Tianyun Duan

Td22438

Part 1

(a) By way of contradiction, suppose false. Then there is a hard-worker H that gets a part time job Jp (which implies that a lazy worker L gets a full time job Jf). Since H now has a part time job, he must have proposed to all full-time jobs already, including Jf.
Case 1: H proposed to Jf before L did, then either Jf accepted H or Jf at that point has a more preferable employee. Either way, it is impossible for Jf to accept L later on. Therefore this is a contradiction.
Case 2: H proposed to Jf after L does, then later in time Jf will definitely accept H(or someone who is more preferable than L) and abandon L, and L would not have had the job Jf when the algorithm terminates.
Thus, proved.

(b) O(n!) * time complexity of isStable()
The first man has n possible choices; the second man has n-1 possible choices; the third man has n-2 …… Therefore, there will be n * (n-1) * (n-2) * … * 1 = n! possibilities. For each possibility, we also need to run isStable() to check if it's an acceptable solution, so we also need to multiply O(n^2) by its time complexity, which may vary depending on different implementations.

(c) While there is a jobless worker W:

```
   Look at the first Job, J, on W's preference list
   If J is not taken:
      pair (W, J)
   Else if J is taken:
      If J prefers W over its current worker W':
         Pair (W, J)
         W' becomes jobless
      Else:
         W remains jobless
```

(d) Proof:
Suppose my algorithm doesn't work, and there is a pair (W, J) where W and J prefer each other over their current job/worker. Let W's current job be J' and J's current worker be W'.

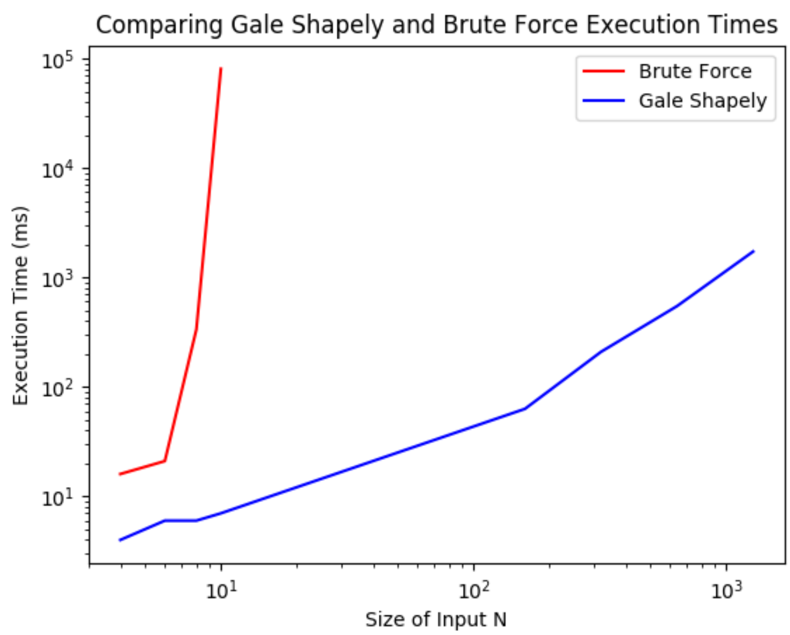
Case 1: W' gets the job J before W proposes to job J.
Then when W proposes, J would abandon W' for W for sure, and W' would not be holding the job J when the algorithm terminates.

Case 2: W' gets the job J after W proposes to job J.
By the nature of the algorithm, J's partner will only be more and more preferable as the algorithm executes. Therefore, there is no way J declined W and later accepted W'.

So the instability is impossible in both cases, therefore it's a contradiction. Thus proved.

(e) It is at most O(n^2), because in the worst case, n workers would need to propose n times.

(f)



Part 4:

(a) No. Counter example:
Two workers and two jobs.
W1 prefers J1 over J2
W2 prefers J1 over J2
J1 is indifferent between W1 and W2
J2 is indifferent between W1 and W2

Then no matter what the matching is, one of the worker is going to be matched with J2, although he prefers J1. But J1 is indifferent between him and the other worker, so the matching is unstable by definition.

(b) Yes.
The proof is the same as Part 1 (d):
Suppose not true, then there is a pair (w, j) who prefers each other over their current partners j' and w'.

Case 1: w proposed to j before w' did. By the nature of the algorithm, j's partner will only get more preferable, so there is no way j declines w and accepts w', a contradiction.
Case2: w proposed to j after w' did. Then w will at some point propose to j, and j will for sure abandon w' for w, since w is more preferable. But it did not, so there is a contradiction.
Thus proved.

(c) Since there is not always a matching without any weak instability, I define stable as not having any strong instability.

My previous algorithm still works, because it prevents any sort of strong instability.

While there is a jobless worker W:
  Look at the first Job, J, on W's preference list
  If J is not taken:
    pair (W, J)
  Else if J is taken:
    If J **strictly** prefers W over its current worker W':
      Pair (W, J)
      W' becomes jobless
    Else:
      W remains jobless