**Maximum Cardinality of Bipartite Matching MCBM**

**Iheb Gafsi**\*

INSAT Student

**Iheb.engineer@gmail.com**

**Definition:**

The maximum cardinality bipartite matching problem is a combinatorial optimization problem that aims to find the maximum number of pairwise non-adjacent edges (matches) in a bipartite graph. In the context of matching, one set represents a group of agents or elements, and the other set represents another group of agents or elements. The goal is to find the largest possible set of pairs where each pair consists of one element from each group and no two pairs share a common element.

To solve the maximum cardinality bipartite matching problem, we can use the Ford-Fulkerson algorithm with a few modifications. First, we create a flow graph from the given bipartite graph, where we add a source node connected to all elements in the first set and a sink node connected to all elements in the second set. The capacity of each edge is set to 1, indicating that each edge can only carry one unit of flow. Next, we apply the Ford-Fulkerson algorithm to find the maximum flow from the source to the sink in this flow graph. The maximum flow value corresponds to the maximum number of pairwise non-adjacent edges (matches) in the original bipartite graph, which represents the solution to the maximum cardinality bipartite matching problem. The Ford-Fulkerson algorithm efficiently computes the maximum flow in the flow graph by iteratively finding augmenting paths and increasing the flow along these paths until no more augmenting paths can be found, ensuring that we obtain the optimal solution to the maximum cardinality bipartite matching problem.

**Use cases:**

The maximum cardinality bipartite matching problem has diverse applications in different fields. It is used to optimize resource allocation in computer science and information technology, transportation and logistics, and bioinformatics. In job assignment scenarios, it helps match job seekers with vacancies, while in transportation, it optimizes cargo and vehicle assignments. In bioinformatics, it analyzes molecular interactions, and in social network analysis, it connects individuals in social networks. The problem's versatility makes it a valuable tool for solving various optimization problems in different domains.

**Example Algorithm:**

In this algorithm we will only use the Ford-Fulkerson algorithm with some edition. We will just add a source node to the beginning and a sink node at the end giving a capacity to every edge and calculating the maximum flow which is going to be the maximum cardinality bipartite matching.

1. # Variables

2. INF = float('inf')

3. # Edge Class

4. class Edge:

5.     def \_\_init\_\_(self, back, front, capactiy):

6.         self.back = back

7.         self.front = front

8.         self.capacity = capactiy

9.         self.residual = None

10.         self.flow = 0

11.     def isResidual(self):

12.         return self.capacity == 0

13.     def remaining\_capactiy(self):

14.         return self.capacity - self.flow

15.     def augment(self, bottleNeck):

16.         self.flow += bottleNeck

17.         self.residual.flow -= bottleNeck

18.

19. class FlowNetwork:

20.     def \_\_init\_\_(self, n, source, sink):

21.         self.n = n

22.         self.source = source

23.         self.sink = sink

24.         self.graph = [[] for \_ in range(n)]

25.         self.visited = [0] \* n

26.         self.visitedToken = 1

27.         self.max\_flow = 0

28.

29.     def add\_edge(self, back, front, capacity):

30.         edge = Edge(back, front, capacity)

31.         residual = Edge(front, back, 0)

32.         edge.residual = residual

33.         residual.residual = edge

34.         self.graph[back].append(edge)

35.         self.graph[front].append(residual)

36.     # Ford Fulkerson Algorithm

37.     def dfs(self, node, flow):

38.         if node == self.sink: return flow

39.

40.         self.visited[node] = self.visitedToken

41.         edges = self.graph[node]

42.         for edge in edges:

43.             if edge.remaining\_capactiy()>0 and self.visited[edge.front] != self.visitedToken:

44.                 bottlneck = self.dfs(edge.front, min(flow, edge.remaining\_capactiy()))

45.                 if bottlneck >0:

46.                     edge.augment(bottlneck)

47.                     return bottlneck

48.         return 0

49.

50.     def find\_max\_flow(self):

51.         f = self.dfs(self.source, INF)

52.         while f!=0:

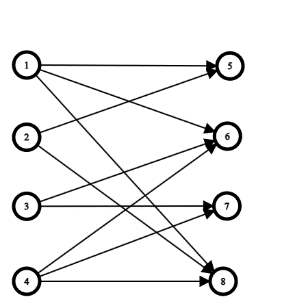
53.             self.visitedToken += 1

54.             self.max\_flow += f

55.             f = self.dfs(self.source, INF)

56.         return self.max\_flow

Here’s a small example for finding the MCBM:



We will use the Python code down below to outline the output of the algorithm on this graph:

1. #Application

2. ford = FlowNetwork(10, 0, 9)

3. # add edges from the source node 0 to the input nodes

4. ford.add\_edge(0, 1, 1)

5. ford.add\_edge(0, 2, 1)

6. ford.add\_edge(0, 3, 1)

7. ford.add\_edge(0, 4, 1)

8. # add edges of the graph

9. ford.add\_edge(1, 5, 1)

10. ford.add\_edge(1, 6, 1)

11. ford.add\_edge(1, 8, 1)

12. ford.add\_edge(2, 5, 1)

13. ford.add\_edge(2, 8, 1)

14. ford.add\_edge(3, 6, 1)

15. ford.add\_edge(3, 7, 1)

16. ford.add\_edge(4, 6, 1)

17. ford.add\_edge(4, 7, 1)

18. ford.add\_edge(4, 8, 1)

19. # add edges from output nodes to the sink node

20. ford.add\_edge(5, 9, 1)

21. ford.add\_edge(6, 9, 1)

22. ford.add\_edge(7, 9, 1)

23. ford.add\_edge(8, 9, 1)

24.

25. mcbm = ford.find\_max\_flow()

26. print(mcbm)

The corresponding output is:

Python>> 4

