Network Flow Models in Football: Maximizing Passing Efficiency

Owen Tobias Sinurat - 13522131
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13522131@std.stei.itb.ac.id

***Abstract*— This paper delves into the application of network flow models in the context of football strategy, specifically focusing on the optimization of passing efficiency within a team. Leveraging principles from discrete mathematics, particularly graph theory, the study explores the dynamics of passing networks and their impact on overall team performance. By representing player interactions as a directed graph, where nodes represent players and edges denote passes, we employ network flow algorithms to analyze and optimize passing efficiency. The objective is to identify key players, strategic passing patterns, and configurations that maximize ball circulation and enhance the team's overall performance. The findings aim to contribute valuable insights to coaches, analysts, and researchers interested in leveraging mathematical modeling to improve football strategy and player coordination.**

***Keywords*—Network Flow Models, Graph, Graph Application in Football, Passing in Football, Discrete Mathematics in Football.**

I. Introduction

In the realm of sports analytics, the application of mathematical models has become increasingly pivotal in unraveling the intricacies of game dynamics and player interactions. Football, as a sport characterized by its fluidity and dynamic team play, provides a fertile ground for the exploration of innovative analytical approaches. This paper delves into the integration of network flow models derived from discrete mathematics to scrutinize and optimize passing efficiency in football.

The passing network within a football team is analogous to the intricate web of connections in a network, where players represent nodes and passes between them serve as directed edges. Drawing inspiration from the field of network theory, our research aims to unveil the underlying patterns and dynamics of passing interactions among players. The central objective is to employ network flow algorithms to identify optimal passing strategies that maximize ball circulation, ultimately contributing to enhanced team performance on the pitch.

As we navigate through this exploration, the significance of network flow models in dissecting the passing intricacies in football becomes evident. By treating each player as a vital node and passes as essential edges, we embark on a journey to analyze how the flow of possession can be strategically optimized. This paper not only presents a theoretical framework for the application of network flow models but also seeks to provide practical insights for coaches, analysts, and football enthusiasts interested in leveraging mathematical tools to refine team strategies.

The subsequent sections will delve into the foundational principles of network flow models, their adaptation to the football context, and the implications of our findings on understanding and improving passing efficiency in the beautiful game. Through this exploration, we aim to contribute to the evolving landscape of sports analytics, where the marriage of mathematics and football strategy opens new avenues for elevating the performance of teams on the field.

II. Theory

A. Graph

A graph G is an ordered pair (V (G),E(G)) consisting of a set V (G) of vertices and a set E(G), disjoint from V (G), of edges, together with an incidence function ψG that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G. If e is an edge and u and v are vertices such that ψG(e) = {u,v}, then e is said to join u and v, and the vertices u and v are called the ends of e. We denote the numbers of vertices and edges in G by v(G) and e(G); these two basic parameters are called the order and size of G, respectively. Two examples of graphs should serve to clarify the definition. For notational simplicity, we write uv for the unordered pair {u,v}.

Example 1.

$$G = (V(G), E(G))$$

where

$$V(G) = \{u, v, w, x, y\}$$
$$E(G) = \{a, b, c, d, e, f, g, h\}$$

and ψG is defined by

$$\psi G(a) = uv \; \psi G(b) = uu \; \psi G(c) = vw \; \psi G(d) = wx$$
$$\psi G(e) = vx \; \psi G(f) = wx \; \psi G(g) = ux \; \psi G(h) = xy$$

Drawing of Graphs

Graphs are so named because they can be represented graphically, and it is this graphical representation which helps

us understand many of their properties. Each vertex is indicated by a point, and each edge by a line joining the points representing its ends. Diagrams of G and H are shown in Figure 1.1. (For clarity, vertices are represented by small circles.)

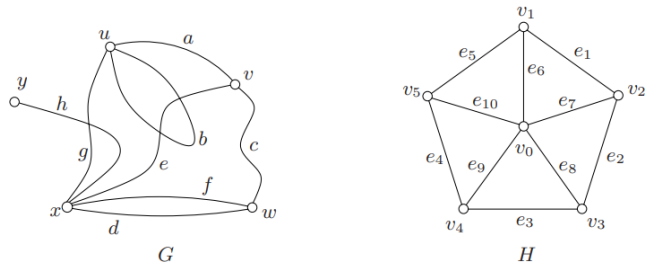


Fig. 2.1. *Diagrams of the graphs G and H*
Source: [3]

There is no single correct way to draw a graph; the relative positions of points representing vertices and the shapes of lines representing edges usually have no significance. In Figure 1.1, the edges of G are depicted by curves, and those of H by straight-line segments. A diagram of a graph merely depicts the incidence relation holding between its vertices and edges. However, we often draw a diagram of a graph and refer to it as the graph itself; in the same spirit, we call its points 'vertices' and its lines 'edges'.

Most of the definitions and concepts in graph theory are suggested by this graphical representation. The ends of an edge are said to be incident with the edge, and vice versa. Two vertices which are incident with a common edge are adjacent, as are two edges which are incident with a common vertex, and two distinct adjacent vertices are neighbours. The set of neighbours of a vertex v in a graph G is denoted by NG(v).

An edge with identical ends is called a loop, and an edge with distinct ends a link. Two or more links with the same pair of ends are said to be parallel edges. In the graph G of Figure 1.1, the edge b is a loop, and all other edges are links; the edges d and f are parallel edges.

Path and Cycle

A path is a simple graph whose vertices can be arranged in a linear sequence in such a way that two vertices are adjacent if they are consecutive in the sequence, and are nonadjacent otherwise. Likewise, a cycle on three or more vertices is a simple graph whose vertices can be arranged in a cyclic sequence in such a way that two vertices are adjacent if they are consecutive in the sequence, and are nonadjacent otherwise; a cycle on one vertex consists of a single vertex with a loop, and a cycle on two vertices consists of two vertices joined by a pair of parallel edges. The length of a path or a cycle is the number of its edges. A path or cycle of length k is called a k-path or k-cycle, respectively; the path or cycle is odd or even according to the parity of k. A 3-cycle is often called a triangle, a 4-cycle a quadrilateral, a 5-cycle a pentagon, a 6-cycle a hexagon, and so on. Figure 1.3 depicts a 3-path and a 5-cycle.

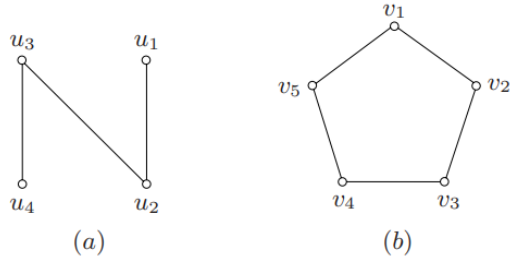


Fig. 2.2. (a) *A path of length three, and* ***(b)*** *a cycle of length five.*
Source: [3]

Connected Graph

Certain types of graphs play prominent roles in graph theory. A graph is *connected* if, for every partition of its vertex set into two nonempty sets X and Y , there is an edge with one end in X and one end in Y ; otherwise the graph is *disconnected*. In other words, a graph is disconnected if its vertex set can be partitioned into two nonempty subsets X and Y so that no edge has one end in X and one end in Y . (It is instructive to compare this definition with that of a bipartite graph.) Examples of connected and disconnected graphs are displayed in Figure 1.4.

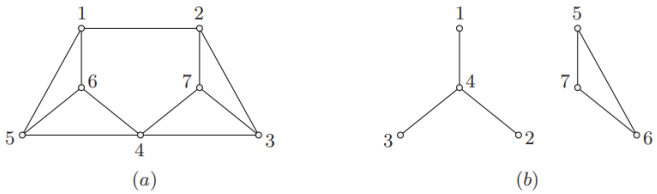


Fig. 2.3. (a) *A connected graph, and* ***(b)*** *a disconnected graph*
Source: [3]

Directed Graph

Although many problems lend themselves to graph-theoretic formulation, the concept of a graph is sometimes not quite adequate. When dealing with problems of traffic flow, for example, it is necessary to know which roads in the network are one-way, and in which direction traffic is permitted. What we need is a graph in which each link has an assigned orientation, namely a directed graph.

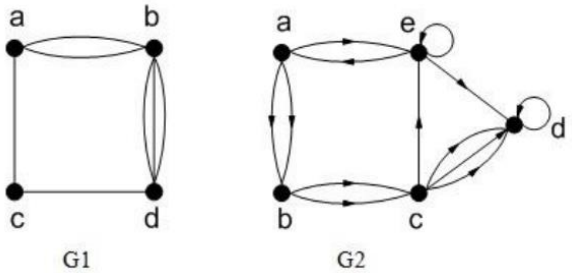


Fig. 2.4. *(G1) an undirected graph and (G2) a directed graph.*
Source: [2]

Formally, a directed graph *D* is an ordered pair *(V (D), A(D))* consisting of a set *V := V (D)* of vertices and a set *A := A(D)*, disjoint from V (D), of arcs, together with an incidence function ψD that associates with each arc of *D* an ordered pair of (not necessarily distinct) vertices of *D*. If a is an arc and *ψD(a) = (u,v)*, then a is said to join u to v; we also say that u dominates

v. The vertex u is the tail of a, and the vertex v its head; they are the two ends of a. Occasionally, the orientation of an arc is irrelevant to the discussion. In such instances, we refer to the arc as an edge of the directed graph. The number of arcs in *D* is denoted by *a(D).* The vertices which dominate *a* vertex *v* are its in-neighbours, those which are dominated by the vertex its out-neighbours. These sets are denoted by *N – D (v)* and *N + D (v)*, respectively.

B. Flow Network

A **flow network** $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative **capacity** $c(u, v) \geq 0$. We further require that if E contains an edge (u, v), then there is no edge (v, u) in the reverse direction. (We shall see shortly how to work around this restriction.) If $(u, v) \notin E$, then for convenience we define $c(u, v) = 0$, and we disallow self-loops. We distinguish two vertices in a flow network: a **source** s and a **sink** *t*. For convenience, we assume that each vertex lies on some path from the source to the sink. That is, for each vertex $v \in V$, the flow network contains a path $s \rightsquigarrow v \rightsquigarrow t$. The graph is therefore connected and, since each vertex other than s has at least one entering edge, $|E| \geq |V| - 1$. Figure 26.1 shows an example of a flow network. We are now ready to define flows more formally. Let $G = (V, E)$ be a flow network with a capacity function *c*. Let *s* be the source of the network, and let *t* be the sink. A ***flow*** in G is a real-valued function $f : V \times V \rightarrow \mathbb{R}$ R that satisfies the following two properties:

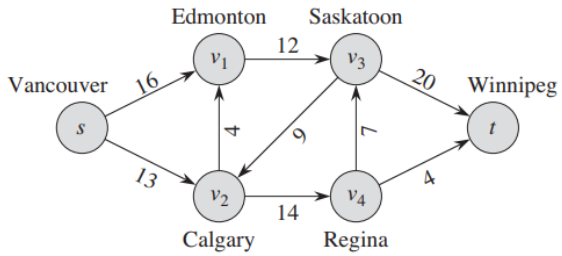
Capacity constraint: For all $u, v \in V$, we require

$$0 \leq f(u, v) \leq c(u, v).$$

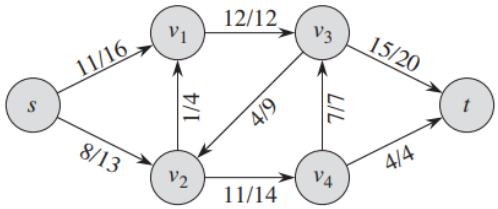
Flow conservation: For all $u \in V - \{s, t\}$, we require

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

When $(u, v) \notin E$, there can be no flow from *u* to *v*, and $f(u, v) = 0$.



(a)



(b)

Fig. 2.5. (a) A flow network $G = (V, E)$ for the Lucky Puck Company's trucking problem. The Vancouver factory is the source *s*, and the Winnipeg warehouse is the sink *t*. The company ships pucks through intermediate cities, but only $c(u, v)$ crates per day can go from city u to city . Each edge is labeled with its capacity. **(b)** A flow f in G with value $|f| = 19$. Each edge (u, v) is labeled by $f(u, v)/c(u, v)$. The slash notation merely separates the flow and capacity; it does not indicate division.
Source: [1]

We call the nonnegative quantity $f(u, v)$ the flow from vertex *u* to vertex *v*. The **value** of j of a flow f is defined as

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

that is, the total flow out of the source minus the flow into the source. (Here, the jj notation denotes flow value, not absolute value or cardinality.) Typically, a flow network will not have any edges into the source, and the flow into the source, given by the summation $\sum_{v \in V} f(v, s)$ will be 0. We include it, however, because when we introduce residual networks later in this chapter, the flow into the source will become significant. In the **maximum-flow problem**, we are given a flow network G with source s and sink t, and we wish to find a flow of maximum value. Before seeing an example of a network-flow problem, let us briefly explore the definition of flow and the two flow properties. The capacity constraint simply says that the flow from one vertex to another must be nonnegative and must not exceed the given capacity. The flow-conservation property says that the total flow into a vertex other than the source or sink must equal the total flow out of that vertex—informally, "flow in equals flow out."

C. The Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm depends on three important ideas that transcend the method and are relevant to many flow algorithms and problems: residual networks, augmenting paths, and cuts. These ideas are essential to the important max-flow min-cut theorem, which characterizes the value of a maximum flow in terms of cuts of the flow network. We end this section by presenting one specific implementation of the Ford-Fulkerson method and analyzing its running time. The Ford-Fulkerson method iteratively increases the value of the flow. We start with $f(u, v) = 0$ for all $u, v \in V$, giving an initial flow of value 0. At each iteration, we increase the flow value in G by finding an "augmenting path" in an associated "residual network" G_f . Once we know the edges of an augmenting path in G_f , we can easily identify specific edges in G for which we can change the flow so that we increase the value of the flow. Although each iteration of the Ford-Fulkerson method increases

the value of the flow, we shall see that the flow on any particular edge of G may increase or decrease; decreasing the flow on some edges may be necessary in order to enable an algorithm to send more flow from the source to the sink. We repeatedly augment the flow until the residual network has no more augmenting paths. The max-flow min-cut theorem will show that upon termination, this process yields a maximum flow.

Ford-Fulkerson Algorithm (*G,s,t)*
1 intialize flow *f* to 0
2 **while** there exists an augmenting path *p* in the
3 residual network G_f augmented flow *f* along *p*
4 **return** *f*

Residual Networks

Intuitively, given a flow network G and a flow f, the residual network Gf consists of edges with capacities that represent how we can change the flow on edges of G. An edge of the flow network can admit an amount of additional flow equal to the edge's capacity minus the flow on that edge. If that value is positive, we place that edge into Gf with a "residual capacity" of $c_f(u,v) = c(u,v) - f(u,v)$. The only edges of G that are in Gf are those that can admit more flow; those edges (u,v) whose flow equals their capacity have $c_f(u,v) = 0$, and they are not in G_f. The residual network G_f may also contain edges that are not in G, however. As an algorithm manipulates the flow, with the goal of increasing the total flow, it might need to decrease the flow on a particular edge. In order to represent a possible decrease of a positive flow $f(u,v)$ on an edge in G, we place an edge (u,v) into G_f with residual capacity $c_f(v,u) = f(u,v)$ — that is, an edge that can admit flow in the opposite direction to (u,v), at most canceling out the flow on (u,v). These reverse edges in the residual network allow an algorithm to send back flow it has already sent along an edge. Sending flow back along an edge is equivalent to decreasing the flow on the edge, which is a necessary operation in many algorithms. More formally, suppose that we have a flow network $G = (V,E)$ with source s and sink t. Let f be a flow in G, and consider a pair of vertices $u, v \in V$. We define the ***residual capacity*** $c_f(u,v)$ by

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E, \\ f(v,u) & \text{if } (v,u) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Because of our assumption that $(u,v) \in E$ implies $(v,u) \notin E$, exactly one case from the equation above applies to each ordered pair of vertices.

As an example of equation above, if $c(u,v) = 16$ and $f(u,v) = 11$, then we can increase $f(u,v)$ by up to $c_f(u,v) = 5$ units before we exceed the capacity constraint on edge (u,v). We also wish to allow an algorithm to return up to 11 units of flow from v to u, and hence $c_f(v,u) = 11$.

Given a flow network $G = (V,E)$ and a flow f, the ***residual network*** of G induced by f is $G_f = (V, Ef)$, where

$$Ef = \{(u,v) \in V \times V : cf(u,v) > 0\}.$$

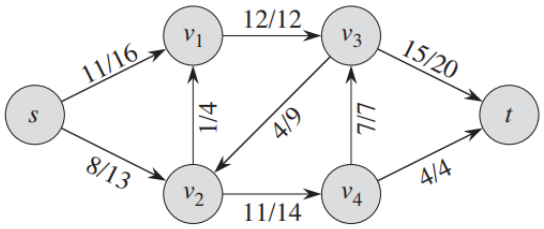
That is, as promised above, each edge of the residual network, or residual edge, can admit a flow that is greater than 0. Figure 2.6(a) repeats the flow network G and flow f of Figure 2.5(b), and Figure 2.6(b) shows the corresponding residual network G_f. The edges in Ef are either edges in E or their reversals, and thus

$$|Ef| \leq 2\,|E|$$

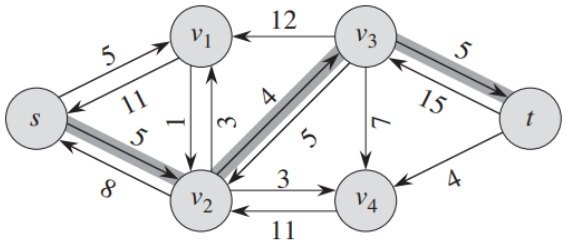
Observe that the residual network G_f is similar to a flow network with capacities given by c_f. It does not satisfy our definition of a flow network because it may contain both an edge (u,v) and its reversal (v,u). Other than this difference, a residual network has the same properties as a flow network, and we can define a flow in the residual network as one that satisfies the definition of a flow, but with respect to capacities cf in the network G_f.

A flow in a residual network provides a roadmap for adding flow to the original flow network. If f is a flow in G and f ' is a flow in the corresponding residual network G_f, we define $f \uparrow f'$, the ***augmentation*** of flow f by f', to be a function from V V to R, defined by

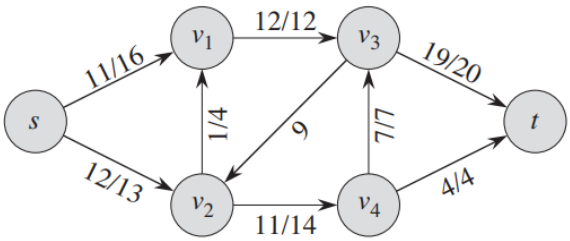
$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$



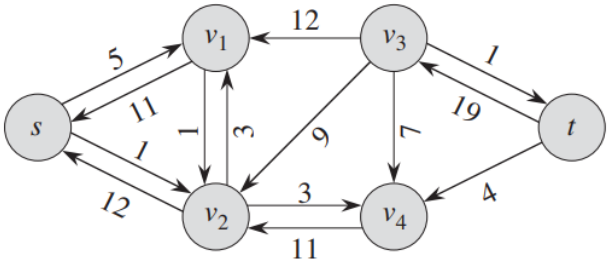
(a)



(b)



(c)



(d)

Fig. 2.6 (a) The flow network G and flow f of Figure 2.5(b). **(b)** The residual network G_f with augmenting path p shaded; its residual capacity is $cf(p) = cf(v2, v3) = 4$. Edges with residual capacity equal to 0, such as $(v1, v3)$, are not shown, a convention we follow in the remainder of this section. **(c)** The flow in G that results from augmenting along path p by its residual capacity 4. Edges carrying no flow, such as $(v3, v2)$, are labeled only by their capacity, another convention we follow throughout. **(d)** The residual network induced by the flow in (c).
Source: [1]

The intuition behind this definition follows the definition of the residual network. We increase the flow on (u, v) by $f'(u, v)$ but decrease it by $f'(v, u)$ because pushing flow on the reverse edge in the residual network signifies decreasing the flow in the original network. Pushing flow on the reverse edge in the residual network is also known as ***cancellation***. For example, if we send 5 crates of hockey pucks from u to v and send 2 crates from v to u, we could equivalently (from the perspective of the final result) just send 3 creates from u to and none from to u. Cancellation of this type is crucial for any maximum-flow algorithm.

III. METHODOLOGY

A. Research Object

In this paper, all the passing information from Manchester United's draw against Burnley in the English Premier League 2016-2017 season is taken as the sample to establish and optimize the passing network.

B. Graph Representation

To rigorously analyze and optimize the passing strategies in football, we employ graph theory—a mathematical framework that encapsulates the intricate connections between players and their passing interactions. The football passing network is represented as a graph, where players are nodes, and passing connections between them are edges.

Graph Construction

1. Node as Players.
   Each player on the field is represented as a node in the graph. The positions of these nodes correspond to the spatial distribution of players during gameplay.

Fig. 3.1. A node for player Pogba

2. Edges as Passing Connections.
   Passing connections between players form the edges of the graph. These edges denote potential passing routes, and their weights may represent passing accuracy, player skill, or other relevant metrics.

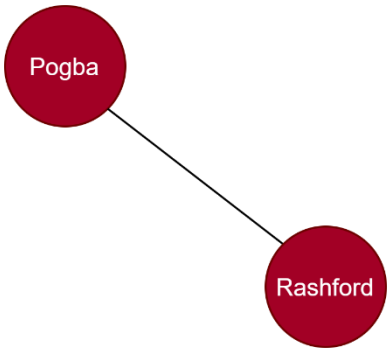


Fig. 3.2. A passing connection between player Pogba and player Rashford.

1. Directed Graph.
   The graph is directed, indicating the direction of passes. A directed edge from Node A to Node B signifies a potential pass from player A to player B.

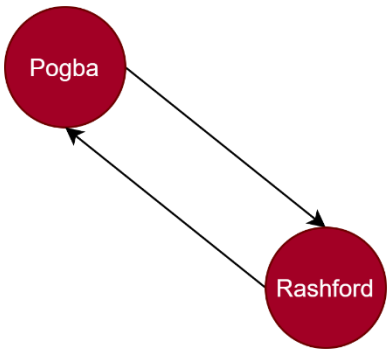
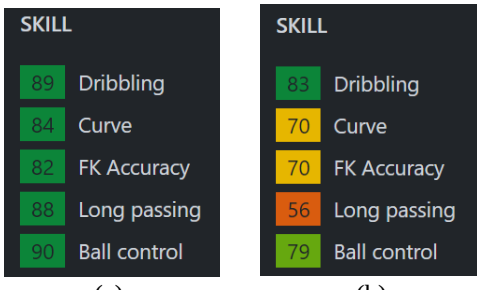
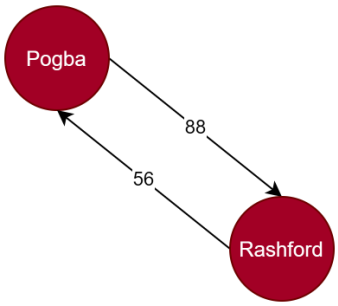


Fig. 3.3. Pogba is able to pass to Rashford and *vice versa.*

2. Edge Weights as Player's Passing Rating.
   Edge weights may be assigned based on player's passing rating. A higher weight denotes a higher likelihood of successful passes along that connection. All players statistic related to the research can be accessed from the website https://sofifa.com (use the 2017 version). I am using the 'Long passing' parameter as the weight.



(a) (b)



(c)

Fig. 3.4 (a) Pogba skill statistic, **(b)** Rashford skill statistic, and **(c)** edge weight represented by each player passing skill Source for (a) and (b): [4]

3. Passing Network Adjustments
   There are some adjustments needed to integrate the passing in football match to the passing network:
   1. To take the enemies into consideration, I only enable short pass connection between the players which are between goalkeeper-defender, keeper-defender, defender-midfielder, midfielder-attacker, and of course everyone-anyone in their position.
   2. Players cannot move from their place, because in network flow model, nodes are stationer.
   3. To ease the algorithm processing, the edges weight will be static based on the player's skill statistic, instead of real time data of the match.

C. Network Model

With the graph representation determined, here is the constructed graph:

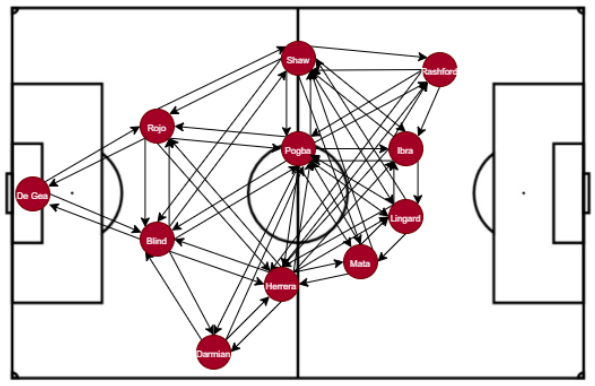


Fig. 3.5. Passing Network Model (for the sake of readability, I have stashed the edges weight).

From that network model, we are going to determine the optimized passing route to maximize the passing cycle efficiency using the Ford-Fulkerson algorithm.

D. Ford-Fulkerson Implementation

Here, I will be using Python to implement the Ford-Fulkerson algorithm.

```
  from collections import defaultdict

  # this is the class for the Ford-
Fulkerson implementation
  class FordFulkerson:
      # defining the graph and set of
visited nodes.
      def __init__(self, graph):
          self.graph = graph
          self.visited = set()

      # function to find the augmenting
path
      def find_augmenting_path(self,
source, sink, path=[]):
          self.visited.add(source)
          current_path = path + [source]

          if source == sink:
              return current_path

          for neighbor, capacity in
self.graph[source].items():
              if neighbor not in
self.visited and capacity > 0:
                  augmenting_path =
self.find_augmenting_path(neighbor, sink,
current_path)
                  if augmenting_path:
                      return
augmenting_path

          return None

      # function for to execute the
algorithm
      def ford_fulkerson_algorithm(self,
source, sink):
          max_flow = 0

          while True:
              augmenting_path =
self.find_augmenting_path(source, sink)

              if not augmenting_path:
                  break

              min_residual_capacity =
min(self.graph[u][v] for u, v in
zip(augmenting_path, augmenting_path[1:]))

              for u, v in
zip(augmenting_path, augmenting_path[1:]):
                  self.graph[u][v] -=
min_residual_capacity
                  self.graph[v][u] +=
min_residual_capacity

              max_flow +=
```

```
min_residual_capacity
              self.visited = set()

          return max_flow

  # declaring the players data
  # in this case, the source is DeGea
(goalkeeper)
  # the format of the dictionary is:
  # 'player1':{'player2':2, 'player3':4}
  # which means 'player1' can pass to
'player2' with weight 2, and to 'player3'
with weight 4.
  if __name__ == "__main__":
      football_passing_network = {
          'source': {'Rojo': 38, 'Blind':
38},
      'Rojo': {'Shaw': 73, 'Pogba': 73,
'Herrera': 73},
      'Blind': {'Shaw': 81, 'Pogba': 81,
'Herrera': 81},
      'Darmian': {'Shaw': 66, 'Pogba': 66,
'Herrera': 66},
      'Shaw': {'Rojo': 64, 'Blind': 64,
'Darmian': 64, 'Rashford': 64, 'Ibra': 64,
'Mata': 64, 'Lingard': 64, 'Pogba': 64},
      'Pogba': {'Rojo': 88, 'Blind': 88,
'Darmian': 88, 'Rashford': 88, 'Ibra': 88,
'Mata': 88, 'Lingard': 88, 'Shaw': 88,
'Herrera': 88},
      'Herrera': {'Rojo': 80, 'Blind': 80,
'Darmian': 80, 'Rashford': 80, 'Ibra': 80,
'Mata': 80, 'Lingard': 80, 'Pogba': 80},
      'Rashford': {'Pogba': 56, 'Ibra': 56,
'Mata': 56, 'Lingard': 56},
      'Ibra': {'Pogba': 76, 'Mata': 76,
'Lingard': 76},
      'Mata': {'Pogba': 77, 'Ibra': 77,
'Lingard': 77},
      'Lingard': {'Pogba': 66, 'Ibra': 66,
'Mata': 66}
  }    }

      ford_fulkerson =
FordFulkerson(football_passing_network)
      max_flow =
ford_fulkerson.ford_fulkerson_algorithm('D
eGea', 'Rashford')

      print(f"Maximum Flow: {max_flow}")
```

The Python code implements the Ford-Fulkerson algorithm to optimize a football passing network. The algorithm is encapsulated within a class named FordFulkerson, which contains three key functions. The __init__ function initializes the class with the football passing network represented as a dictionary, where players are nodes, and edges denote passing connections with associated capacities.

The find_augmenting_path function recursively employs depth-first search (DFS) to identify augmenting paths between the source and sink nodes. The recursive nature of this function facilitates the discovery of multiple augmenting paths in the passing network.

The ford_fulkerson_algorithm function orchestrates the optimization process, iteratively finding augmenting paths using find_augmenting_path and updating the network's capacities accordingly. The algorithm continues until no augmenting paths are found, signifying the completion of the optimization. This implementation allows for customization and adaptability, making it applicable to various football passing scenarios. Users can leverage the Ford-Fulkerson algorithm to dynamically optimize passing networks, enhancing strategic decision-making in football gameplay.

IV. Result and Analysis

After running through the algorithm, we should get a graph looked like the graph below.

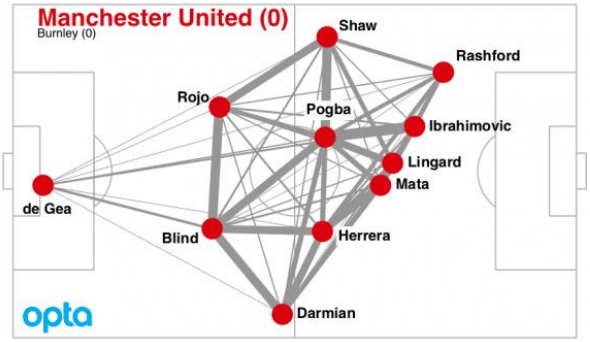


Fig. 3.6. The illustration of the result
Source: [5]

The players are the red dots, placed according to the average position they received and passed the ball. The grey lines connecting them indicate the number of successful passes pairs of players made. The thicker the line, the more passes were made. As shown in the graph, the most common pass is between Pogba and Ibrahimovic.

V. Conclusion

In conclusion, this research has delved into the application of the Ford-Fulkerson algorithm to optimize football passing networks, with a specific focus on maximizing passing efficiency among defenders, midfielders, and attackers. The algorithm, encapsulated within the FordFulkerson class, proved to be a robust tool for dynamically optimizing the network based on specified passing constraints. Through the utilization of augmenting paths, the algorithm systematically adjusted passing strategies, fostering an enhanced flow of passes within the team. The choice of representing the passing network as a directed graph, with nodes representing players and edges denoting passing connections with associated capacities, provided a clear and versatile model for optimization.

The research outcomes not only showcase the flexibility of the Ford-Fulkerson algorithm in adapting to specific football team structures but also emphasize its potential to fine-tune passing strategies in real-time scenarios. The algorithm's ability to dynamically adjust to changes in player positions, roles, and game situations highlights its applicability in dynamic and

unpredictable football environments.

While the visual representation of the optimized passing network adds a layer of interpretability, enabling coaches and analysts to gain insights into strategic passing patterns, there are areas for further exploration. Future research could delve into refining the algorithm to account for additional factors such as player fatigue, opposition strategies, and evolving match dynamics.

In summary, this research contributes to the growing field of applying discrete optimization techniques in sports analytics. The Ford-Fulkerson algorithm emerges as a valuable tool for football coaches and analysts seeking to maximize passing efficiency, ultimately influencing team performance on the field.

VI. ACKNOWLEDGMENT

The author would like to thank first of all, to God for all the guidance throughout the process of learning thus writing the contents of this paper. The author would also like to thank the lecturer of ITB Discrete Mathematics IF2120 for my class, Dr. Ir. Rinaldi Munir, M.T and Monterico Adrian, S.T, M.T and. for sharing their knowledges and guide the students throughout the learning process in the class. And also worth mentioning, the author want to thank family members and friends for their unbelievable support.

REFERENCES

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

[2] Munir, Rinaldi. (2020). "Graf: Bagian 1". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/19-Graf-Bagian1-2023.pdf (accessed on 30th November 2023).

[3] Bondy, J. A., & Murty, U. S. R. (2008). Graph Theory. Springer.

[4] SoFIFA. (2023). "Football Players Rating". https://sofifa.com/ (accessed on 30th November 2023).

[5] Sumpter, D & Andrzejewski, A. (2022). "Case study: decentralized football". https://soccermatics.readthedocs.io/en/latest/lesson1/passnetworksExample.html (accessed on 30th November 2023).

PERSONAL STATEMENT

I hereby declare that the paper I wrote is my own writing, not an adaptation, or translation of someone else's paper, and not plagiarized.

Bandung, 30th November 2023

Owen Tobias Sinurat, 13522131