BIPARTITE MATCHING

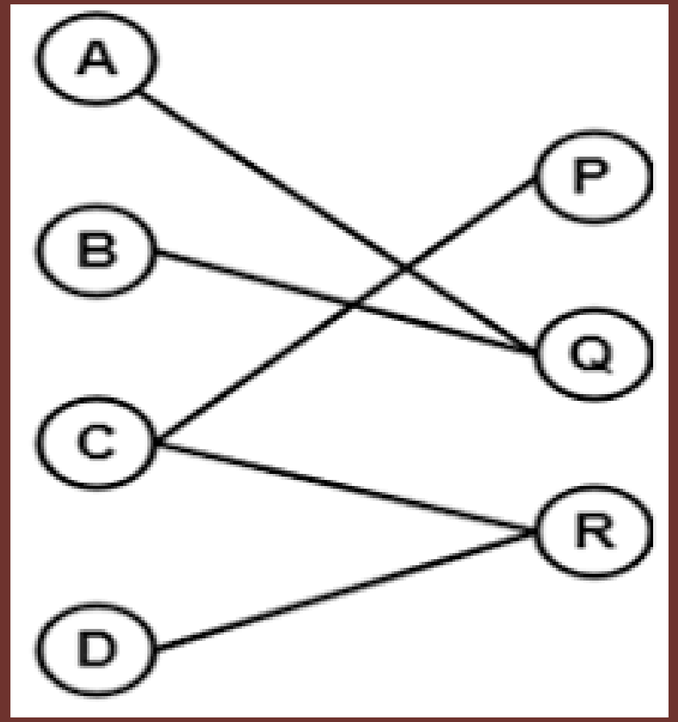
Syed Haroon Shoaib
Yashwanth Karuparthi

DAA Presentation

Introduction

Bipartite graph

Bipartite graph (bigraph) has vertices that are divided into two disjoint and independent sets U and V, where every edge connects a vertex in U to one in V.



Bipartite matching

Matching in Bigraph is a set of edges chosen in such a way that no two edges share an endpoint

The objective of bigraph matching is to find the largest possible matching in the given graph

It ensures that maximum possible vertices are paired without violating the bipartite property

Paper Summary

Load Balancing Algorithm Based on Weighted Bipartite Graph for Edge Computing

- Presents load balancing algo for edge computing, to achieve efficient utilization of network edge resources
- Addresses the issue of each computing nodes that is made to carry task requests beyond its limits which delays completion of tasks
- Proposes LBA-EC which adopts 2 phases:
  - Initially matching tasks to different edge servers
  - Optimal allocation of tasks to server based on energy consumption and completion time indicators

Edge-Weighted Online Bipartite Matching

- Deals with matching problem where one side of bipartite graph is known in advance, and other side arrives one by one
- Goal is to assign the arriving vertices to ones on other side while maximizing the total weight of the assigned edges
- Proposes an efficient algorithm that leverages a subroutine, Online Correlation Selection (OCS), which selects vertices from pairs while negatively correlating the decisions across different pairs

Paper Summary

Optimizing Bipartite Matching in Real-World Applications by Incremental Cost Computation

- Uses Kuhn-Munkres algo for minimum cost bigraph matching in real world appplications
- Computing the edge weights outweighs the computation of optimal assignment itself
- Proposes a new algorithm that reduces the cost of computing the edge weights by only calculating those that are most likely to be in the optimal solution
- This is achieved by using incremental Kuhn-Munkres incrementally computes exact edge-costs only when necessary, utilizing the Lower-Bound Module in the process

Neural Bipartite Matching

- Message-passing Neural Networks and Principal Neighbourhood Aggregation were used as models for GNN
- 300 bigraphs were used for training and 50 for validation
- Both the models have exhibited 99.7% accuracy rate across all test sets

Paper Summary

Dynamic Load Balancing Algorithm Based on Optimal Matching of Weighted Bipartite Graph

- Aims to balance dynamic load across servers efficiently by using weighted bigraph model - servers & tasks are represented as vertices
- Each edge in graph is weighted based on task amount and each server load capacity
- Applies Kuhn-Munkres to find maximum weighted server-task matching

Practical Applications

Resource Allocation

- Bipartite matching is used in resource allocation problems, such as matching organ donors with recipients in transplant networks, matching students with schools or courses in education systems, or matching advertisers with slots on webpages in online advertising

Job Scheduling

- Bigraph models the relationship between available resources and the tasks or jobs that need to be completed
- Goal is to efficiently allocate resources to tasks in order to optimize certain objectives, such as minimizing makespan, maximizing throughput, or balancing workloads among resources.

Practical Applications

Stable Roommates Problem

- A group of individuals express their preferences for potential roommates, and the goal is to form stable pairs that satisfy everyone's preferences

Wireless Communication Networks

- Bigraphs are used to assign devices to frequencies of communications channels
- Bipartite matching algorithms assist in finding an assignment that minimizes interference between channels, maximizing the utilization of available frequencies
- This is particularly crucial in scenarios like cellular networks

Practical Applications

Job Recruitment

- match job seekers with job openings based on their preferences and qualifications.
- Particularly useful in online job portals and recruitment platforms, where algorithms can efficiently match candidates to suitable job listings, optimizing the hiring process for both employers and job seekers.

Relevant equations

Flow Conservation Constraints

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

Where $f(u,v)$ represents the flow from node u to node vv,and V is the set of nodes.

Capacity Constraints

For each edge (u,v), the flow $f(u,v)$ must satisfy:

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

Balance Constraints

$$x_{ij} = 1$$

Where xij is a binary variable indicating whether edge i is matched to edge j, and V is the set of vertices in the corresponding partite set

Ride Hailing/ Sharing

Ride Hailing/ Sharing

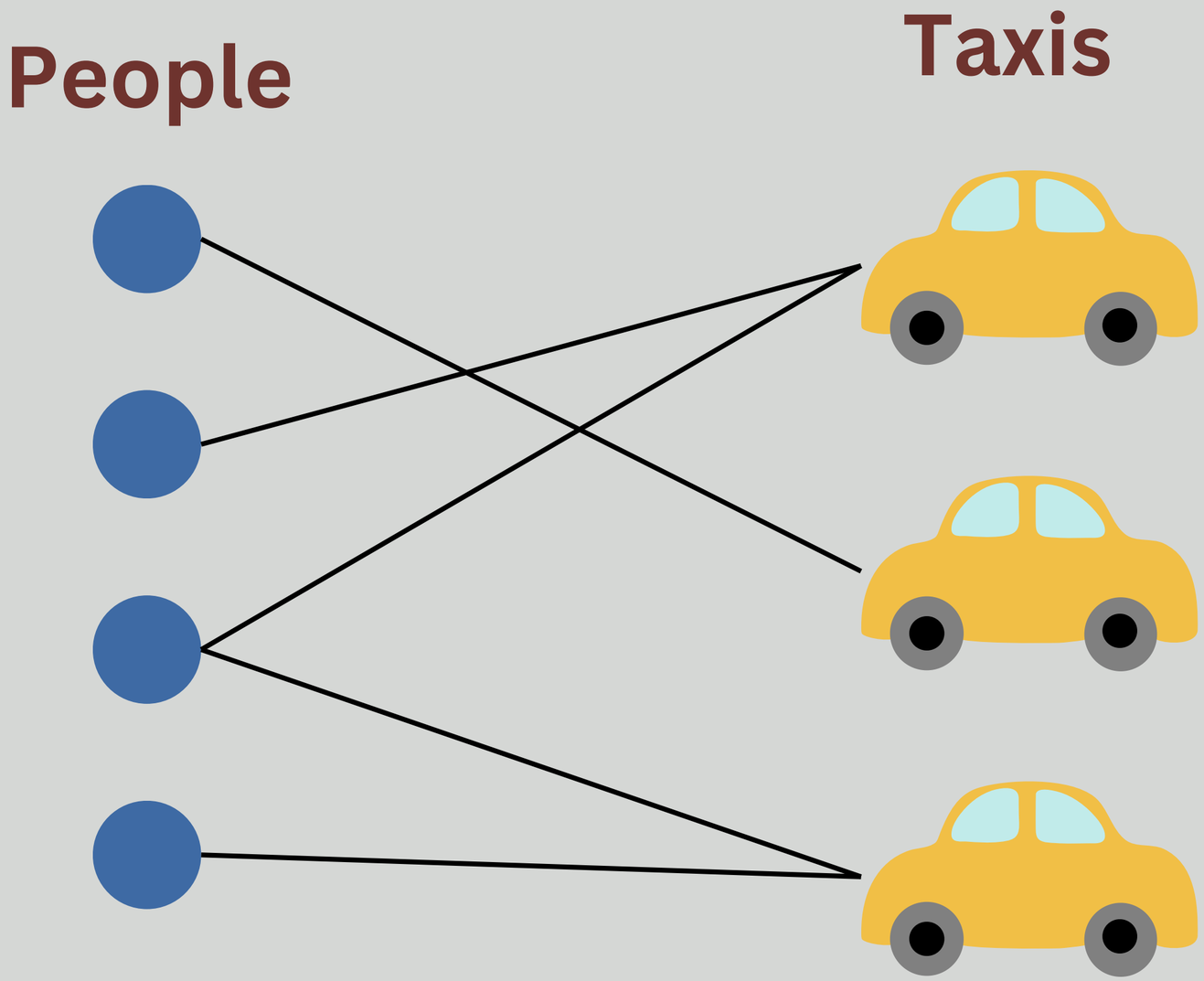
People

Ride Hailing/ Sharing

People

CR

Ride Hailing/ Sharing



Code implementation

```
int main() {

 vertex vertex_A, vertex_B;
 uint uid_A, uid_B;

uint size_matching = 0;
  for (uint i = 0; i < size_A; i++) {
    if (matched[i] == UNMATCHED) {
      fill(visited, visited + MAX_INSTANCE_SIZE,
false);
      if (augment_path(i)) {
        size_matching++;
      }
    }
  }

printf("Size of maximum matching: %d\n",
size_matching);
  for (uint i = 0; i < size_A; i++) {
    if (matched[i] != UNMATCHED) {
      vertex matched_A = uid_to_vertex(i);
      vertex matched_B = uid_to_vertex(matched[i]);

      printf("A%d B%d\n", matched_A.id, matched_B.id);
    }
  }

  return 0;
}
```

Code implementation

```
bool augment_path(uint uid) {
  visited[uid] = true;

  for (uint i = 0; i < graph[uid].size(); i++) {
    uint neighbour = graph[uid][i];
    if (visited[neighbour]) {
      continue;
    }

    if (matched[neighbour] == UNMATCHED) {
      matched[uid] = neighbour;
      matched[neighbour] = uid;
      return true;
    } else if (matched[neighbour] != uid) {
      visited[neighbour] = true;
      if (augment_path(matched[neighbour])) {
        matched[uid] = neighbour;
        matched[neighbour] = uid;
        return true;
      }
    }
  }

  return false;
}
```

Result & Time Complexity

Input

A0 B0
A1 B0
A2 B0
A2 B1
A3 B1
B2 A3
A4 B2
B1 A4
B3 A4
B4 A4

Output

A0 B0
A2 B1
A3 B2
A4 B3

Result & Time Complexity

Input

A0 B0
A1 B0
A2 B0
A2 B1
A3 B1
B2 A3
A4 B2
B1 A4
B3 A4
B4 A4

Output

A0 B0
A2 B1
A3 B2
A4 B3

Time complexity

O(V*E)

Comparative table

	Ford-Fulkerson	Kuhn-Munkres	Edmonds-Karp	Dinic's algo	MKM (Malhotra Kumar, Maheshwari algo)
Complexity	O(EU)	$O(V^3)$	$O(VE^2)$	$O(V^2E)$	$O(V^3)$
Advantages	Can be augmented with different methods for finding augmenting paths	Guaranteed to find the optimal solution	Guaranteed termination in polynomial time for integer capacities	Highly efficient and can handle large networks	Can handle large bipartite graphs
Disadvantages	Not guaranteed to terminate in polynomial time for all inputs	Not suitable with negative values	May not be efficient for most bipartite matching	Complex Implementation	limited to bipartite graphs. Not suitable for general flow networks

References

- Dynamic Load Balancing Algorithm Based on Optimal Matching of Weighted Bipartite Graph – https://ieeexplore.ieee.org/document/9970344
- Edge-Weighted Online Bipartite Matching – https://dl.acm.org/doi/full/10.1145/3556971
- Optimizing Bipartite Matching in Real-World Applications by Incremental Cost Computation – https://ieeexplore.ieee.org/document/10038818
- Neural Bipartite Matching – https://dl.acm.org/doi/abs/10.14778/3450980.3450983
- Load Balancing Algorithm Based on Weighted Bipartite Graph for Edge Computing – 5)https://arxiv.org/abs/2005.11304

THANK YOU