# STABLE FIXTURES PROBLEM – MANY TO MANY EXTENSION OF STABLE ROOMMATES PROBLEM

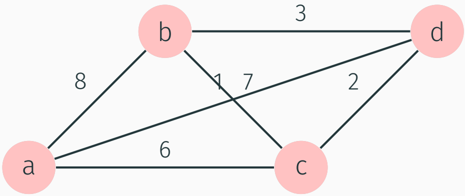
Saurabh Garg

July 1, 2016

Purdue University

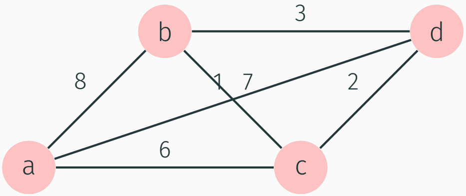
# IDEA

- 1–Matching :



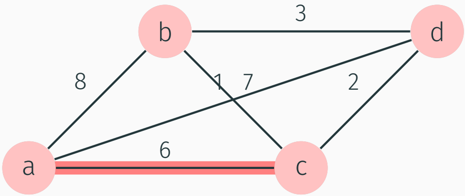
# IDEA

- 1–Matching :



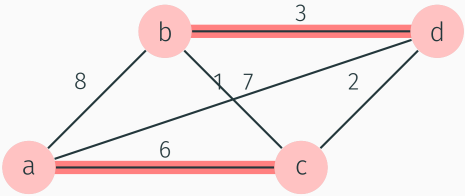
# IDEA

- 1–Matching :



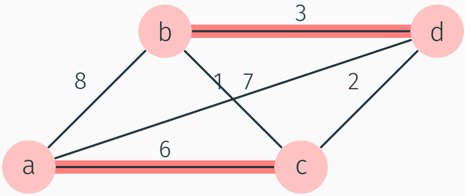
# IDEA

- 1–Matching :

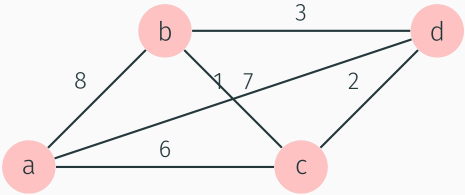


# IDEA

- 1–Matching :

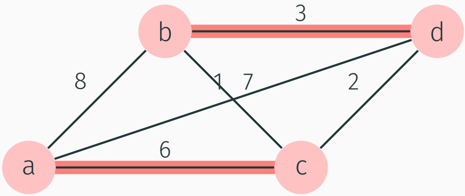



- Stable 1–Matching :

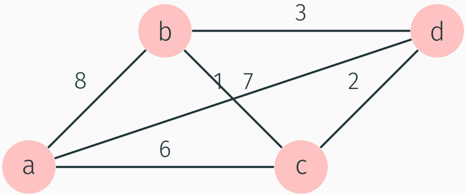


# IDEA

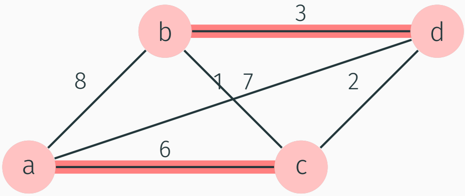
- 1–Matching :



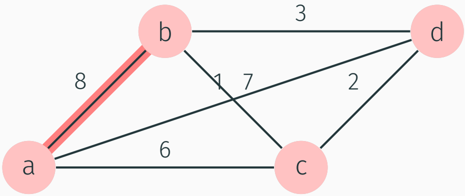

- Stable 1–Matching :



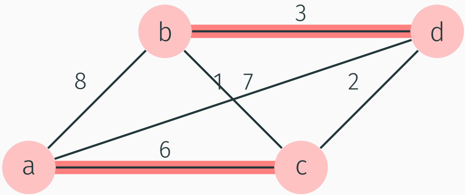
- 1–Matching :



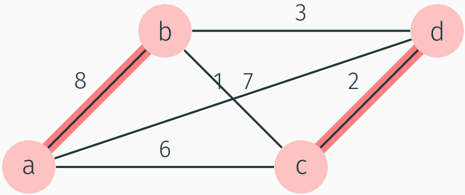

- Stable 1–Matching :



- 1–Matching :




- Stable 1–Matching :



# ABSTRACT

Stable fixtures problem is generalization of Stable Roommates problem in which each participant seeks to be matched with a number of others.

# FORMAL DEFINITION OF PROBLEM

- Given a general weighted directed graph G and an array b(v) of non negative values.

# FORMAL DEFINITION OF PROBLEM

- Given a general weighted directed graph G and an array b(v) of non negative values.
- The objective is to chose a subset of edges M such that at most b(v) edges in M are incident on each vertex v, and subject to this restriction we maximize the sum of the weights of the edges in M.

# ALGORITHM

# ALGORITHM

The Algorithm is in two phase :

- **Phase-1 :** Reduced the preference list by a sequence of bids and rejections

The Algorithm is in two phase :

- **Phase-1 :** Reduced the preference list by a sequence of bids and rejections
- **Phase-2 :** Removes cycle to conclude the existence of stable matching

**Idea :** This phase involves a sequence of proposals from each vertex v for another vertices's in the order of decreasing weights until they have made no less then b(v) proposals which were not rejected

---

**for all** i in N **do**
  **while** $|A_i| < \min(b_i, |P_i|)$ **do** ▷ $A_i$ is list of proposals made
    $x_j \leftarrow$ first player not in $A_i$
    $x_i$ bids for $x_j$ and $x_j$ becomes target for $x_i$
    **if** $|B_j| \geq b_j$ **then** ▷ $B_i$ is list of proposals received
      $x_k \leftarrow c_j$th bidder for $x_j$
      **for all** successors of $x_l$ of $x_k$ in $P_j$ **do**
        Remove all $x_l$ neighbours from P's, A's and B's(if any)
          ▷ This may lead to **atmost** one rejection by $x_j$
      **end for**
    **end if**
  **end while**
**end for**

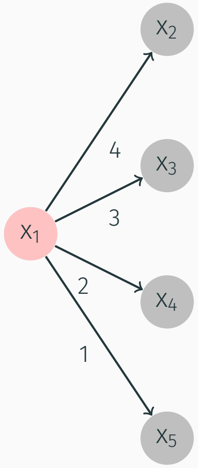
# EXAMPLE

$$
\begin{array}{lllll}
x_1\ (2): & x_2 & x_3 & x_4 & x_5 \\
x_2\ (1): & x_3 & x_4 & x_5 & x_1 \\
x_3\ (1): & x_4 & x_5 & x_1 & x_2 \\
x_4\ (1): & x_5 & x_1 & x_2 & x_3 \\
x_5\ (1): & x_1 & x_2 & x_3 & x_4
\end{array}
$$

Fig : Initial preference list

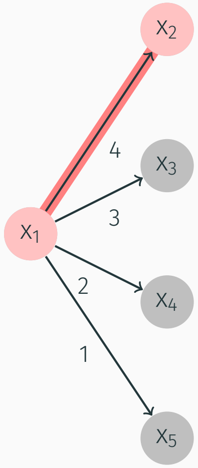
# ILLUSTRATION

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_3$ | $x_4$ | $x_5$ | $x_1$ |
| $x_3$ (1) : | $x_4$ | $x_5$ | $x_1$ | $x_2$ |
| $x_4$ (1) : | $x_5$ | $x_1$ | $x_2$ | $x_3$ |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | $x_4$ |



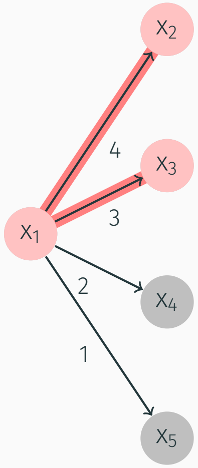
# ILLUSTRATION

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_3$ | $x_4$ | $x_5$ | $x_1$ |
| $x_3$ (1) : | $x_4$ | $x_5$ | $x_1$ | $x_2$ |
| $x_4$ (1) : | $x_5$ | $x_1$ | $x_2$ | $x_3$ |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | $x_4$ |



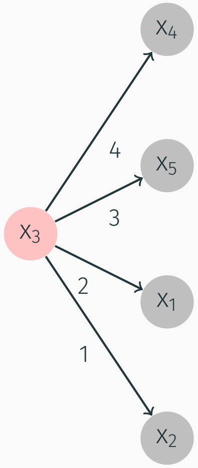
# ILLUSTRATION

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_3$ | $x_4$ | $x_5$ | $x_1$ |
| $x_3$ (1) : | $x_4$ | $x_5$ | $x_1$ | $x_2$ |
| $x_4$ (1) : | $x_5$ | $x_1$ | $x_2$ | $x_3$ |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | $x_4$ |



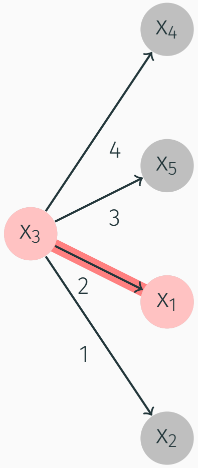
# ILLUSTRATION

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_3$ | $x_4$ | $x_5$ | $x_1$ |
| $x_3$ (1) : | $x_4$ | $x_5$ | $x_1$ | $x_2$ |
| $x_4$ (1) : | $x_5$ | $x_1$ | $x_2$ | $x_3$ |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | $x_4$ |



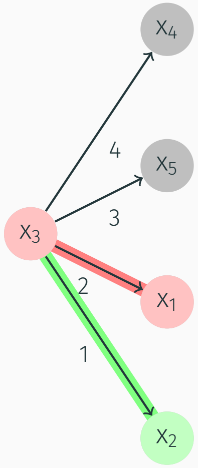
# ILLUSTRATION

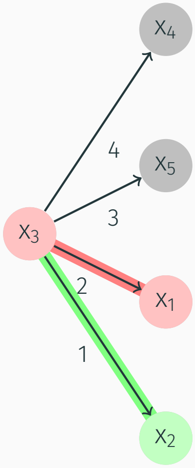
| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_3$ | $x_4$ | $x_5$ | $x_1$ |
| $x_3$ (1) : | $x_4$ | $x_5$ | $x_1$ | $x_2$ |
| $x_4$ (1) : | $x_5$ | $x_1$ | $x_2$ | $x_3$ |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | $x_4$ |

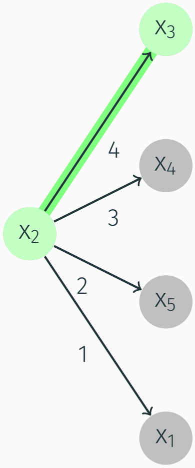


# ILLUSTRATION

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_3$ | $x_4$ | $x_5$ | $x_1$ |
| $x_3$ (1) : | $x_4$ | $x_5$ | $x_1$ | $x_2$ |
| $x_4$ (1) : | $x_5$ | $x_1$ | $x_2$ | $x_3$ |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | $x_4$ |




X4
X5
X3
X1
X2
4
3
2
1


X3
X4
X2
X5
X1
4
3
2
1

# EXAMPLE

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_4$ | $x_5$ | $x_1$ | |
| $x_3$ (1) : | $x_5$ | $x_1$ | | |
| $x_4$ (1) : | $x_1$ | $x_2$ | | |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | |

Fig :Reduced Preference list after Phase-1

Uses the above reduced Graph

**Idea :** This phase search for possible cycles and removes them. This phase terminates when no list is long* or atleast one list is short**.

*long : if $|p_i| > \min(b_i, P_i)$ **short if $|p_i| < \min(b_i, P_i)$

This Phase primarily comprises of two steps :

- (i) Cycle Detection
- (ii) Cycle removal

```
function DETECT_CYCLE
    Cycle = ∅
    while Until any x_i repeats do
        x_ik = last in x_jk's list who was not proposed (worst bidder)
        x_j(k+1) = first in x_ik's list who was not proposed (next target)
    end while
    return Cycle                    ▷ Cycle is = ((x_i0, x_j0), ....(x_ik, x_jk)...)
end function
while (there is no short list and some long list) do
    ρ =DETECT_CYCLE()
    Remove cycle from the graph
end while
if some list is short then
    No stable matching
else
    Stable Matching exist and reduced G is itself the answer
end if
```

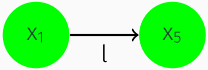
# EXAMPLE

f: next target

l :worst bidder

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_4$ | $x_5$ | $x_1$ | |
| $x_3$ (1) : | $x_5$ | $x_1$ | | |
| $x_4$ (1) : | $x_1$ | $x_2$ | | |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | |

Fig :Reduced Preference list after Phase-1



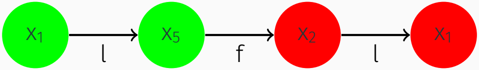
# EXAMPLE

f: next target

l :worst bidder

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_4$ | $x_5$ | $x_1$ | |
| $x_3$ (1) : | $x_5$ | $x_1$ | | |
| $x_4$ (1) : | $x_1$ | $x_2$ | | |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | |

Fig :Reduced Preference list after Phase-1

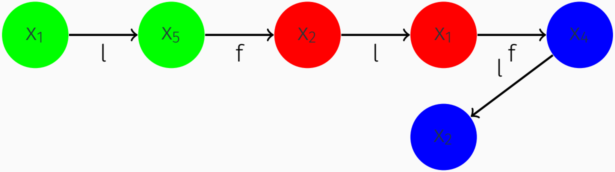


f: next target

l :worst bidder

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_4$ | $x_5$ | $x_1$ | |
| $x_3$ (1) : | $x_5$ | $x_1$ | | |
| $x_4$ (1) : | $x_1$ | $x_2$ | | |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | |

Fig :Reduced Preference list after Phase-1



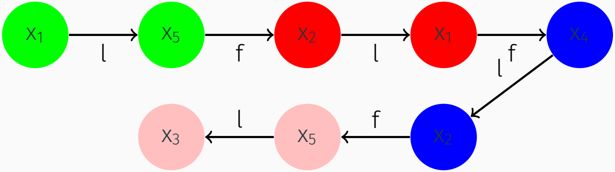
# EXAMPLE

f: next target

l :worst bidder

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_4$ | $x_5$ | $x_1$ | |
| $x_3$ (1) : | $x_5$ | $x_1$ | | |
| $x_4$ (1) : | $x_1$ | $x_2$ | | |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | |

Fig :Reduced Preference list after Phase-1



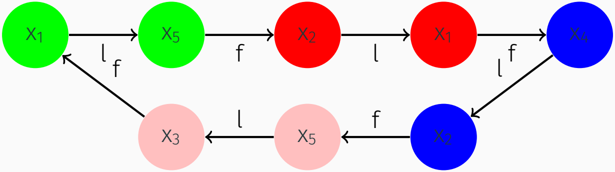
# EXAMPLE

f: next target

l :worst bidder

| | | | | |
|---|---|---|---|---|
| $x_1$ (2) : | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| $x_2$ (1) : | $x_4$ | $x_5$ | $x_1$ | |
| $x_3$ (1) : | $x_5$ | $x_1$ | | |
| $x_4$ (1) : | $x_1$ | $x_2$ | | |
| $x_5$ (1) : | $x_1$ | $x_2$ | $x_3$ | |

Fig :Reduced Preference list after Phase-1



| | | |
|---|---|---|
| $x_1$ (2) : | $x_3$ | $x_4$ |
| $x_2$ (1) : | $x_5$ | |
| $x_3$ (1) : | $x_1$ | |
| $x_4$ (1) : | $x_1$ | |
| $x_5$ (1) : | $x_2$ | |

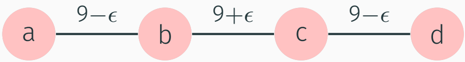
Fig : Preference list after Phase-2

# COMPARISON

The solution generated by this algorithm is always stable whereas there is possibility that optimal solution is not stable.

**For example :**

- (i) Optimal Solution

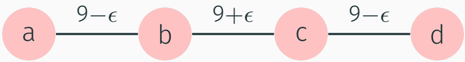


## RELATIONSHIP WITH OPTIMAL SOLUTION

The solution generated by this algorithm is always stable whereas there is possibility that optimal solution is not stable.

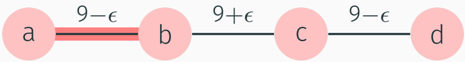
**For example :**

- (i) Optimal Solution



The solution generated by this algorithm is always stable whereas there is possibility that optimal solution is not stable.
**For example :**

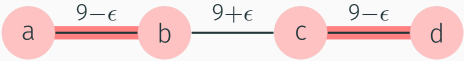
- (i) Optimal Solution



The solution generated by this algorithm is always stable whereas there is possibility that optimal solution is not stable.

**For example :**

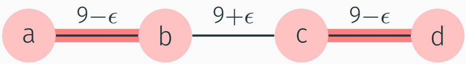
- **(i) Optimal Solution**



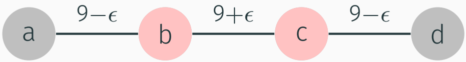
The solution generated by this algorithm is always stable whereas there is possibility that optimal solution is not stable.

**For example :**

- (i) Optimal Solution



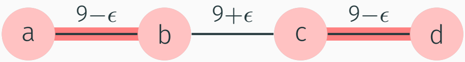

- (ii) Stable Solution



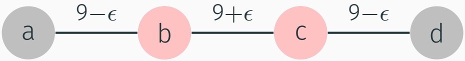
The solution generated by this algorithm is always stable whereas there is possibility that optimal solution is not stable.

**For example :**

- (i) Optimal Solution



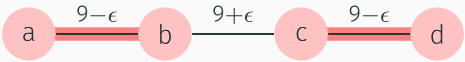

- (ii) Stable Solution



The solution generated by this algorithm is always stable whereas there is possibility that optimal solution is not stable.

**For example :**

- (i) Optimal Solution




- (ii) Stable Solution

- **B-Suitor** algorithms generates a half approximation solution for general undirected graph. The solution generated as mentioned above is also stable.

- B-Suitor algorithms generates a half approximation solution for general undirected graph. The solution generated as mentioned above is also stable.
- For general undirected graphs phase–1 of the algorithm is sufficient because preference list is symmetric.

# B-SUITOR

- **B-Suitor** algorithms generates a half approximation solution for general undirected graph. The solution generated as mentioned above is also stable.
- For general undirected graphs phase–1 of the algorithm is sufficient because preference list is symmetric.
- But for general directed graphs we need phase–2 after phase–1 to symmetrize the graph so that resulted graph is stable and each edge has at most b(v) edges going out of them.

- Thus Phase–1 of this algorithm resembles with B–Suitor and is sufficient for weighted undirected graph.

- Thus **Phase–1** of this algorithm resembles with **B–Suitor** and is sufficient for weighted undirected graph.
- But we need phase–2 in case of **directed weighted graphs**.

# IMPLEMENTATION

- **Preprocessing :** Graph we get as input is unsymmertric weighted graph. In this phase all unsymmetric edges are removed and graph is sorted in order of decreasing weights. Complexity is O(mlog(n))

# DETAILS

- **Preprocessing :** Graph we get as input is unsymmertric weighted graph. In this phase all unsymmetric edges are removed and graph is sorted in order of decreasing weights. Complexity is O(mlog(n))
- **Phase-1 :** In the current implementation graph is stored as vector of maps and in this phase by a sequence of bids and rejections graph is reduced. Complexity is O(mlog(n)).

- **Preprocessing :** Graph we get as input is unsymmertric weighted graph. In this phase all unsymmetric edges are removed and graph is sorted in order of decreasing weights. Complexity is O(mlog(n))
- **Phase-1 :** In the current implementation graph is stored as vector of maps and in this phase by a sequence of bids and rejections graph is reduced. Complexity is O(mlog(n)).
- **Phase-2 :** In the current implementation using sets of bidders and proposers cycles are identified and removed until there are no more cycles which mean graph becomes symmetric or atleast one list becomes short. Complexity is O(m).

# ANALYSIS

| Num. of Nodes | Num. of edges | Time for Phase−1 | No. of cycles | Time for Phase−2 | Output |
|---|---|---|---|---|---|
| 10 (2) | 62 | 0.0001 | 2 | 0.00001 | Exist |

**Table:** Analysis for various sparse graphs

# ANALYSIS

| Num. of Nodes | Num. of edges | Time for Phase−1 | No. of cycles | Time for Phase−2 | Output |
|---|---|---|---|---|---|
| 10 (2) | 62 | 0.0001 | 2 | 0.00001 | Exist |
| 735323 (10) | 5158388 | 3.66433 | 0 | 0 | Exist |

Table: Analysis for various sparse graphs

# ANALYSIS

| Num. of Nodes | Num. of edges | Time for Phase−1 | No. of cycles | Time for Phase−2 | Output |
|---|---|---|---|---|---|
| 10 (2) | 62 | 0.0001 | 2 | 0.00001 | Exist |
| 735323 (10) | 5158388 | 3.66433 | 0 | 0 | Exist |
| 916428 (2) | 5105039 | 3.0453 | 9 | 0.174925 | Does not exist |

Table: Analysis for various sparse graphs

# ANALYSIS

| Num. of Nodes | Num. of edges | Time for Phase−1 | No. of cycles | Time for Phase−2 | Output |
|---|---|---|---|---|---|
| 10 (2) | 62 | 0.0001 | 2 | 0.00001 | Exist |
| 735323 (10) | 5158388 | 3.66433 | 0 | 0 | Exist |
| 916428 (2) | 5105039 | 3.0453 | 9 | 0.174925 | Does not exist |
| 916428 (5) | 5105039 | 2.6884 | 2 | 0.0571 | Exist |

**Table:** Analysis for various sparse graphs

# ANALYSIS

| Num. of Nodes | Num. of edges | Time for Phase−1 | No. of cycles | Time for Phase−2 | Output |
|---|---|---|---|---|---|
| 10 (2) | 62 | 0.0001 | 2 | 0.00001 | Exist |
| 735323 (10) | 5158388 | 3.66433 | 0 | 0 | Exist |
| 916428 (2) | 5105039 | 3.0453 | 9 | 0.174925 | Does not exist |
| 916428 (5) | 5105039 | 2.6884 | 2 | 0.0571 | Exist |
| 1382908 (20) | 16539643 | 7.72464 | 186 | 3.76724 | Does not exist |

**Table:** Analysis for various sparse graphs

# ANALYSIS

| Num. of Nodes | Num. of edges | Time for Phase−1 | No. of cycles | Time for Phase−2 | Output |
|---|---|---|---|---|---|
| 10 (2) | 62 | 0.0001 | 2 | 0.00001 | Exist |
| 735323 (10) | 5158388 | 3.66433 | 0 | 0 | Exist |
| 916428 (2) | 5105039 | 3.0453 | 9 | 0.174925 | Does not exist |
| 916428 (5) | 5105039 | 2.6884 | 2 | 0.0571 | Exist |
| 1382908 (20) | 16539643 | 7.72464 | 186 | 3.76724 | Does not exist |
| 2394385 (10) | 5021410 | 1.7377 | 0 | 0 | Exist |

**Table:** Analysis for various sparse graphs

| Num. of Nodes | Num. of edges | Time for Phase−1 | No. of cycles | Time for Phase−2 | Output |
|---|---|---|---|---|---|
| 10 (2) | 62 | 0.0001 | 2 | 0.00001 | Exist |
| 735323 (10) | 5158388 | 3.66433 | 0 | 0 | Exist |
| 916428 (2) | 5105039 | 3.0453 | 9 | 0.174925 | Does not exist |
| 916428 (5) | 5105039 | 2.6884 | 2 | 0.0571 | Exist |
| 1382908 (20) | 16539643 | 7.72464 | 186 | 3.76724 | Does not exist |
| 2394385 (10) | 5021410 | 1.7377 | 0 | 0 | Exist |

**Table:** Analysis for various sparse graphs

# Questions?