# The Gale-Shapely Algorithm and Stable Pairings

Foqia Shahid and Rafael Jovel

May 22, 2021

## 1 What is the Stable-Marriage Problem?

The Stable-Marriage problem as set out by David Gale and Lloyd Shapley is an assignment problem that takes into account the preferences of two groups to be paired with member of the other.

The end goal of the problem is to create a stable pairing.

**Stable Pairing:** A stable pairing is one in which there does not exist a single pair in which both members prefer each other to their current partner under the matching.

The original definition of this problem related to a group of $n$ men proposing to $n$ women for their hand in marriage, hence the name the Stable-Marriage Problem. $n$ is specifically a whole number larger than 2 to avoid a trivial problem. In such a situation, every potential groom and every potential bride had their preferences for the members of the other group, ranked comparatively to the members of the other group. This, however, is not the only variation of this problem that could make use of Stable Pairings.

## 2 What are some common variations of this problem?

Some other variations of this problem are as follows:

1. **Stable Roommate Problem** In this scenario, two groups of students must find a roommate from the other group and thus rank their preferences based on things like living habits, sleep schedules, and common interests to find their most ideal roommate for the semester.
2. **Hospital/Resident Pairing:** In this situation there are a set of students coming out of medical school and moving into their residency and various hospitals ready to accept their new class of residents with both groups having their own set of preferences based on openings in departments for the hospitals and based on preferred practice for the medical school students.
3. **Lab Partner Pairing:** In this scenario two groups of students have to choose a partner from the other group, accounting for scheduling conflicts, difference in skill sets, and in the case of students studying remotely, time zones! Thus, each group ranks their preferences for the other group to create a stable pairing.

## 3 The Gale-Shapley Algorithm

The Gale-Shapley Algorithm runs in the following way.

First, the first person in the proposer group proposes to their top choice. That top choice then accepts them as they have yet to receive any other offer to that point.

Then, the second person in the proposer group proposes to their top choice leading to three potential options:

1. The proposed woman is not engaged so she accepts
2. The proposed woman is engaged but she prefers this offer to her current partner so she accepts. In this casee, the previous partner is no longer engaged and returns to the proposer group.
3. The proposed woman is engaged but she doesnt prefer this offer to her current partner so she rejects. In this case, the proposer will propose to their next top choice, repeating this cycle from step 1.

This cycle is repeated for each proposer until everyone is engaged. The pairings at the end will be stable according to Theorem 1.

**Theorem 1:** The Gale-Shapley algorithm results in all stable pairings.

*Proof.* **By Contradiction:** Suppose there exists an unstable pair i.e a pair (m, w) such that both members prefers each other to their current partners under the matching. However, if m prefers w to his current partner, he must have already proposed to her. It follows that he must have been rejected by w who prefers someone else. □

It is important that a stable matching always exists because otherwise this problem would serve no empirical purpose. An assignment that does not adequately address the preferences of the two groups would make the problem moot from its core premise.

**Theorem 2:** The Gale-Shapley algorithm provides the best possible pairings for the proposing groups

*Proof.* To prove this, we will first show that the order of proposers does not matter.

Suppose the proposers go in order of $m_1, m_2, m_3, \ldots$ i.e $m_1$ proposes before $m_2$. Within a given cycle, each proposer proposes to their top choice even if the woman is already engaged. Thus, within a cycle, the ordering of proposers doesn't matter since everyone gets to propose to their top choice.

Furthermore, since each proposer proposes to their top choice and only proposes again if they are rejected, this means that each proposer gets their best possible pairing provided the competition from the other proposers.

Since order does not matter and each proposer gets to proposer in order to their top choices, the algorithm is optimized to give them as as high as preference as possible based on the preferences of the acceptor group.

□

**Corollary:** The algorithm does not provide the best possible pairings for the accepting group

*Proof.* The algorithm provides the best possible pairing for the proposing group by not letting the acceptor group make offers i.e. the women only accept offers and do not make any themselves. This implies that each man is matched to his best possible choice and each woman is matched to her best possible offer (as the woman may not get her top choice/s as offers). Thus, the algorithm doesn't provide the best possible pairing for the accepting (womens) group.

If the roles were switched and the women were the proposing group, then the algorithm would give the best possible pairing for the women.

□

**Note: At most the Gale-Shapley Algorithm will finish in $n^2$ moves as at most a single proposer can be rejected $n-1$ times before they are necessarily paired with someone in the acceptor group.**

## 4 Python Implementation

```python
def gale_shapley(N, men,mpreferences, women, wpreferences):
  pairs =list(range(N))
  mFree = []
  wFree = []
  Proposed = []
  for i in range(N): #Initialize all men and women to be free
    mFree.append(True)
    wFree.append(True)
    Proposed.append(0)

  freeM_index = 0;
  w_index = 0
  #While there is a free man m who still has a women w to propose to
  while mFree[freeM_index] == True and Proposed[freeM_index] < N:
    prop = nProposed(Proposed, freeM_index)
    #w:= first women  on m's list NOT YET proposed to
    w = mpreferences[freeM_index][prop]
    w_index = findindex_wFree(women,w)
    if wFree[w_index] == True: #if w is free
      pairs[freeM_index]= w #then (m,w) become engaged
      mFree[freeM_index] = False
      wFree[w_index] = False
      Proposed[freeM_index] +=1
    else: #some pair (m',w) already exists
      m_row = []
      m_row = wpreferences[w_index]
      m = findm(men, freeM_index)
      m_current = m_row.index(m)
      m_other_inpairs = pairs.index(w)
      m_ = findm(men, m_other_inpairs)
      m_other = m_row.index(m_)

      if wPrefers(m_current, m_other) == True:#w prefers m (this offer) to m'(other
       offer)
        index_in_pairs_m_other = findindex_mFree(men, m_)
        mFree[index_in_pairs_m_other] = True
        pairs[freeM_index]= w #(m, w) become engaged
        mFree[freeM_index] = False
        wFree[w_index] = False
        Proposed[freeM_index] +=1
      else:
        Proposed[freeM_index] +=1 #(m',w) remain engaged
        continue
    freeM_index = (freeM_index + 1)% N;
    if(mFree[freeM_index] == False):
      freeM_index = (freeM_index + 1)% N;
  print(pairs)


#Below are some helper methods
#Gives the number of proposals for man at Proposed[index]
def nProposed(Proposed, index):
  return Proposed[index]


#Determines index of w in the wFree list
def findindex_wFree(women, w):
  return women.index(w)


#Determines index of m in the mFree list
def findindex_mFree(men, m):
  return men.index(m)

#Checks to see if the woman w prefers
#the current offer to the previously accepted offer
#m_current = index of m_current in wpreferences
#m_other = index of m_other in wpreferences

def wPrefers(m_current, m_other):
  if m_current < m_other: #m_current is prefers over m_other
    return True
  else:
    return False

#Tells us who m is given the index
def findm(men, freeM_index):
  return men[freeM_index]
```

**Note: This code and the line by line comments to explain it are also in the notebook!**

## 5 Example Problems

We have two example problems within our documented python code! The first of which is a generic problem and the second of which is a more clear example of the stable marriage problem.

**Ex. 1** In this example we have a generic case where the proposer and acceptor groups are of sizes 4. The proposer group has the members $E, F, G$, and $H$ while the acceptor group has members $A, B, C$, and $D$.

Their preferences are as follows:

**Proposer Group**

|  |  |  |  |  |
|---|---|---|---|---|
| **E** | $C$ | $B$ | $A$ | $D$ |
| **F** | $B$ | $C$ | $A$ | $D$ |
| **G** | $C$ | $A$ | $B$ | $D$ |
| **H** | $D$ | $A$ | $C$ | $B$ |

**Acceptor Group**

|  |  |  |  |  |
|---|---|---|---|---|
| **A** | $F$ | $E$ | $H$ | $G$ |
| **B** | $E$ | $G$ | $F$ | $H$ |
| **C** | $G$ | $H$ | $F$ | $E$ |
| **D** | $E$ | $F$ | $G$ | $H$ |

Based on these preferences the following should happen:

(a) $E$ proposes to $C$ and is accepted.

(b) $F$ proposes to $B$ and is accepted.

(c) $G$ proposes to $C$ and is accepted sending $E$ back into the proposal group as $C$ most prefers $G$.

(d) Now back in the proposer group, $E$ proposes to $B$ and is accepted over $F$ because of $B$'s preferences. This sends $F$ back to the proposal group.

(e) $F$ then proposes to $C$ and is rejected as $C$ still prefers $G$ as they had $G$ as their first choice.

(f) Then, $F$ proposes to $A$ and is accepted as $A$ has not yet been proposed to.

(g) $H$ proposes to $D$ and is accepted.

At that point the cycle is over and the stable pairings are reached which them being $(E, B), (F, A), (G, C), (H, D)$. This result is shown in our documented code, showing us that the case of $n = 4$ works as intended.

**Ex. 2** In this second example we are applying the python code to a case that deals with names in a situation that is exactly to the Stable-Marriage problem, with men proposing to women, with the groups being of size $n = 5$ here.

The proposer group has the members of Evan, Ali, Zach, Cody, and Brian. The acceptor group has the members of Sara, Zoe, Maddy, Eli, May.

The preferences of the two groups are as follows:

**Proposer Group**

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| **Evan** | Sara | Zoe | Maddy | Eli | May |
| **Ali** | Maddy | May | Zoe | Sara | Eli |
| **Zach** | Maddy | Sara | Eli | May | Zoe |
| **Cody** | May | Zoe | Sara | Eli | Maddy |
| **Brian** | May | Sara | Eli | Zoe | Maddy |

**Acceptor Group**

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| **Sara** | Evan | Brian | Cody | Zach | Ali |
| **Zoe** | Brian | Ali | Evan | Cody | Zach |
| **Maddy** | Evan | Zach | Ali | Brian | Cody |
| **Eli** | Zach | Ali | Brian | Cody | Evan |
| **May** | Ali | Cody | Evan | Zach | Brian |

Based on these preferences the following should happen:

1. Evan proposes to Sara and is accepted.
2. Ali proposes to Maddy and is accepted.
3. Zach proposes to Maddy and is accepted sending Ali to propose again.
4. Ali proposes to May and is accepted.
5. Cody proposes to May and is rejected as May prefers Ali.
6. Cody proposes to Zoe and is accepted.
7. Brian proposes to May and is rejected as May prefers Ali.
8. Brian proposes to Sara and is rejected again as Sara prefers Evan.
9. Brian proposes to Eli and is accepted.

This leaves us with the following pairings:

(Evan,Sara),(Ali, May),(Zach,Maddy),(Cody,Zoe),(Brian,Eli)

This same result is shown in our documented code showing that the implementation of the algorithm also more generally works here!

## 6 Bibliography

1. https://www.youtube.com/watch?v=0m_YW1zVs-Q
2. https://en.wikipedia.org/wiki/Gale%E2%80%93Shapley_algorithm
3. https://www.overleaf.com/learn/latex/Code_listing