Problem stabilnih porok

Vesna Zupanc, Valentina Pucer, Jure Lah

19. februar 2017

1 Problem stabilnega ujemanja

Problem stabilnega ujemanja je problem iskanja stabilnega prirejanja na dveh enako velikih množicah, kjer so za vsak element dane urejene preference. Prirejanje je preslikava elementov iz ene množice v drugo množico. Prirejanje ni stabilno, če:

1. obstaja element A v prvi množici, ki preferira nek element B iz druge množice pred trenutnim elementom in
2. tudi B preferira A pred elementom, kateremu je trenutno prirejen.

Z drugimi besedami je prirejanje stabilno, če ne obstaja nobeno prirejanje (A, B), da bi bila A in B na boljšem kot trenutno.

2 Uporaba

Algoritmi za iskanje rešitev problema stabilnega ujemanja imajo v realnem življenju več različnih uporab. Najbolj znana med njimi je razporeditev diplomantov medicine v bolnišnice, kot prvo zaposlitev. Pomembna uporaba stabilnega ujemanja je tudi pri določanju uporabnikov strežnikom v večjih internetnih sistemih. Milijarde uporabnikov dostopajo do spletnih strani, videov in ostalih storitev na internetu, kar zahteva določanje posameznega uporabnika enemu (potencialnemu) izmed mnogih strežnikov po svetu, ki ponujajo to storitev. Vsak uporabnik preferira strežnike, ki so mu dovolj blizu, da omogočajo hitrejši odgovor za določeno storitev. Na drugi strani strežnik preferira tiste uporabnike, ki so zanj cenejši.

Opomba. Prirejanje je lahko le med moškim in žensko.

Poroke so stabilne Naj bosta Ana in Bor oba zaročena, vendar ne drug z drugim. Ob zaključku algoritma ne obstaja možnost, da oba preferirata drug drugega pred svojim trenutnim partnerjem. Če Bor preferira Ano pred svojo trenutno partnerko, potem jo je moral zaprositi preden je zaprosil trenutno partneko. Če je Ana sprejela zaroko, a se na koncu z njim ni poročila, pomeni, da ga je zamenjala za nekoga boljšega. Če pa ga je zavrnila, je že bila z nekom boljšim.

D. Gale in L. S. Shapley sta l. 1962 dokazala, da je vedno mogoče rešiti problem stabilnih porok tako, da so vse poroke stabilne, za poljubno enako število žensk in moških ter predstavila algoritem, ki to stori.

3 Naš problem

Naša naloga je sprogramirati algoritem, ki reši problem, torej izračuna stabilno ujemanje ter ima polinomsko zahtevnost in napisati metodo z nastavkom problema v celoštevilkem linearnem programu, ki nam prav tako da rešitev. Na koncu bomo obe metodi primerjali. Za algoritem s polinomsko zahtevnostjo smo si izbrali kar Gale-Shapleyev algoritem, ki je na kratko opisan v nadaljevanju.

3.1 Gale - Shapleyev algoritem

Podanih imamo n moških in n žensk. Vsak/-a izmed njih je ocenil/-a vse osebe naspro- tnega spola tako, da jim je podelil/-a vsa števila od 1 do n v vrstnem redu padajoče všečnosti.

Algoritem poteka v t.i. krogih. Vsak od njih sestoji iz dveh delov: dejanj moških ter dejanj žensk:

- Vsak moški "prosi za roko"žensk po padajočem redu všečnosti.
  - Ko se zaroči, preneha s prošnjami.
  - Če zaročenka razdre zaroko, nadaljuje s prošnjami od tam naprej, kjer je ostal.
- Vsaka ženska čaka na ponudbo nekega snubca.
  - Sprejme prvo ponudbo, ki jo dobi.
  - Če je ze zaročena z moškim m, snubi pa jo boljši moški m', razdre zaroko z m in se zaroči z m'.

Vsaka sprememba zaročenca ženske poveča njeno zadovoljstvo. Vsaka sprememba zaročenke moškega zmanjša njegovo zadovoljstvo. Ta proces se ponavlja dokler niso vsi zaročeni.

Časovna zahtevnost tega algoritma je $O(n^2)$, torej je to polinomska zahtevnost. Algoritem zagotavlja, da na koncu ni moškega in ženske, ki nista zaročena, saj jo je on sigurno zaprosil za roko v nekem trenutku (ker moški na koncu zaprosi vsako žensko, če je to potrebno) in ker je zaprošena, mora biti z nekom zaročena tudi ona (lahko tudi z drugim, saj ga lahko zamenja za boljšega).

```
#v funkcijo vpišemo število moških in žensk (enako) ter matriki preferenc
GSA <- function(n=n,mPref=M,zPref=Z){
  #Moški, ki so samski (na začetku so vsi)
  m.samski <- 1:n
```

```
  zarocenke <- rep(0,n)
  #Število žensk, ki jih zaprosi moški
  m.zgodovina <- rep(0,n)
  while (length(m.samski) !=0){
    for (j in m.samski){
      m.zgodovina[j] <- m.zgodovina[j] + 1 #Zasprosi eno več kot prej
      #j-ti moški v samskih zaprosi žensko, ki jo naslednjo preferira
      zaprosena <- mPref[m.zgodovina[j],j]
      if (zarocenke[zaprosena] ==0){
        #če je zaprošena ženska samska, potem se zaroči z moškim, ki
        #jo je zaprosil
        zarocenke[zaprosena] <- j
        #odstranimo j-tega moškega iz seznama samskih
        m.samski <- m.samski[-match(j, m.samski)]
      }
      else if (match(j,zPref[,zaprosena]) < match(zarocenke[zaprosena],
               zPref[,zaprosena])){
        # Če je vrednost j-tega moškega pri zaprošeni ženski višja
        # kot vrednost s katerim je trenutno zaročena,
        #zapusti trenutnega moškega in se zaroči z j-tim
        zapusceni <- zarocenke[zaprosena] #moški, ki ga bo zapustila
        zarocenke[zaprosena] <- j #nov zaročenec
        #odstranimo j-tega moškega iz seznama samskih
        m.samski <- m.samski[-match(j, m.samski)]
        #dodamo moškega, ki ga je zapustila v seznam samskih
        m.samski <- c(m.samski,zapusceni)
      }
    }
  }
  pari <- data.frame("Moški"=zarocenke,"Ženska"=1:n)
  return(list(mPref=mPref,zPref=zPref,ujemanje=pari))
}
```

3.2 ILP algoritem

Formulacija ILP. Predstavimo dvodelni graf $G = (M \cup Z, E)$. Naj bo M množica vseh moških in Z množica vseh žensk ter naj $uv \in E$ predstavlja možnost, da se moški (u) lahko poroči z žensko (v). Uporabimo binarno spremenljivko x_{ij}, ki je

enaka 1, če sta vozlišči i in j povezana in 0 sicer. Zapišemo:

$$\begin{aligned}
\min \quad & \sum_{i,j} x_{ij} \ \forall i \in M, \forall j \in Z \\
\text{p.p.} \quad & \sum_{i} x_{ij} = 1 \ \forall i \in M \\
& \sum_{j} x_{ij} = 1 \ \forall j \in Z \\
& \sum_{i >_u v} x_{ui} + \sum_{j >_v u} x_{vj} + x_{uv} \geq 1
\end{aligned}$$

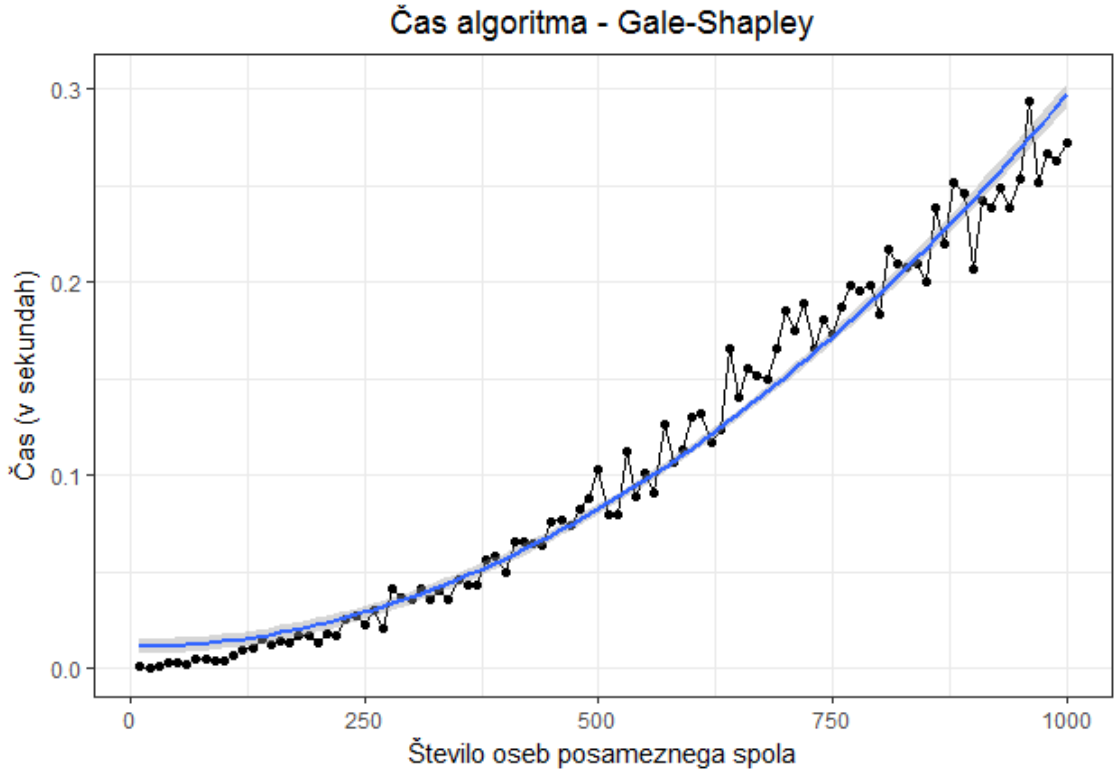
Prva dva pogoja nam zagotavljata, da ima vsako vozlišče natanko eno povezavo (torej vsaka ženska je zaročena z natanko enim moškim in obratno). Tretji, kjer vsota po $i >_u v$ predstavlja vsa vozlišča, ki jih u preferira nad v in $j >_v u$ vsa vozlišča, ki jih v preferira nad u, pa zagotavlja, da ne obstaja nobena povezava med dvema drugima vozliščema, ki bi jo preferirali nad trenutno (z drugimi besedami zagotavlja stabilnost parov).

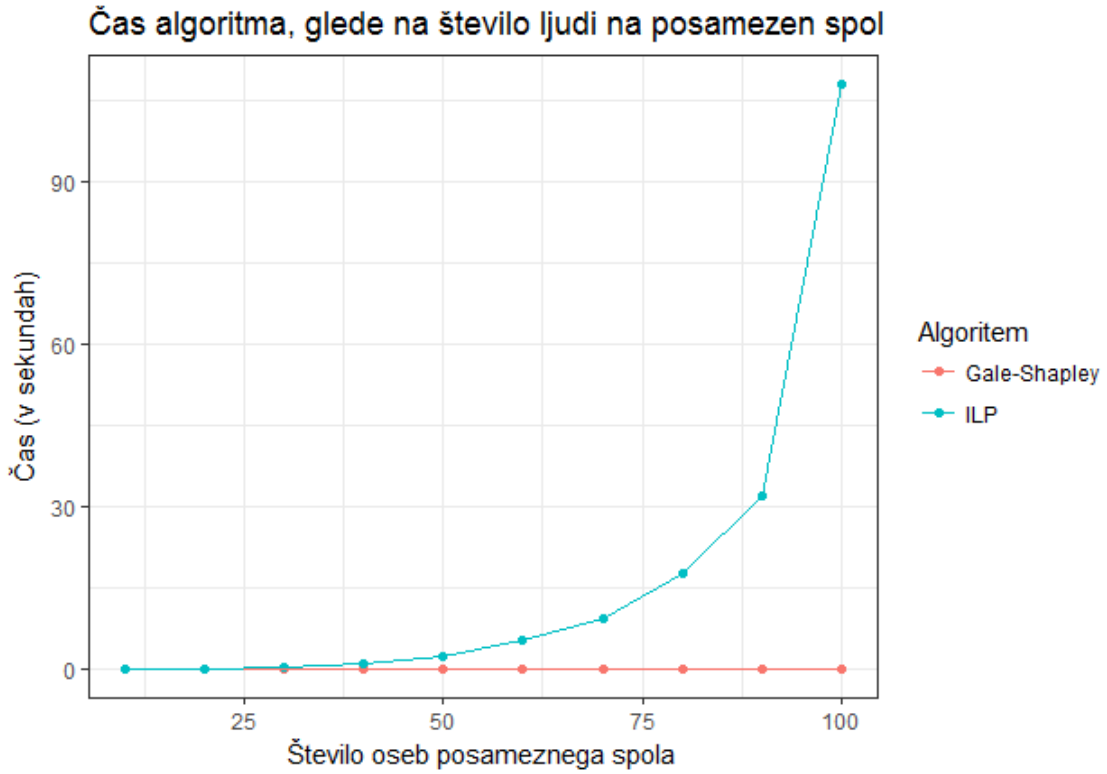
```
p = MixedIntegerLinearProgram(maximization = False)
x = p.new_variable(binary = True)
from random import shuffle
n=10
moski = []
zenske = []
pari=[]
for i in range(n):
    M = [j for j in range(n)]
    shuffle(M)
    moski.append(M)
    Z = [j for j in range(n)]
    shuffle(Z)
    zenske.append(Z)

p.set_objective(sum(sum(x[i,j] for j in range(n)) for i in range(n)))
for i in range(n):
    p.add_constraint(sum(x[i,j] for j in range(n))== 1)
for j in range(n):
    p.add_constraint(sum(x[i,j] for i in range(n))== 1)

for u, a in enumerate(moski):
    for v, b in enumerate(zenske):
        p.add_constraint(sum(x[u, i] for i in a[:a.index(v)]) +
        sum(x[i, v] for i in b[:b.index(u)]) + x[u, v] >= 1)

p.solve()
```

3.3 Primerjava algoritmov

Primerjali smo koliko časa potrebuje posamezen program za izračun stabilnih parov. Ker je čas, ki ga potrebuje Gale Shapleyev algoritem, za izračun stabilnih parov precej krajši kot celoštevilski linearni program smo izračunali čas za število ljudi posameznega spola do 1000, medtem ko smo za ILP uporabili število ljudi do 100. Pri izračunu nismo upoštevali časa, ki je bil potreben za generiraje preferenc za posamezen par.

Prvi graf prikazuje čas izračuna stabilnih parov za Gale-Shapleyev algoritem (število oseb do 1000) drugi pa primerjavo časa, ki ga potrebujeta algoritma za enako število oseb (do 100). Vidimo lahko, da ILP algoritem potrebuje za izračun parov pri največjem številu oseb več kot 90 sekund, medtem ko za enako število ljudi Gale - Shapleyev algoritem porabi manj kot sekundo.




Čas algoritma, glede na število ljudi na posamezen spol
Čas (v sekundah)
90
60
30
0
25
50
75
100
Število oseb posameznega spola
Algoritem
Gale-Shapley
ILP