Application of Graph Theory for Similarity-Based Player Recommendation in Career Mode EA Sports FC

Reinsen Silitonga - 13524093
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: reinsilitonga12@gmail.com , 13524093@std.stei.itb.ac.id

***Abstract*— This paper presents a graph-theoretic approach for developing a similarity-based player recommendation system in EA Sports FC Career Mode. The system addresses the challenge faced by managers of smaller clubs who need to identify effective, budget-friendly players that match their tactical requirements. By representing players as vertices and their attribute similarities as weighted edges, a complete weighted graph is constructed using Euclidean distance calculations. The implementation incorporates K-Means clustering to group players with similar characteristics, enhancing recommendation efficiency and relevance. A bipartite graph structure is utilized to analyze player positional suitability across various tactical roles. The system is implemented using Python with libraries including pandas, scikit-learn, and scipy, demonstrating effective identification of "hidden gems" - players with similar tactical profiles to expensive alternatives but at more affordable prices. Results show that the graph-based approach successfully models complex player relationships and provides actionable recommendations for strategic transfer decisions in Career Mode.**

Keywords— graph theory; player recommendation system; EA Sports FC; similarity analysis; K-means clustering; weighted graph; adjacency matrix; bipartite graph; Euclidean distance; career mode; football simulation; transfer system;

EA Sports FC is a leading football simulation game offering a realistic and intricate experience, particularly in its Career Mode. Here, players assume the role of a club manager, overseeing finances, team strategy, and crucial player recruitment. The game features an extensive database of players, each with numerous attributes (e.g., Pace, Shooting, Passing), unique Play Styles (special abilities often optimized from real-world data), and specific Player Roles (tactical responsibilities within a formation). This depth, combined with an interactive transfer system , makes manually identifying and comparing players a complex and time-consuming task, especially with the expanded player database that now includes women's squads[1].

Fig. 1. Cover of EA Sports FC 25. *(source: https://www.playstation.com/en-id/games/ea-sports-fc/)*

A significant portion of EA Sports FC's users enjoy the challenge of leading smaller, less successful teams to glory. This "road to glory" often means operating with limited transfer budgets, making it difficult to acquire top-tier players with high overall ratings or established reputations. Consequently, managers frequently struggle to find effective players who fit their tactical vision without breaking the bank.

This scenario highlights the critical need for an intelligent player recommendation system. Such a system is designed to help managers discover "hidden gems"—players who possess similar key characteristics and tactical suitability to expensive, highly-rated players, but are available at a more affordable price point. By analyzing detailed player attributes, Play Styles, and Player Roles, the system can identify individuals whose combined abilities make them highly effective for specific tactical roles, even if their overall rating isn't elite. This empowers managers of smaller clubs to make shrewd, budget-conscious transfers that align with their long-term vision.

To achieve this, I propose a graph-theoretical approach. In this system, players, their attributes, Play Styles, and tactical roles are represented as nodes (vertices). The relationships and interactions between them become edges (links), weighted to reflect their strength or similarity. Graph theory provides a powerful framework to model these complex interdependencies,

analyze patterns, and ultimately recommend players who best fit a manager's tactical needs and budget constraints, fostering stronger, more competitive squads. This paper details how graph theory principles can be applied to build such a sophisticated player recommendation system for EA Sports FC Career Mode, enabling more informed and strategic transfer decisions.

I. Theoretical Foundation

A. Graph

A graph G is formally defined as a pair $G=(V,E)$, where V is a non-empty set of vertices (also referred to as nodes or points), and E is a set of edges (also known as links or lines) that connect pairs of vertices. The set $V=\{v1,v2,...,vn\}$ must not be empty, meaning that a graph must contain at least one vertex. In contrast, the set $E=\{e1,e2,...,em\}$ can be empty, indicating that a graph can exist without any connections between its vertices. Based on the presence of loops or multiple edges, graphs are generally classified into two main types:

1. Simple Graph: A graph that contains neither loops (edges connecting a vertex to itself) nor multiple edges (two or more edges connecting the same pair of vertices) is called a simple graph.
2. Non-Simple Graph (Unsimple Graph): A graph that contains multiple edges or loops is termed a non-simple graph. Non-simple graphs can be further categorized:
    - Multigraph: A graph that contains multiple edges between the same pair of vertices.
    - Pseudograph: A graph that contains loops (edges connecting a vertex to itself).

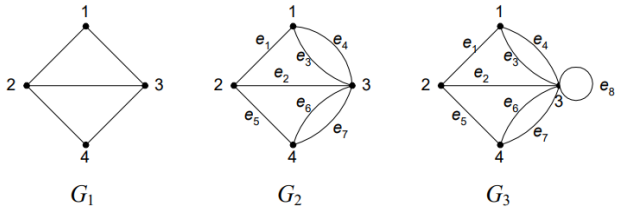

Fig. 2. Simple Graph (G1), Multigraph (G2), Pseudograph (G3). *(source: RinaldiMunir/Matdis)*

Based on the orientation of edges, graphs are distinguished as:

1. Undirected Graph: A graph where the edges do not have an orientation or direction. The relationship between two connected vertices is symmetrical.
2. Directed Graph (Digraph): A graph where each edge has a specified orientation or direction. The relationship between two connected vertices is asymmetrical. A directed graph can also be a Directed Multigraph if it allows multiple directed edges between the same ordered pair of vertices.

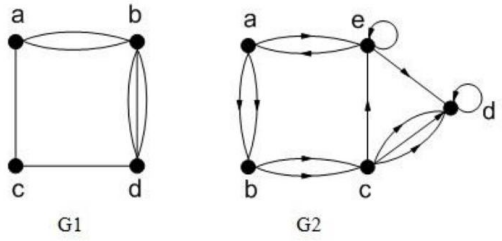

Fig. 3. Undirected Graph (G1), Directed Graph (G2). *(source: RinaldiMunir/Matdis)*

Several Special Graphs

1. Complete Graph: A complete graph is a simple graph in which every vertex has an edge to all other vertices. A complete graph with n vertices is denoted by Kn. The number of edges in a complete graph with n vertices is n(n – 1)/2.
2. Cycle Graph: A cycle graph is a simple graph in which every vertex has a degree of two. A cycle graph with n vertices is denoted by Cn.
3. Regular Graph: A graph in which every vertex has the same degree is called a regular graph. If each vertex has a degree r, the graph is referred to as an r-regular graph. The number of edges in a regular graph is nr/2.
4. Bipartite Graph: A graph G whose set of vertices can be divided into two subsets V1 and V2, such that each edge in G connects a vertex in V1 to a vertex in V2, is called a bipartite graph and is denoted as G(V1, V2).
5. Weighted Graph: A weighted graph is a graph in which each edge is assigned a value (weight)[2].

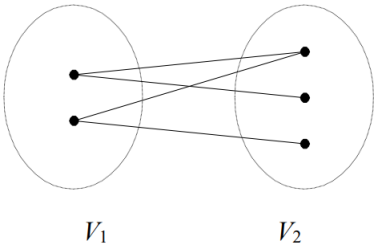

Fig. 4. Bipartite Graph. *(source: RinaldiMunir/Matdis)*

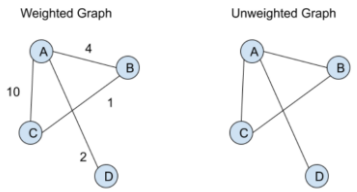

Fig. 5. Weighted and Unweighted Graph. *(source: RinaldiMunir/Matdis)*

In the context of this player recommendation system, a graph is a mathematical structure comprising a non-empty set of vertices (or nodes) and a set of edges (or links) that connect pairs of these vertices. Vertices in our model represent various entities such as players, their attributes, play styles, position, skills etc. Edges define the relationships between these entities; for instance, an edge might link a player to an attribute they possess

or connect two players based on their attribute similarity. These edges can be undirected, signifying a symmetrical relationship (e.g., mutual similarity between two players), or directed, indicating an asymmetrical relationship (e.g., a player's assigned primary position). Crucially, edges can be weighted, meaning they carry a numerical value that quantifies the strength, cost, or importance of the relationship. In our system, this weight can represent the degree of attribute similarity (where a lower weight implies higher similarity) or a player's "Role Familiarity" with a specific position. Other important terms include adjacent vertices (directly connected by an edge), the degree of a vertex (the number of edges connected to it, indicating its connectivity), and a path (a sequence of connected vertices and edges, essential for understanding indirect relationships or "distances" between players in the graph).

B. Graph Representation of Player Data

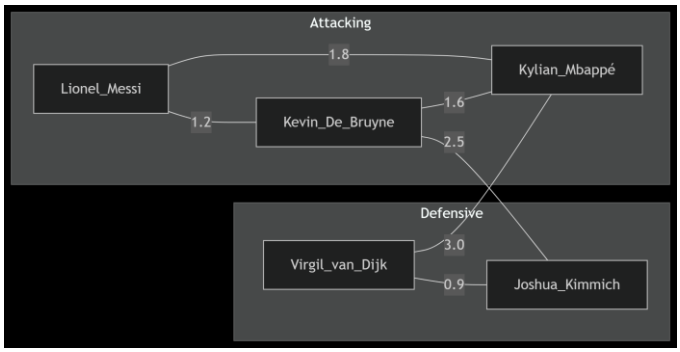

Fig. 6. Example Graph Representation of Player Data

The foundation of this recommendation system involves transforming the extensive EA Sports FC 25 player data into a graph structure. Specifically, each unique player within the game's database is modeled as a distinct vertex (V) in the graph. Accompanying each player vertex are their comprehensive numerical attributes, which form the core data points for similarity assessment. An edge connects two player vertices (u and v) and quantitatively represents the similarity between their respective attribute sets. A critical aspect of these edges is their weight *(w(u, v))*, which serves to quantify this similarity. To calculate these weights, standard statistical metrics such as Euclidean distance or cosine similarity are employed. For example, if Euclidean distance is selected, the weight between two players u and v, each described by n attributes, is computed using the formula:

$$w(u,v) = \sqrt{\sum_{i=1}^{n}(ui - vi)^2}, \quad (2.1)$$

where ui and vi denote the values of the i-th attribute for players u and v, respectively. A crucial interpretation is that a *lower* edge weight signifies a *higher* degree of attribute similarity, meaning players connected by smaller weights are more alike in their profiles. This entire process ideally culminates in the creation of a weighted complete graph, where theoretically, every player vertex is connected to every other player vertex. The weight of each edge in this complete graph directly corresponds to the calculated attribute distance, ensuring that all potential similarity relationships across the player database are comprehensively modeled.

C. K-Means Clustering

K-Means is a prototype-based, simple partitional clustering algorithm that attempts to find K non-overlapping clusters. These clusters are represented by their centroids (a cluster centroid is typically the mean of the points in that cluster). The clustering process of K-means is as follows. First, K initial centroids are selected, where K is specified by the user and indicates the desired number of clusters. Every point in the data is then assigned to the closest centroid, and each collection of points assigned to a centroid forms a cluster. The centroid of each cluster is then updated based on the points assigned to that cluster. This process is repeated until no point changes clusters.

It is beneficial to delve into the mathematics behind K-means. Suppose D = {x1,..., xn} is the data set to be clustered. K-means can be expressed by an objective function that depends on the proximities of the data points to the cluster centroids as follows:

$$\sum_{k=1}^{K}\sum_{x\in Ck}\pi x dist(x, mk), \quad (2.2)$$

where x is the weight of x, nk is the number of data objects assigned to cluster Ck, K is the number of clusters set by the user, and the function "dist" computes the distance between object x and centroid mk, 1 <= k <= K. While the selection of the distance function is optional, the squared Euclidean distance, i.e. $\| x - m \|^2$, has been most widely used in both research and practice. The iteration process introduced in the previous paragraph is indeed a gradient-descent alternating optimization method that helps to solve Eq. (2.2), although often converges to a local minimum or a saddle point[3].

In the context of our system, K-Means is applied to the player data, which is represented by the distances derived from the weighted adjacency matrix. The algorithm operates by iteratively assigning each player vertex to one of K pre-defined clusters, where K is the number of desired clusters. This assignment is based on minimizing the distance between the player's attribute vector and the centroid (mean attribute vector) of the cluster. After all players are assigned, the centroids of the clusters are recalculated based on the new assignments. This iterative process continues until the cluster assignments no longer change significantly, or a specified maximum number of iterations is reached. The primary output of this algorithm is a set of distinct, homogeneous clusters, effectively grouping players with highly similar statistical profiles. For example, players might be grouped into categories like 'Fast Wingers,' 'Creative Midfielders,' or 'Ball-Playing Defenders,' based on their intrinsic characteristics. By first categorizing players into these clusters, the recommendation system can efficiently narrow down the search space for similar players, thereby enhancing the relevance and speed of the subsequent recommendation process.

D. Bipartite Graph for Position Analysis

The concept of a bipartite graph offers a specialized and powerful tool within our system for analyzing player suitability across various in-game positions. A bipartite graph is a type of graph whose vertices can be divided into two disjoint and independent sets, U and V, such that every edge connects a vertex in U to one in V, and no edges exist within U or within

V. In our specific application, one set of vertices (V1) would exclusively represent the players, while the second, distinct set (V2) would represent the various playable positions available in EA Sports FC 25 (e.g., Striker (ST), Central Attacking Midfielder (CAM), Central Midfielder (CM), Center Back (CB), Right Back (RB), Left Wing (LW), Right Wing (RW)). An edge connecting a player from V1 to a position from V2 would signify that the player can effectively perform in that particular role. This structural representation allows for a clear, intuitive visualization and systematic analysis of player positional versatility. It goes beyond simple attribute matching by explicitly modeling which roles a player is capable of fulfilling, which is critical for tactical decision-making in Career Mode. For instance, this approach can help managers identify players who are proficient in multiple roles, thereby adding valuable versatility and strategic depth to their squad, especially when operating with limited budgets

Consider the following table illustrating a small example of player positional suitability:

TABLE I. EXAMPLE PLAYER POSITIONAL SUITABILITY

Player Name	Primary Position	Other Suitable Positions
Kylian Mbappé	ST	LW, RW
Kevin De Bruyne	CAM	CM
Virgil van Dijk	CB	
Trent Alexander-Arnold	RB	CM
Jude Bellingham	CM	CAM, CDM

E. Adjacency Matrix

Another method is to represent a graph using an adjacency matrix, typically denoted as A. To create this matrix, start by assigning a number to each vertex, so the set of vertices becomes $V=\{v1,v2,\ldots,vn\}$ for a graph with n vertices. The adjacency matrix **A** is then an $n \times n$ matrix, where each entry is determined based on the following rule:

$$Aij = \begin{cases} 1, if\ (vi, vj) \in E \\ 0, otherwise \end{cases} \quad (2.3)$$

The Adjacency Matrix (A**)** is the fundamental data structure chosen for efficiently storing and accessing the intricate similarity relationships between players in our weighted graph. For a graph encompassing N players, an $N \times N$ matrix A is constructed. Each element A_{ij} within this matrix directly holds the weight of the edge connecting player i and player j. This weight, as established in Section B, quantifies their attribute similarity (using Euclidean distance). Conventionally, if a player is being compared to themselves, A_{ii} would be 0. If no direct relationship or a very high dissimilarity exists (though less common in a complete similarity graph), the corresponding A_{ij} might be represented as infinity or a very large number. This matrix serves as an instantaneous lookup table, allowing for rapid retrieval of the similarity value between any given pair of players. Its structure makes it particularly efficient for analyzing connections across a potentially dense network of player profiles, as it provides direct access to every pairwise relationship, which is crucial for algorithms like shortest path and clustering[4].

Here's an example of an Adjacency Matrix representing the similarity relationships between a small set of hypothetical players. Let's consider three players: Player 1, Player 2, and Player 3. The values in this matrix represent the Euclidean distance between their normalized attributes, where a lower value signifies higher similarity. Using hypothetical Euclidean distance values for these three players:

- Player 1 to Player 2: 1.5
- Player 1 to Player 3: 2.8
- Player 2 to Player 3: 0.9 (meaning Player 2 and Player 3 are quite similar)

The 3×3 Adjacency Matrix would be:

$$\begin{matrix} 0.0 & 1.5 & 2.8 \\ 1.5 & 0.0 & 0.9 \\ 2.8 & 0.9 & 0.0 \end{matrix}$$

This 3x3 matrix represents three players, with 3 rows and 3 columns corresponding to each player. The main diagonal elements (A_{11}, A_{22}, A_{33}) are all 0.0, indicating that a player's distance to themselves is zero. The off-diagonal elements represent pairwise dissimilarities: $A_{12} = 1.5$ shows the Euclidean distance between Player 1 and Player 2, $A_{13} = 2.8$ is the distance between Player 1 and Player 3, and $A_{23} = 0.9$ reflects the distance between Player 2 and Player 3. The matrix is symmetric, meaning $A_{ij} = A_{ji}$ for example, the distance from Player 1 to Player 2 (A_{12}) is the same as from Player 2 to Player 1 (A_{21}). This compact structure efficiently captures all pairwise similarities (distances) between players in the graph, providing a clear representation of their relationships.

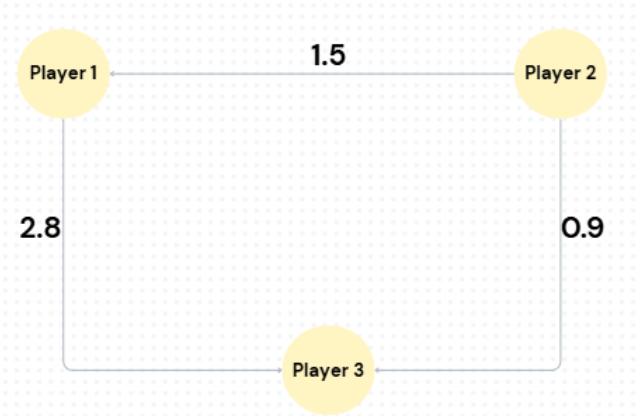

Fig. 7. 3x3 Adjacency Matrix Graph.

II. IMPLEMENTATION

The proposed graph-theoretic player recommendation system for EA Sports FC 25 is implemented in Python, leveraging key libraries such as pandas for data manipulation,

scikit-learn for data preprocessing and clustering, and *scipy.spatial.distance* for similarity calculations. The implementation is encapsulated within a Player Recommender class, designed for modularity and ease of use.

A. Data Modelling as a Graph

```
def __init__(self, data_path, n_clusters=10, random_state=42):
    """
    Initializes the PlayerRecommender.

    Args:
        data_path (str): The file path for the player dataset (CSV).
        n_clusters (int): The number of clusters for K-Means.
        random_state (int): Random state for reproducibility of clustering.
    """
    self.data_path = data_path
    self.n_clusters = n_clusters
    self.random_state = random_state
    self.df = None
    self.scaled_attributes = None
    self.distance_matrix = None
    self.player_names = None
    self.player_indices = None
    self._load_and_preprocess_data()
    self._calculate_similarity()
    self._cluster_players()

def _load_and_preprocess_data(self):
    """
    Loads the dataset and preprocesses it.

    This involves selecting relevant attributes, handling potential missing
    values, and scaling the data.
    """
    try:
        # Load the dataset
        self.df = pd.read_csv(self.data_path)
    except FileNotFoundError:
        print(f"Error: The file '{self.data_path}' was not found.")
        # Create a dummy dataframe to avoid further errors if file not found
        self.df = pd.DataFrame({
            'Name': ['Messi', 'Ronaldo', 'Mbappe', 'De Bruyne', 'Haaland'],
            'OVR': [93, 92, 91, 91, 91],
            'PAC': [85, 89, 97, 76, 89],
            'SHO': [92, 93, 89, 86, 91],
            'PAS': [91, 81, 80, 93, 65],
            'DRI': [95, 87, 92, 86, 80],
            'DEF': [34, 34, 36, 61, 45],
            'PHY': [64, 75, 76, 78, 88]
        })
        print("Using a dummy dataset for demonstration.")
```

```
# Define the core attributes to be used for similarity calculation
# These attributes cover all major aspects of a player's ability
attributes = [
    'Acceleration', 'Sprint Speed', 'Positioning', 'Finishing', 'Shot Power',
    'Long Shots', 'Volleys', 'Penalties', 'Vision', 'Crossing', 'Free Kick Accuracy',
    'Short Passing', 'Long Passing', 'Curve', 'Agility', 'Balance', 'Reactions',
    'Ball Control', 'Dribbling', 'Composure', 'Interceptions', 'Heading Accuracy',
    'Def Awareness', 'Standing Tackle', 'Sliding Tackle', 'Jumping', 'Stamina',
    'Strength', 'Aggression'
]

# For the dummy dataset, we'll use a smaller set of attributes
if 'OVR' in self.df.columns and 'Acceleration' not in self.df.columns:
    attributes = ['OVR', 'PAC', 'SHO', 'PAS', 'DRI', 'DEF', 'PHY']

# Filter out goalkeepers and select only the relevant attribute columns
if 'Position' in self.df.columns:
    self.df = self.df[self.df['Position'] != 'GK'].copy()

# Fill any potential missing values with the median of the column
for col in attributes:
    if col in self.df.columns:
        self.df[col] = pd.to_numeric(self.df[col], errors='coerce')
        self.df[col].fillna(self.df[col].median(), inplace=True)
    else:
        print(f"Warning: Attribute '{col}' not found in the dataset. It will be ignored.")


# Ensure all required attributes exist before proceeding
final_attributes = [attr for attr in attributes if attr in self.df.columns]
if not final_attributes:
    raise ValueError("None of the specified attributes were found in the dataset.")


# Store player names for later retrieval
self.player_names = self.df['Name'].copy()
# Create a mapping from player name to index for quick lookups
self.player_indices = pd.Series(self.df.index, index=self.df['Name'])

# Normalize the attributes using StandardScaler
scaler = StandardScaler()
self.scaled_attributes = scaler.fit_transform(self.df[final_attributes])
```

Fig. 8. Python code for data modelling as a graph.

The initial phase of implementation involves translating the conceptual graph structure into practical data representations within the Python environment. Player data, typically sourced from a CSV file, is loaded into a *pandas.DataFrame*. Each row in this Data Frame inherently represents a vertex in our graph, corresponding to an individual player. Relevant numerical attributes, such as Pace, Shooting, Passing, Dribbling, Defending, and Physicality, along with numerous granular attributes (e.g., Acceleration, Vision, Composure), are extracted to form the multi-dimensional feature vectors for each player. To ensure fair comparisons and prevent attributes with larger numerical ranges from dominating similarity calculations, these raw attribute values undergo a crucial normalization step using *sklearn.preprocessing.StandardScaler*. This process transforms the data to have a mean of zero and unit variance, aligning all attributes to a comparable scale. The player names and player indices (a mapping from player name to their Data Frame index) are also stored to facilitate efficient lookups and human-readable outputs. This preprocessed and scaled attribute matrix (*self.scaled_attributes*) effectively becomes the foundation upon which the graph's vertices and their inherent properties are defined, ready for the computation of relationships.

B. Similarity Measurement

```
def _calculate_similarity(self):
    """
    Calculates the similarity between all pairs of players.

    This method computes the Euclidean distance between the normalized
    attribute vectors of the players and creates a square distance matrix.
    """
    # pdist calculates the condensed distance matrix (upper triangle)
    condensed_distance_matrix = pdist(self.scaled_attributes, 'euclidean')
    # squareform converts it into a full, symmetric distance matrix
    self.distance_matrix = squareform(condensed_distance_matrix)
    print("Similarity (distance matrix) calculated successfully.")
```

Fig. 9. Python code for calculating similarity.

With the player attributes normalized, the next critical step is to quantify the relationships between all pairs of players, which form the edges of our weighted graph. The Euclidean distance is chosen as the primary similarity metric. The scipy.spatial.distance.pdist function is employed to efficiently compute the pairwise Euclidean distances between all rows (players) in the *self.scaled_attributes* matrix. This function returns a condensed distance matrix, which is then converted into a full, symmetric distance matrix using *scipy.spatial.distance.squareform*. This *self.distance_matrix* explicitly represents the weighted adjacency matrix of our graph. Each entry A_{ij} in this matrix stores the Euclidean distance between player i and player j. A lower distance value indicates a higher degree of similarity between the players, directly reflecting the weight of the conceptual edge connecting them. This pre-computation of all pairwise similarities is a foundational step, allowing for rapid retrieval of similarity information when generating recommendations.

C. Clustering Players

```
def _cluster_players(self):
    """
    Groups players into clusters based on their attributes.

    Uses the K-Means algorithm on the scaled attributes to categorize
    players into distinct groups. The cluster labels are added to the main dataframe.
    """
    kmeans = KMeans(n_clusters=self.n_clusters, random_state=self.random_state, n_init=10)
    # We fit on the scaled attributes, as K-Means also relies on distance
    self.df['Cluster'] = kmeans.fit_predict(self.scaled_attributes)
    print(f"Players clustered into {self.n_clusters} groups successfully.")
```

Fig. 10. Python code for clustering players.

To enhance the efficiency and relevance of recommendations, particularly in a large player database,

players are grouped into homogeneous clusters based on their attributes. The K-Means Clustering algorithm, from *sklearn.cluster*, is applied to the *self.scaled*_attributes matrix. The n_clusters parameter, configured during the Player Recommender initialization (e.g., n_clusters=15), dictates the number of distinct player groups the algorithm will form. The random state parameter ensures reproducibility of the clustering results across different runs. The n_init=10 parameter is also specified to run the K-Means algorithm multiple times with different centroid seeds and choose the best result, mitigating the risk of converging to suboptimal local minima. After fitting the K-Means model, each player in the original Data Frame is assigned a Cluster label. This clustering step serves to segment the player base into categories like 'attacking midfielders', 'defensive midfielders', 'central defenders', and 'wingers', making the subsequent recommendation process more focused. When a recommendation is sought for a particular player, the system can first identify their cluster, then search for similar players primarily within that cluster, significantly reducing the search space and increasing the likelihood of finding tactically relevant alternatives.

D. Recommendation System

```python
def get_recommendations(self, player_name, k=5):
    """
    Provides k similar player recommendations for a given target player.

    Args:
        player_name (str): The name of the player to get recommendations for.
        k (int): The number of similar players to recommend.

    Returns:
        pandas.DataFrame: A DataFrame containing the top k recommended players,
                          sorted by similarity (distance).
    """
    if player_name not in self.player_indices:
        # Find the closest match to the provided player name
        from difflib import get_close_matches
        matches = get_close_matches(player_name, self.player_names, n=1, cutoff=0.6)
        if not matches:
            return f"Player '{player_name}' not found in the dataset."
        player_name = matches[0]
        print(f"Player not found. Did you mean '{player_name}'?")


    # 1. Get the index and cluster of the target player
    player_idx = self.player_indices[player_name]
    player_cluster = self.df.loc[player_idx, 'Cluster']

    # 2. Get the distances from the target player to all other players
    player_distances = self.distance_matrix[player_idx]

    # 3. Create a DataFrame with player names, distances, and clusters
    recommendations_df = pd.DataFrame({
        'Name': self.player_names,
        'Distance': player_distances,
        'Cluster': self.df['Cluster']
    })

    # 4. Filter for players in the same cluster, excluding the player themselves
    cluster_recommendations = recommendations_df[
        (recommendations_df['Cluster'] == player_cluster) &
        (recommendations_df['Name'] != player_name)
    ]

    # 5. Sort the players by distance (lower is more similar) and return the top k
    top_k_recommendations = cluster_recommendations.sort_values(by='Distance').head(k)

    # Merge with original dataframe to show player attributes
    result = pd.merge(top_k_recommendations, self.df, on='Name')
    return result[['Name', 'Distance', 'OVR' if 'OVR' in self.df.columns else 'PAC',
                   'Position' if 'Position' in self.df.columns else 'SHO']]
```

Fig. 11. Python code for recommendation main function

The core functionality of providing player recommendations is encapsulated in the *get_recommendations* method. When a manager requests recommendations for a *player_name*, the system first attempts to locate this player in the dataset. A robust fuzzy matching mechanism using *difflib.get_close_matches* is incorporated to handle minor spelling variations or incomplete inputs, suggesting the closest match if the exact name isn't found. Once the target player's index (*player_idx*) and their assigned Cluster are identified, the system proceeds to retrieve the pre-calculated distances from the *self.distance_matrix*. Specifically, self.distance_matrix[*player_idx*] yields a one-dimensional array containing the Euclidean distances from the target player to every other player in the dataset. This array effectively represents the "shortest paths" (direct distances) from the target player to all other nodes in the complete, weighted graph.

A pandas Data Frame is then constructed to consolidate player names, their distances to the target player, and their respective cluster assignments. The recommendation logic then applies two crucial filters: it includes only players belonging to the same cluster as the target player and explicitly excludes the target player themselves from the results to avoid self-recommendation. Finally, the filtered recommendations are sorted in ascending order by their Distance (lower distance indicates higher similarity), and the top k players are returned. The output Data Frame typically includes essential information such as 'Name', 'Distance', 'OVR', and 'Position', providing a concise and actionable list of highly similar and tactically relevant alternatives for the manager to consider, directly leveraging the graph's structure to identify "hidden gems" and budget-friendly options that fit a specific tactical need.

E. Example Output of the Recommendation System

```
Similarity (distance matrix) calculated successfully.
Players clustered into 15 groups successfully.

--- Finding recommendations for: Kevin De Bruyne ---
Top 20 most similar players to Kevin De Bruyne:
                    Name  Distance  OVR Position
0          Bruno Fernandes  2.707212   87      CAM
1          Martin Ødegaard  2.858904   89       CM
2           İlkay Gündoğan  3.047296   87       CM
3      Alexis Mac Allister  3.111400   86       CM
4         Hakan Çalhanoğlu  3.427303   86      CDM
5             Douglas Luiz  3.471140   83       CM
6          Marcel Sabitzer  3.500762   84       CM
7              Luka Modrić  3.838633   86       CM
8       Henrikh Mkhitaryan  3.987483   83       CM
9   Trent Alexander-Arnold  4.040713   86       RB
10           Jonas Hofmann  4.067295   82      CAM
11           Jonas Hofmann  4.067295   64       CM
12        Teun Koopmeiners  4.101230   83       CM
13             Pascal Groß  4.106677   81      CDM
14           Lucas Paquetá  4.125972   82       LM
15          Enzo Fernández  4.144605   82       CM
16         Rodrigo De Paul  4.152155   84       CM
17            Aleix García  4.179754   84       CM
18      Lorenzo Pellegrini  4.257162   83      CAM
19                    Koke  4.268837   83      CDM
20         Joshua Kimmich  4.273294   86       RB
```

Fig. 12. Output of the Recommendation System

The following image displays a sample output from the implemented Player Recommendation System, showcasing recommendations for "Kevin De Bruyne". This output highlights the system's ability to identify similar players based on attribute-based Euclidean distances and K-Means clustering, presenting them with their name, calculated distance, Overall Rating (OVR), and Position.

III. Conclusion

A. Key Findings

This research demonstrates that a weighted graph effectively models player similarity in EA Sports FC 25. By representing players as vertices and attribute distances as edge weights, intricate relationships can be captured. Furthermore, the application of clustering algorithms, such as K-Means, successfully groups players based on their statistical attributes, often correlating well with their primary positions and playing styles (e.g., grouping all "Fast Wingers" together) .

B. Limitation

Despite its effectiveness, this system has certain limitations:

- Static Attributes Assumption: The current model assumes that player attributes are static. It does not account for the dynamic changes in player form, development, or real-world performance that might influence their perceived similarity or value.
- Computational Complexity: For a complete graph with thousands of vertices (players), the computation of the full adjacency matrix and the shortest path for every pair can become computationally intensive .

C. Future Work

Several avenues exist for future enhancement of this system:

- Dynamic Graphs: Integrating dynamic graph theory concepts to model changes in player attributes over time, incorporating form fluctuations or player development curves .
- Graph Coloring for Scheduling: Exploring the application of graph coloring theory to optimize match scheduling or tournament brackets, minimizing conflicts based on team or player availability.

Acknowledgment

I am grateful to Almighty God for successfully completing the IF1220 Discrete Mathematics paper. My sincere thanks go to Mr. Dr. Ir. Rinaldi Munir, M.T., as the class lecturer, who has provided the necessary understanding and knowledge to complete this paper, as well as offering various learning media and references that were very helpful in working on this discrete mathematics paper. Furthermore, I extend my gratitude to my family, friends, seniors, and everyone in the academic environment who has provided support in completing this paper. I would also like to express special thanks to CJM, the EA Sports FC game YouTuber, who provided me with inspiration for the title of this paper. It is my hope that this paper will not only serve as a mere academic assignment but also be beneficial for various audiences and act as a motivation in the field of education. I also hope that the discussions within this paper can continue to be expanded and developed further. Finally, this paper is not perfect and still has shortcomings. Regarding this, the author sincerely apologizes for any imperfections.

References

[1] E. Arts, "EA SPORTS FC 25 | Pitch Notes - Career Mode Deep Dive," EA, Aug. 07, 2024. https://www.ea.com/games/ea-sports-fc/fc-25/news/pitch-notes-fc-25-career-mode-deep-dive (accessed Jun. 9, 2025).

[2] R. Munir, "Matematika Diskrit," Itb.ac.id, 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/matdis.htm (accessed Jun. 10, 2025).

[3] Junjie Wu, Advances in K-means clustering: a data mining thinking. Berlin; New York: Springer, 2012.

[4] M. Muldoon, "MATH20902 Discrete Mathematics || The University of Manchester | School of Mathematics," Manchester.ac.uk, 2020. https://personalpages.manchester.ac.uk/staff/mark.muldoon/Teaching/Discrete Maths/CourseMate rials WithPodcasts.htm (accessed Jun. 20, 2025).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2025

Reinsen Silitonga 13524093