Gale–Shapley (Deferred Acceptance) Algorithm: A Simple Implementation Guide

Francesco Balocco

February 17, 2026

What problem does Gale–Shapley solve?

You have two equal-sized sets of agents (e.g., **men** and **women**). Each agent has a **ranked list** of the agents on the other side (a strict ordering, from best to worst).

Goal. Find a **one-to-one matching** such that there is **no blocking pair**.

Blocking pair (key concept). A man m and a woman w form a **blocking pair** if:

- m prefers w to his assigned partner, *and*
- w prefers m to her assigned partner.

If such a pair exists, the matching is **not stable** because m and w would both want to deviate.

What do you need as input (for coding)?

You need two preference dictionaries:

- `proposer_prefs[p]`: a list of acceptors in descending preference.
- `acceptor_prefs[a]`: a list of proposers in descending preference.

Example (shape).

```
proposer_prefs = {
  'W01': ['M04','M01','M10', ...],
  'W02': ['M10','M06','M01', ...],
  ...
}
acceptor_prefs = {
  'M01': ['W07','W04','W01', ...],
  'M02': ['W06','W02','W08', ...],
  ...
}
```

How do you get preference lists from data? (the practical step)

In applications (including the speed-dating exercise), you often start from a **score** for each potential pair:

- a predicted probability (e.g., $\hat{p} = P(\texttt{decision} = 1)$), or
- any other numeric "utility" score.

To run Gale–Shapley, you only need the **rank order** of those scores:

- For each woman w: sort all men m by $\hat{p}_{w,m}$ from highest to lowest. That is `proposer_prefs[w]` if women propose.
- For each man m: sort all women w by $\hat{p}_{m,w}$ from highest to lowest. That is `acceptor_prefs[m]` in the same run.

Important detail: ties. If two scores are exactly equal, break ties deterministically (e.g., by the partner ID). This produces a **strict** ranking, which is what the standard algorithm expects.

The algorithm (high level intuition)

Gale–Shapley is also called **deferred acceptance**.

- The proposing side proposes to their favorite acceptable option *they have not proposed to yet*.
- The receiving side *tentatively* keeps the best proposal so far (according to their own ranking) and rejects the rest.
- Rejected proposers try again next round with their next best choice.

It stops when no proposer wants (or is able) to make a new proposal.

The algorithm (exact steps you can implement)

Assume "proposers" propose to "acceptors".

Data structures (recommended)

- `free`: queue/list of currently free proposers.
- `next_idx[p]`: which acceptor index proposer p will propose to next.
- `engaged[a]`: the proposer currently held by acceptor a (a tentative engagement).
- `acceptor_rank[a][p]`: a fast lookup of how acceptor a ranks proposer p (smaller = better).

Pseudocode

```
build acceptor_rank[a][p] for all acceptors a and proposers p
free = all proposers
next_idx[p] = 0 for all proposers
engaged = {}   # acceptor -> proposer

while free is not empty:
  p = pop one proposer from free
  if next_idx[p] == len(proposer_prefs[p]):
    continue  # p has proposed to everyone

  a = proposer_prefs[p][ next_idx[p] ]
  next_idx[p] += 1

  if a not in engaged:
    engaged[a] = p
  else:
    current = engaged[a]
    if acceptor_rank[a][p] < acceptor_rank[a][current]:
      engaged[a] = p
      push current back into free
    else:
      push p back into free

return match where each proposer p is matched to the acceptor that holds them
```

How to switch "women propose" vs. "men propose"

Nothing in the code changes except which side you pass in as the proposer dictionaries:

- **Women propose**: proposers = women, acceptors = men.
- **Men propose**: proposers = men, acceptors = women.

Important output property (one sentence). The proposing side gets its **best stable outcome** (women-optimal or men-optimal stable matching).

A tiny example (so you can sanity-check your code)

Women propose. Each woman lists men from best to worst:

```
W1: M1 > M2
W2: M1 > M2
M1: W2 > W1
M2: W1 > W2
```

Round 1: W1→M1, W2→M1. M1 keeps W2 and rejects W1.
Round 2: W1→M2. M2 accepts.
Result: (W2,M1) and (W1,M2). This is stable, and it is best (among stable matchings) for women because women proposed.

A minimal Python implementation (matches the notebook style)

```python
def gale_shapley(proposer_prefs, acceptor_prefs):
    acceptor_rank = {
        a: {p: r for r, p in enumerate(lst)}
        for a, lst in acceptor_prefs.items()
    }
    free = list(proposer_prefs.keys())
    next_idx = {p: 0 for p in proposer_prefs}
    engaged = {}  # acceptor -> proposer

    while free:
        p = free.pop(0)
        if next_idx[p] >= len(proposer_prefs[p]):
            continue
        a = proposer_prefs[p][next_idx[p]]
        next_idx[p] += 1

        if a not in engaged:
            engaged[a] = p
            continue

        current = engaged[a]
        if acceptor_rank[a].get(p, 10**9) < acceptor_rank[a].get(current, 10**9):
            engaged[a] = p
            free.append(current)
        else:
            free.append(p)

    match = {p: None for p in proposer_prefs}
    for a, p in engaged.items():
        match[p] = a
    return match
```

Common implementation pitfalls (and how to avoid them)

- **Slow ranking checks**: do not use `list.index()` inside the main loop for acceptors; precompute `acceptor_rank`.
- **Incomplete lists**: the standard proof assumes complete strict rankings. If you drop some partners, decide what "unacceptable" means (usually: they are ranked last or not included at all).
- **Ties**: if two partners have the same score, you must break ties deterministically (e.g., by ID) to get a strict list.
- **Unequal sizes**: the algorithm still runs, but some proposers may end up unmatched (`None`).
- **Interpreting "utility"**: Gale–Shapley uses only *rank order*, not cardinal values. Any monotone transformation of your scores yields the same preferences.

How to test your code

1. Start with a tiny toy example (2–3 agents per side) where you can compute the outcome by hand.
2. Verify **stability**: check whether any blocking pair exists.
3. Run both regimes (women propose / men propose) and compare outcomes.