
FAQ Fonctionnalités.....	8
🚀 Installation	9
Prérequis	9
Installer les dépendances	9
Démarrage rapide	9
1. Installation et activation du projet.....	9
2. Utilisation de base (configuration par défaut)	9
3. Seuil personnalisé	9
4. Traitement d'activité unique	9
5. Mode verbeux avec visualisations.....	10
6. Stratégie de poids personnalisée	10
7. Fichiers d'entrée personnalisés	10
Structure du projet	11
Configuration.....	12
Format des données d'entrée	13
Profils CSV (data/input/profiles.csv)	13
Activités CSV (data/input/activities.csv).....	13
🚀 Comment ça marche	14

1. Transformation des compétences	14
2. Algorithme TOPSIS	14
3. Classement	14
Exemple de sortie	14
4. Options de ligne de commande	16
5. Stratégies des poids	17
1. Poids uniformes (par défaut)	17
2. Pondérations basées sur les besoins	17
3. Poids personnalisés	17
6. Formules TOPSIS.....	17
Formule standard (par défaut).....	17
Formule variante	17
7. Exemples d'utilisation.....	18
Exemple 1 : Trouver le meilleur développeur back-end	18
Exemple 2 : Ajuster le seuil pour différentes normes	18
Exemple 3 : Générer un rapport complet avec des visualisations	18
Utilisation de l'API (Python).....	18
🚀 Exemples concrets	19
Exemple 1 : TOPSIS standard vs variant.....	19
Exemple 2 : TOPSIS standard	20
Exemple 3 : variante TOPSIS	23
Exemple 4 : contrainte d'affectation HONGROISE (affectation 1 à 1) et formule de proximité = variante	24
Références.....	26
Algorithme TOPSIS.....	26
Multicritères Décision Analyse	26
Licence	26
Contribuer.....	26
Auteur.....	26

Système de sélection de profils

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

1. La méthode **MCAP** 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 allez le découvrir dans la suite de ce document.

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 :

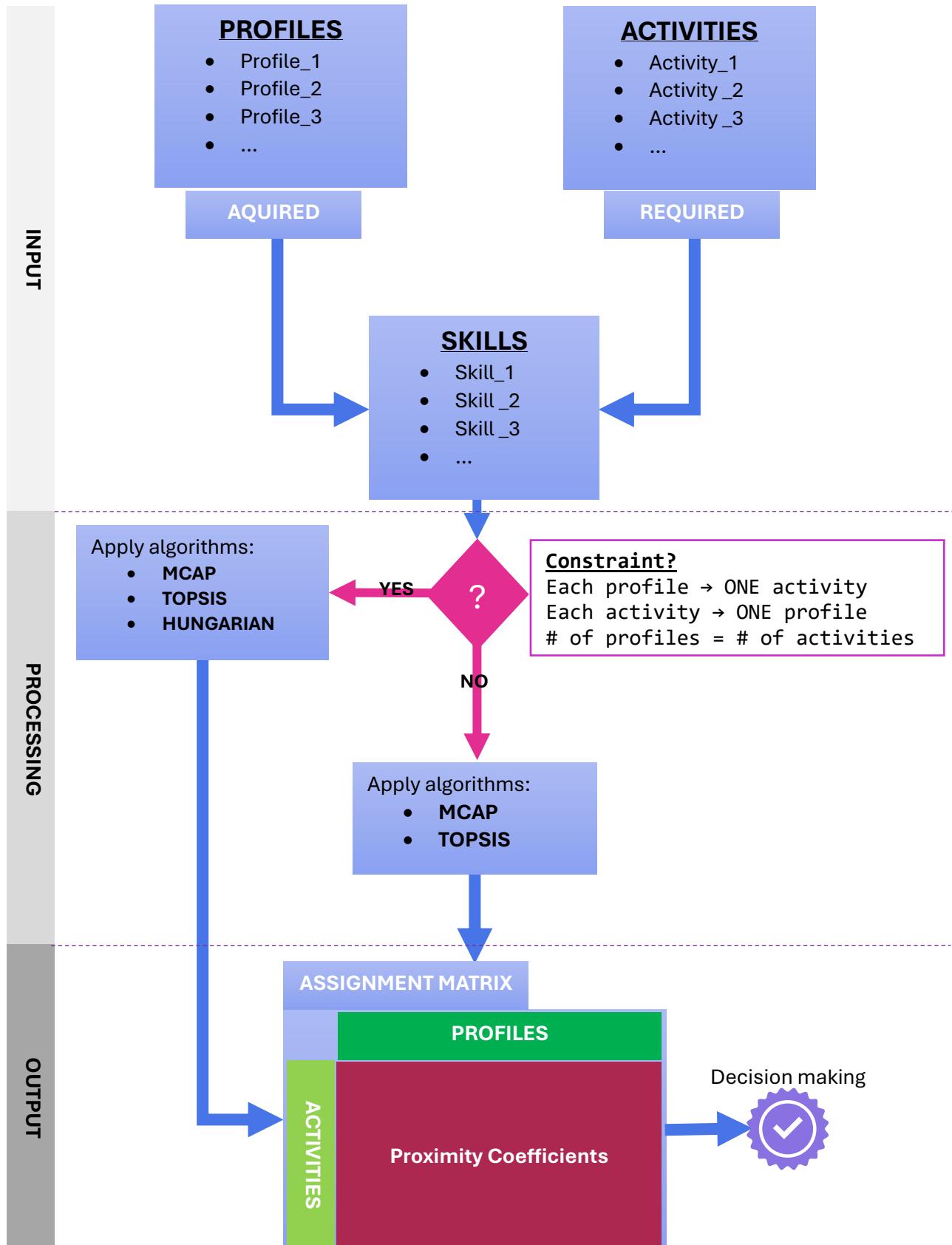
- **Compétences >= Seuil** : Considérées comme des critères **bénéfiques** (plus le seuil est élevé, mieux c'est)
- **Compétences < Seuil** : considérées comme des critères **non bénéfiques** (un seuil inférieur est acceptable)

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).

Cette discrimination des candidats et des activités, est propre au problème de sélection des profils vis-à-vis des activités qu'ils doivent réaliser. Cependant, comme on va l'évoquer ci-dessous (voir [Généralisation de la méthode à d'autre domaines métiers](#)), cet algorithme composé ne s'arrête pas à ce problème, mais peut être généralisé à une multitude de domaines. Par conséquent, la distinction des critères **bénéfiques** de ceux **non bénéfiques**, pourrait avoir une définition complètement différente via une fonction discriminatoire D dépendante du **seuil** fixé comme dans note cas :

$$D(\text{comp}) = \begin{cases} 1 & \text{Si } \text{comp} \geq \text{seuil} \\ 0 & \text{Si } \text{comp} < \text{seuil} \end{cases}$$

Schéma fonctionnel global



Généralisation de la méthode à d'autre domaines métiers

Bien que dans la suite, avec les exemples donnés, l'accent sera mis sur des profils et activité IT, cette méthode s'applique à de multiples domaines métier comme la **RH**, **l'éducation**, la **santé**, **banques et finance**, le **transport**, le **marketing**, le **juridique** etc.

Les principales répercussions positives sur le métier consistent à :

- Adapter les ressources aux besoins
- Allouer les ressources de manière optimale
- Tenir compte de plusieurs critères concurrents
- Trouver les bons compromis entre les profiles et les activités à mener
- Utiliser ce modèle en tant qu'outil d'aide à la décision

A titre d'exemple, on peut l'appliquer dans le **domaine médical** en prenant :

1. La liste des médecins spécialisés et les maladies sur lesquelles sont compétents avec le niveau de spécialisation reconnu (avec une graduation de 0 à 5) :

	Maladie_1	Maladie_2	...
Médecin_1	1	4	...
Médecin_2	5	2	..
...

2. La liste des patients avec les maladies dont ils souffrent avec le niveau d'atteinte (en prenant la même graduation de 0 à 5) :

	Maladie_1	Maladie_2	...
Patient_1	3	0	...
Patient_2	4	2	..
...

Le but recherché, est de trouver la bonne affectation des médecins aux patients :

	Patient_1	Patient_2	...
Médecin_1	Coeffcient_affectation_1_1	Coeffcient_affectation_1_2	...
Médecin_2	Coeffcient_affectation_2_1	Coeffcient_affectation_2_2	..
...

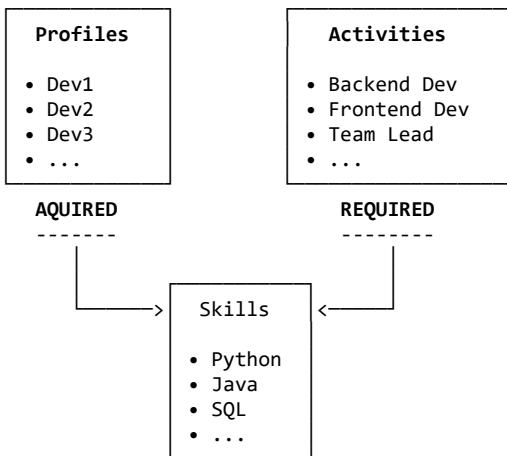
Dans chaque domaine d'application, on peut ajuster les critères communs entre les profiles et les candidats (ici : profiles=médecins, candidats=patients !).

Illustrons le schéma général par quelques scénarios ci-dessous :

Scénario 1 : Classement des profils pour chaque activité

Utilisation: 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

- Normalize profile skills (make comparable)
- Apply weights (importance of each skill)
- Find IDEAL profile (best in all skills)
- Find WORST profile (worst in all skills)
- 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:

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

Frontend Development:

- Dev2 (Score: 0.91) ★ Best match
- 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

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)
Sortie	Liste classée par activité	Devoirs individuels
Flexibilité	Plusieurs candidats par activité	Un profil par activité
Optimisation	Niveau d'activité individuel	Niveau mondial
Algorithmes	MCAP + TOPSIS	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 requirements.txt
```

Démarrage rapide

1. Installation et activation du projet

1. Cloner the dépôt GITHUB:

```
git clone https://github.com/ayezza/topsis_profiles_selection
cd topsis_profiles_selection
```

2. Créer l'environnement virtuel propre au projet:

```
# Windows, Linux, or macOS
python -m venv venv
```

3. Activer l'environnement virtuel:

a. Sur Windows (Command Prompt):

```
venv\Scripts\activate.bat
```

b. SurWindows (PowerShell):

```
venv\Scripts\Activate.ps1
```

c. Sur Linux/macOS:

```
source venv/bin/activate
```

3. Install dependencies:

```
pip install -r requirements.txt
```

2. 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/

3. Seuil personnalisé

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

4. Traitement d'activité unique

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

5. Mode verbeux avec visualisations

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

6. Stratégie de poids personnalisée

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

7. Fichiers d'entrée personnalisés

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

Structure du projet

```
topsis_profiles_selection/
├── main.py                                # Main CLI entry point
├── config.json                            # Configuration file
├── requirements.txt                      # Python dependencies
├── README.md                               # This file
└── src/
    ├── core/
    │   ├── __init__.py
    │   ├── profile_processor.py
    │   ├── topsis_engine.py
    │   ├── optimal_assignment.py
    │   └── ...
    └── visualization/
        └── charts.py

    ├── tests/                                # ✨ NEW: Test scripts
    │   ├── __init__.py
    │   ├── README.md
    │   ├── test_formula_comparison.py
    │   ├── test_hungarian_assignment.py
    │   ├── test_threshold_comparison.py
    │   └── example_heatmap_fontsize.py

    ├── benchmark/                             # ✨ NEW: Benchmark scripts
    │   ├── __init__.py
    │   ├── README.md
    │   ├── benchmark_performance.py
    │   └── run_benchmark_simple.py

    └── data/                                  # Data directory
        ├── input/
        ├── output/
        │   ├── figures/
        │   ├── rankings/
        │   ├── visualizations/
        └── benchmark/

    └── docs/                                 # Documentation
```

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 (Meilleur critère)" ,
      "gourmand" : "Méthode Gourmand (Meilleur critère)" ,
      "auto" : "Méthode Auto (Meilleur critère)"
    }
  }
}
```

```

"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 $O(n^3)$.
- **Greedy**: Utilise l'affectation greedy au mieux (fonctionne avec toutes les dimensions). Cet algorithme rapide utilise l'optimisation locale par itération sur les activités pour affecter la totalité des activités. Il est rapide et d'ordre de complexité $O(n^2 \log n)$.

Le tableau ci-dessous résume les 3 options :

Fonctionnalité	Auto	Algorithme Greedy	Algorithme Hongrois
Optimalité	Tout dépend de l'algorithme	Sous-optimal	✓ Globalement optimal
Dimension	N'importe lequel	n × m avec n pas nécessairement = m	n × n
Complexité	Dépend (Greedy ou Hongrois)	✓ O(n² log n)	O(n³)
Cas d'utilisation	Défaut	Dimensions inégales	Dimensions égales (n)
Score total	Variable	Peut être inférieur	Maximum possible

Il vous appartient de sélectionner l'option qui vous convient (Greedy ou Hongrois) en fonction de l'objectif recherché.

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** = Correspondance parfaite avec la solution idéale
- **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 des meilleurs 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 en ligne de commande, ce qui offre une flexibilité totale.

```
usage: main.py [-h] [-c CONFIG] [--profiles PROFILES] [--activities ACTIVITIES]
                [--threshold THRESHOLD] [--min-threshold MIN_THRESHOLD]
                [--max-threshold MAX_THRESHOLD] [--activity ACTIVITY]
                [--weight-strategy {uniform,requirement_based}]
                [--proximity-formula {standard,variant}]
                [-v] [--viz] [-o OUTPUT]
                [--assignment] [--assignment-method {auto,hungarian,greedy}]
```

Options:

-h, --help	Show help message
-c, --config	Configuration file path
--profiles	Profiles CSV file path
--activities	Activities CSV file path
--threshold	Skill level threshold (default: 3.0)
--min-threshold	Minimum skill level (default: 0.0)
--max-threshold	Maximum skill level (default: 5.0)
--activity	Process only specific activity
--weight-strategy	Weight generation strategy
--proximity-formula	TOPSIS proximity formula
-v, --verbose	Enable verbose output
--viz, --visualize	Generate visualizations
-o, --output	Output directory
--assignment	Enable optimal assignment (Hungarian/Greedy)
--assignment-method	{auto,hungarian,greedy}
	Assignment method (overrides config)

Examples:

```
# Use default configuration, the most simple
python main.py

# Use custom configuration
python main.py -c my_config.json

# Override threshold
python main.py --threshold 3.5

# Process single activity
python main.py --activity "Backend_Development"

# Use custom input files
python main.py --profiles data/my_profiles.csv --activities data/my_activities.csv

# Verbose mode with visualizations
python main.py -v --viz

# Use different weight strategy
python main.py --weight-strategy requirement_based

# Enable optimal assignment with Hungarian/Greedy algorithm
python main.py --assignment

# Force Hungarian algorithm (requires activities/profiles CSV equal dimensions)
python main.py --assignment --assignment-method hungarian

# Use greedy assignment approach
python main.py --assignment --assignment-method greedy
```

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 égale à 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)

```
from pathlib import Path
import sys
sys.path.insert(0, 'src')

from core.profile_processor import ProfileProcessor, load_profiles_from_csv, load_activities_from_csv

# Load data
profiles_df = load_profiles_from_csv('data/input/profiles.csv')
activities_df = load_activities_from_csv('data/input/activities.csv')

# Create processor
processor = ProfileProcessor(
    profiles_df=profiles_df,
    activities_df=activities_df,
    threshold=3.0,
    min_threshold=0.0,
    max_threshold=5.0,
    proximity_formula='standard'
)

# Process all activities
results = processor.process_all_activities(
    weight_strategy='uniform',
    verbose=True
)

# Get specific activity results
backend_results = processor.results['Backend_Development']
best_profile = backend_results['best_alternative']
print(f"Best profile for Backend Development: {best_profile}")

# Save results
processor.save_results(Path('data/output'))
```

Exemples concrets

Exemple 1 : TOPSIS standard vs variant

Activities_sample.csv

Activity,Skill_1,Skill_2,Skill_3,Skill_4,Skill_5,Skill_6,Skill_7,Skill_8,Skill_9,Skill_10
 Activity_1,5,4,5,3,2,5,3,2,4,5
 Activity_2,3,5,2,4,2,4,4,2,2,3
 Activity_3,3,3,3,5,5,4,5,5,3,2
 Activity_4,5,2,5,3,2,5,3,2,5,4
 Activity_5,4,3,3,3,2,4,3,3,3,5

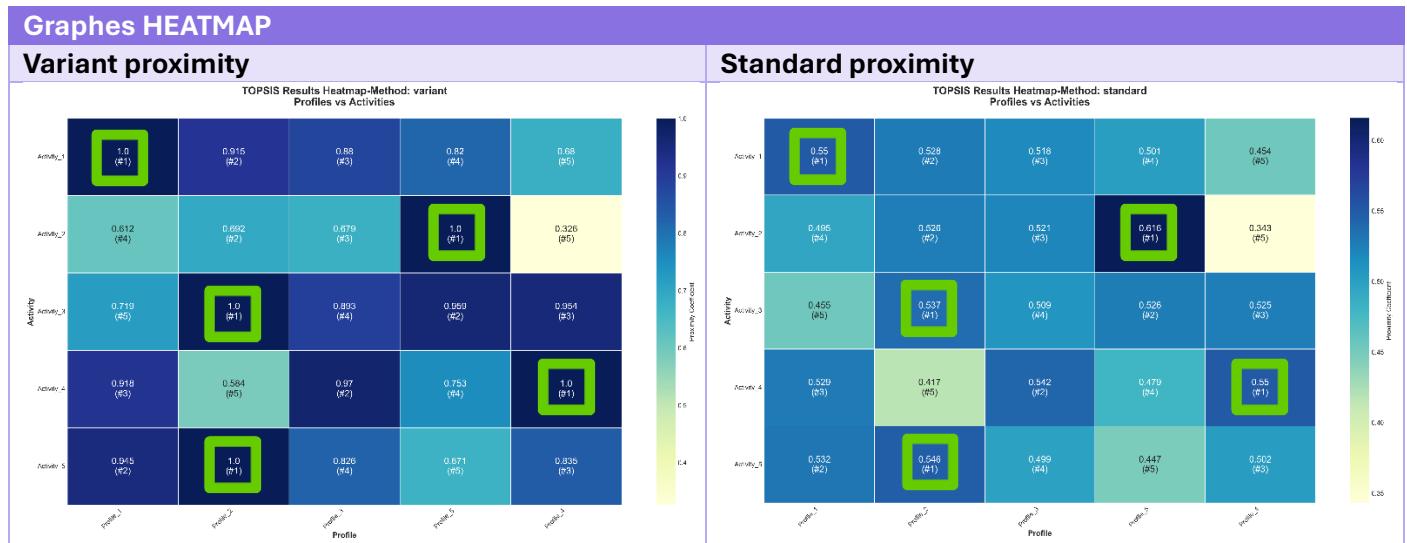
Profiles_sample.csv

Profile,Skill_1,Skill_2,Skill_3,Skill_4,Skill_5,Skill_6,Skill_7,Skill_8,Skill_9,Skill_10
 Profile_1,0,3,4,1,2,5,4,2,3,4
 Profile_2,1,5,3,5,4,4,0,3,2,3
 Profile_3,3,2,2,3,3,4,0,2,4,2
 Profile_4,1,0,5,0,4,5,4,4,4,2
 Profile_5,2,3,0,5,5,3,2,1,2,1

Ranking	
Variant proximity	Standard proximity
Activity,Rank 1,Rank 2,Rank 3	Activity,Rank 1,Rank 2,Rank 3
Activity_1, Profile_1 (1), Profile_2 (0.9152), Profile_3 (0.8803)	Activity_1, Profile_1 (0.5500), Profile_2 (0.5280), Profile_3 (0.5183)
Activity_2, Profile_5 (1), Profile_2 (0.6923), Profile_3 (0.6786)	Activity_2, Profile_5 (0.6156), Profile_2 (0.5258), Profile_3 (0.5208)
Activity_3, Profile_2 (1), Profile_5 (0.9587), Profile_4 (0.9536)	Activity_3, Profile_2 (0.5369), Profile_5 (0.5264), Profile_4 (0.5251)
Activity_4, Profile_4 (1), Profile_3 (0.9698), Profile_1 (0.9185)	Activity_4, Profile_4 (0.5498), Profile_3 (0.5423), Profile_1 (0.5287)
Activity_5, Profile_2 (1), Profile_1 (0.9448), Profile_4 (0.8354)	Activity_5, Profile_2 (0.5464), Profile_1 (0.5323), Profile_4 (0.5016)

Les deux méthodes fournissent le même classement, mais la méthode variante est plus lisible avec des écarts plus larges pour les rangs au-delà du 1^{er}.

Cela se voit plus clairement sur les graphes de type heatmap :



Exemple 2 : TOPSIS standard

Activités.csv

Profile	Python	Java	SQL	Communication	Leadership	Problem_Solving	Teamwork	Project_Management	Data_Analysis	Cloud_Computing
Profile_1	5	3	4		4	2		5	4	
Profile_2	4	5	3		5	4		4	5	
Profile_3	3	2	5		3	3		4	3	
Profile_4	5	4	5		4	4		5	4	
Profile_5	2	3	2		5	5		3	5	
Profile_6	4	4	4		3	3		4	4	
Profile_7	5	5	4		2	2		5	3	
Profile_8	3	4	5		4	3		4	3	
Profile_9	4	3	3		5	5		3	4	
Profile_10	5	5	5		3	2		5	3	
Profile_11	2	2	3		4	4		3	4	
Profile_12	3	4	4		5	4		4	3	
Profile_13	4	3	5		3	3		5	3	
Profile_14	5	4	4		4	5		4	4	
Profile_15	3	5	3		4	3		3	4	

Profils.csv

Activity	Python	Java	SQL	Communication	Leadership	Problem_Solving	Teamwork	Project_Management	Data_Analysis	Cloud_Computing
Backend_Development	5	4	5		3	2		5	3	
Frontend_Development	3	5	2		4	2		4	4	
Team_Lead	3	3	3		5	5		4	5	
Data_Engineer	5	2	5		3	2		5	3	
DevOps_Engineer	4	3	3		3	2		4	3	
Project_Manager	2	2	2		5	5		4	5	
Full_Stack_Developer	4	4	4		4	3		4	3	
Database_Admin	3	2	5		3	2		4	3	
Cloud_Architect	4	3	3		3	2		5	4	
Tech_Lead	4	4	4		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 \geq 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

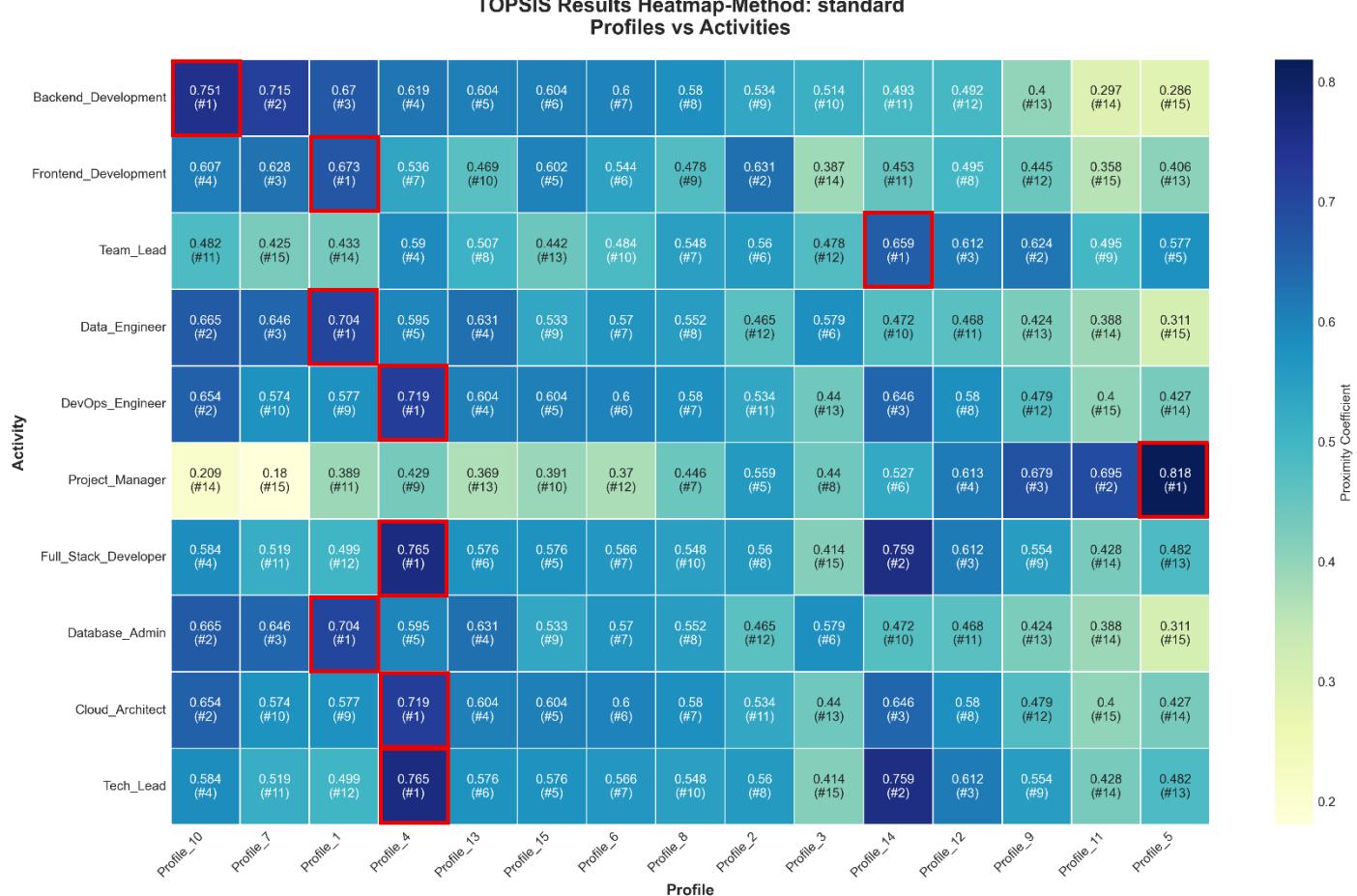
Best Profile for Each Activity

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

=====

PROCESS COMPLETED SUCCESSFULLY

=====

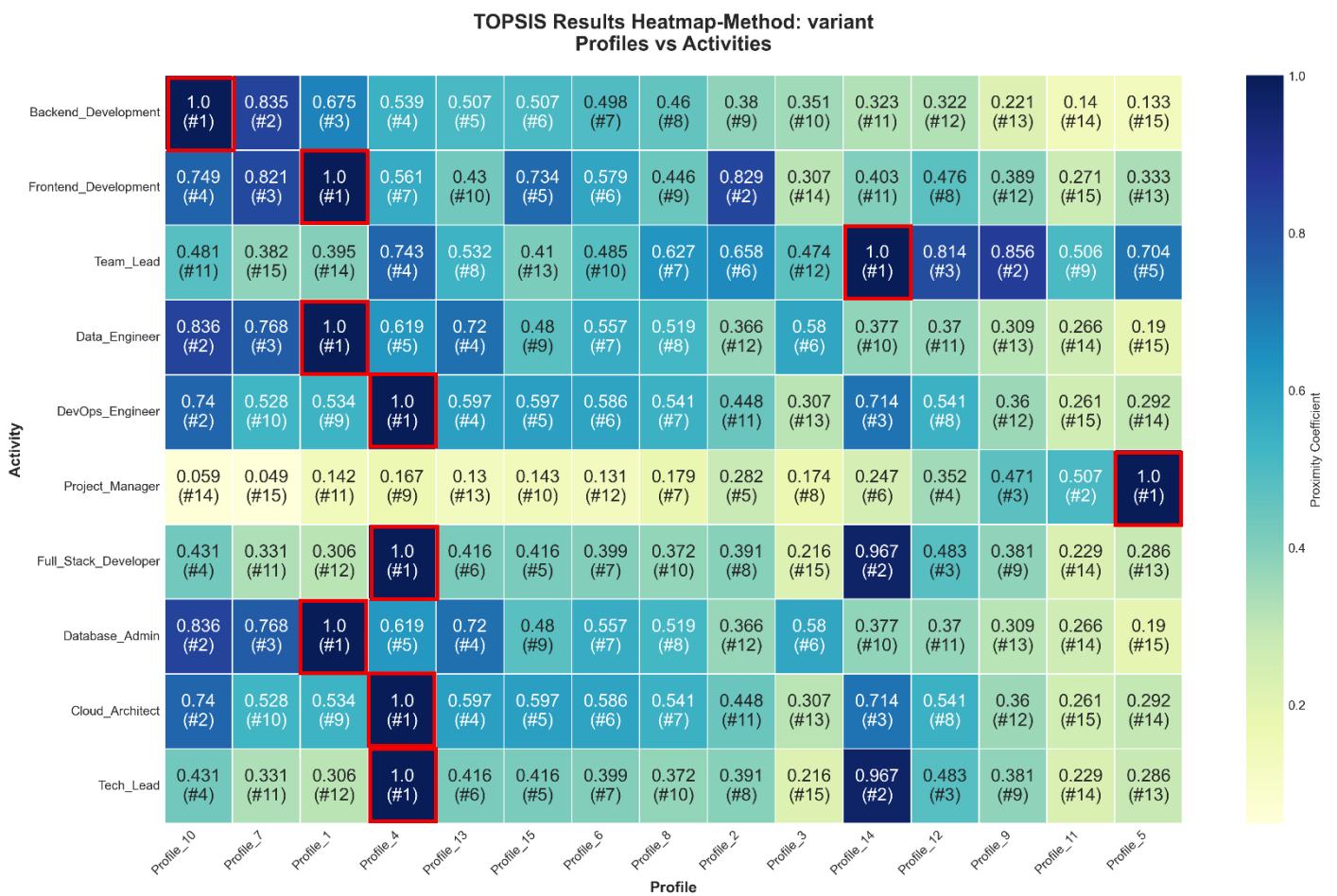


En changeant la formule de proximité = `'standard'` en `proximity_formula = 'variant'` on obtient la même matrice d'affectation dans laquelle il est plus facile à identifier les affectations optimales en repérant tout simplement les 1 :

Exemple 3 : 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 ! Tout le monde a le droit de travailler (mais aussi l'obligation selon les situations individuelles).

Exemple 4 : 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) :

Activities_2.csv

Profile	Python	Java	SQL	Communication	Leadership	Problem_Solving	Teamwork	Project_Management	Data_Analysis	Cloud_Computing
Profile_1	5	4	5	3	2	5	3	2	4	5
Profile_2	3	5	2	4	2	4	4	2	2	3
Profile_3	3	3	3	5	5	4	5	5	3	2
Profile_4	5	2	5	3	2	5	3	2	5	4
Profile_5	4	3	3	3	2	4	3	3	3	5
Profile_6	2	2	2	5	5	4	5	5	2	2
Profile_7	4	4	4	4	3	4	4	3	3	4
Profile_8	3	2	5	3	2	4	3	2	4	3
Profile_9	4	3	3	3	2	5	3	3	4	5
Profile_10	4	4	4	5	5	5	5	5	4	4

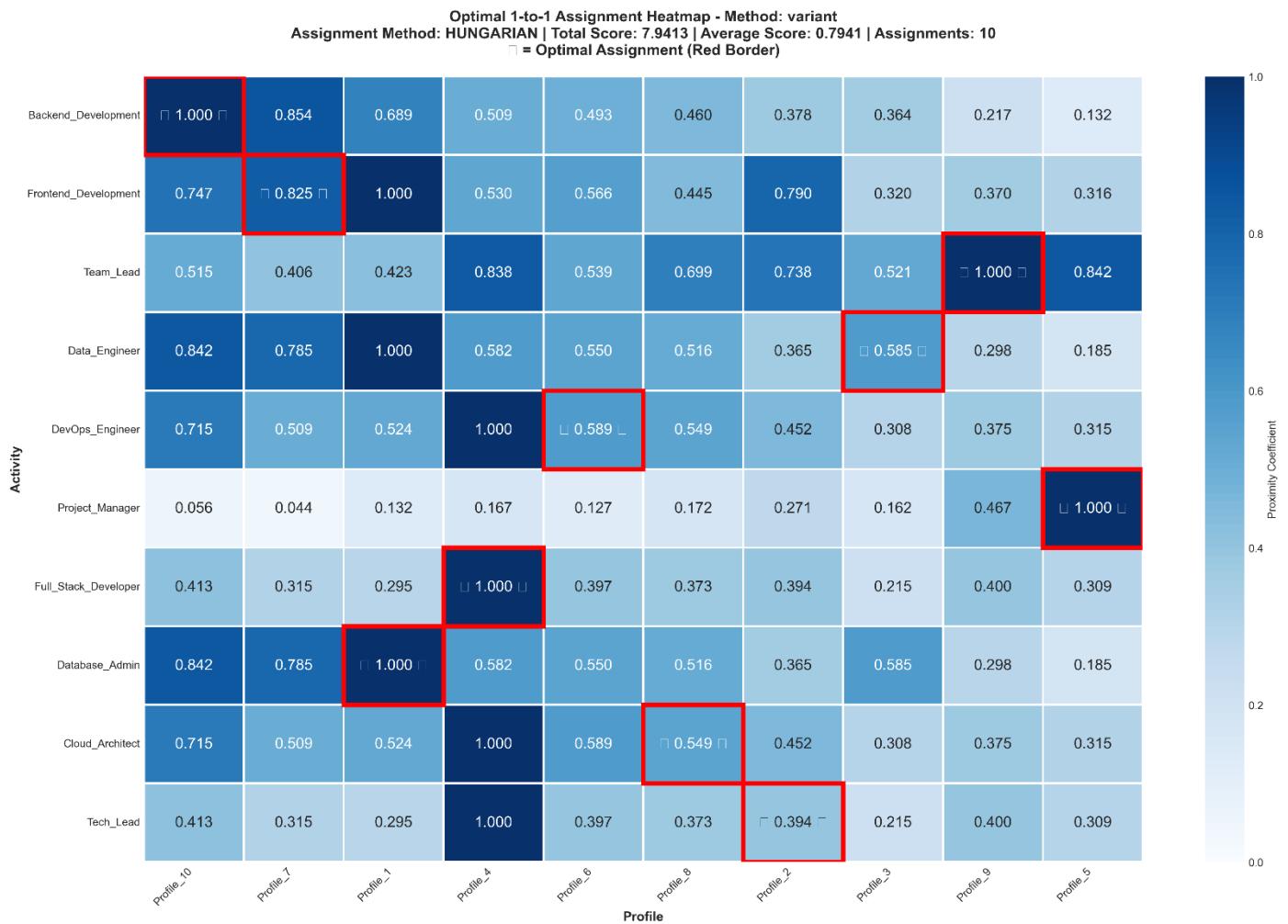
Profiles_2.csv

Activity	Python	Java	SQL	Communication	Leadership	Problem_Solving	Teamwork	Project_Management	Data_Analysis	Cloud_Computing
Backend_Development	5	3	4	4	2	5	4	2	3	4
Frontend_Development	4	5	3	5	4	4	5	3	2	3
Team_Lead	3	2	5	3	3	4	3	2	5	2
Data_Engineer	5	4	5	4	4	5	4	4	4	5
DevOps_Engineer	2	3	2	5	5	3	5	5	2	1
Project_Manager	4	4	4	3	3	4	4	3	4	4
Full_Stack_Developer	5	5	4	2	2	5	3	1	5	5
Database_Admin	3	4	5	4	3	4	4	3	4	3
Cloud_Architect	4	3	3	5	5	3	5	4	3	2
Tech_Lead	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 graphe heatmap :



Nous constatons que même si 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 le cas **standard de la formule de proximité** comme dans le cas **de la variante**. L'application de **la formule standard** et **de la variante de 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 des scores:

- **Standard** : 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 des deux articles :

- [Profile Assignment System \(MCAP\)](#) : Abdel YEZZA (Ph.D), 2025
- [TOPSIS Algorithm Implementation](#) : 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