Matching

Ian Teixeira Barreiro

Janeiro 2022

1 Introdução

A classe de problemas que será discutida na presente secção é aquela onde é necessário fazer a associação entre objetos ou pessoas de dois grupos distintos em pares, acomodando restrições que esses objetos ou pessoas impõe. Dito de outra maneira é necessário fazer o "casamento"dos membros de cada um desses grupos. Este tipo de problema, chamado à partir de agora de *matching*, aparece em diversos cenários, tais como o pareamento de dançarinos de acordo com as preferências de cada um sobre com quem preferem dançar, a formação de casais por um serviço de casamento, a associação de um estudante a uma universidade, entre outros. De fato, tais problemas são diferentes daqueles geralmente estudados em economia pois podem ser vistos como um "mercado de dois lados": diferente dos mercados normais em que os objetos comprados geralmente não possuem preferências, nesse caso deve ser feita a associação entre dois seres que ambos possuem preferências próprias. O problema, portanto é de qual é o melhor algorítimo para fazer essa organização em pares e qual será o resultado global atingido utilizando essa estratégia de pareamento.

A presente secção utilizará como referências o livro "Microeconomia: Uma Abordagem Moderna, 8a ed."de Hal Varian e o texto "College Admissions and the Stability of Marriage"de David Gale e Lloyd Shapley. Ela se estruturará da seguinte maneira: i) primeiro será feita uma descrição tipo de problema com que estamos lidando, tendo como referência uma instância particular do problema que será utilizada para ilustrar o conceito de estabilidade e solução ótima; ii) em seguida será provido um algorítimo que encontra soluções para esse tipo de problema, e será provado que essas soluções são estáveis e ótimas; iii) por fim será descrita uma implementação computacional para resolver problemas de *matching*.

2 Descrição do Problema

Como foi dito, existem várias situações em que se veem problemas de *matching*. Por questão de simplicidade, tomaremos como referência o problema do casamento entre pessoas de dois grupos. Imagine que existem dois grupos, cada um com n membros, que contrataram um serviço para fazerem o casamento

entre seus membros. Cada membro de cada grupo possui preferências perfeitamente ordenadas (não existem empates) acerca de com quem preferem ser pareados no outro grupo. Para ilustrar o que queremos dizer, observe a tabela a seguir, em que α, β, γ e δ são membros de um grupo, chamado grupo 1, e A, B, C e D são membros de outro, chamado grupo 2.

	A	B	C	D
α	(1, 3)	(2, 2)	(3, 1)	(4, 3)
β	(1, 4)	(2, 3)	(3, 2)	(4, 4)
γ	(3, 1)	(1, 4)	(2, 3)	(4, 2)
δ	(2, 2)	(3, 1)	(1, 4)	(4, 1)

Nessa tabela, os valores a esquerda indicam as ordens de preferência dos membros do grupo 1 e os valores a direita indicam as ordens de preferência dos membros do grupo 2. Queremos fazer o pareamento dos membros desses grupos respeitando essas preferências. Como se parecerá uma solução para esse problema?

2.1 Estabilidade

Em primeiro lugar temos que uma solução para esse problema deverá ser estável. Um determinado casamento dos membros dos grupos é dito instável se houver como formar um novo par em que a situação de ambos os membros desse novo par é melhorada. Como exemplo, tomemos a situação em que α está casado a B e A está casado a β. A situação em que α é casado a A é mais preferida para ambos. Portanto, se possível, ambos tentarão desfazer seu pareamento atual para formar esse novo par. Uma solução para o problema de *matching* não deve ter nenhum casamento instável, caso contrário, não poderá ser considerada uma solução, já que na primeira oportunidade os membros do par instável tentarão desfazer seu vínculo.

2.2 Soluções Ótimas

Podem haver várias soluções estáveis para um problema de *matching*, mas há soluções estáveis que são superiores a outras. Uma solução estável é dita uma solução ótima se nenhum aplicante preferiria estar em outra solução estável. Dita de outra maneira, a solução ótima deve ser pelo menos tão boa para todos os membros quanto qualquer outra solução estável. Apenas cumprindo com essa condição uma solução pode ser dita ótima.

3 Algorítimo para Encontrar Soluções Estáveis

Diante dos conceitos expostos anteriormente, é fácil perceber que a solução que buscamos para o nosso problema é uma solução estável ótima. No entanto, podemos nos indagar se existe uma solução estável ótima para todo problema

de *matching* ou até se existem soluções estáveis? A resposta é que existem sempre soluções estáveis para o tipo de problema descrito acima, e elas podem ser encontradas pelo procedimento exposto a seguir.

Passo 1: Cada membro de um grupo faz propostas de casamento para seu membro mais preferido do outro grupo.

Passo 2: Cada membro do outro grupo recebe os pedidos de casamento, e rejeita todos os pedidos que não o pedido mais preferido, que ele mantém em uma lista de pretendentes. Caso já houvesse uma pessoa na lista de pretendentes, se um novo pedido é mais preferido que o dessa pessoa, ela é removida da lista de pretendentes em favor da nova pessoa mais preferida.

Passo 3: Todos aqueles que foram rejeitados fazem propostas de casamento para seu próximo membro mais preferido do outro grupo.

Passo 4: Iteramos os passos anteriores até que todos os membros do outro grupo tenham recebido uma proposta. Nesse ponto, encerra-se a iteração e todos se casam com base na lista de pretendentes.

Tal procedimento levará obrigatoriamente a uma configuração estável. Isso por que caso um membro de um grupo preferisse estar casado com outro membro de outro grupo, já que ele fez pedidos de acordo com sua preferência, isso significa que ele fez um pedido de casamento para essa pessoa mais preferida e foi rejeitado, de modo que a pessoa está casada com alguém que ela mais prefere, não havendo instabilidade.

É importante notar que a depender de qual grupo tomamos como o grupo de início, serão encontradas soluções estáveis distintas.

3.1 Este Algorítimo Encontra Soluções Ótimas?

Outra prova importante é a de que o procedimento citado acima, além de encontrar soluções estáveis, também encontra soluções ótimas. Definamos um casamento como possível caso haja uma solução estável em que ele ocorre. Imaginemos uma situação em que estão sendo feitos pareamentos e ninguém foi rejeitado ainda. Suponhamos que α e β preferem A, mas A prefere β em relação a α. Qualquer pareamento entre α e A é instável, pois é mais interessante para A e para β que eles sejam pareados juntos. Assim, o pareamento entre α e A é impossível (não há solução estável em que ela ocorra). Desse modo, α deverá fazer propostas de casamento para os próximos membros de sua lista de preferência, até o ponto em que ocorrerá um pareamento possível, que será o mais preferido para α dentre os pareamentos possíveis para ele. Assim, o algorítimo, ao iterar as preferências dos participantes do mais preferido para o menos preferido, ao encontrar uma solução estável também encontrará uma solução em que cada indivíduo está pareado com o seu par mais preferido dentre os pares possíveis.

4 Implementação Computacional

A implementação computacional será descrita função por função. A primeira função foi criada para organizar as propostas feitas pelas pessoas no primeiro

grupo. Nesse sentido, primeiramente são criadas duas listas: a lista organiza_propostas que receberá listas no mesmo número que a quantidade de pessoas no grupo 2, servindo para armazenar as propostas recebidas por cada pessoa do grupo 2 e a lista solteiros, que armazena todos aqueles que são solteiros. As propostas são organizadas for meio de um loop que busca o primeiro elemento de cada ordenação de preferências na matriz de preferências (correspondente à pessoa mais preferida de cada membro do grupo 1). Caso a pessoa mais preferida seja ficar solteiro, a pessoa do grupo 1 será adicionada à lista dos solteiros. Caso contrário, a proposta da pessoa é registrada na lista que organiza as propostas. Por fim, são retornadas ambas as listas criadas dentro da função.

```
1
2  def organiza_propostas(lista_preferencias, n_pessoas_g2):
3
4      """
5      Com base na lista de preferencias do primeiro grupo, na lista
6      de membros do primeiro grupo e no numero de pessoas no segundo
7      grupo, forma uma lista com as propostas que cada pessoa do
8      segundo grupo recebeu. O primeiro elemento da lista representa
9      as propostas que a primeira pessoa do segundo grupo recebeu e
10     assim por diante.
11     ---------------------------------
12
13     lista_preferencias: lista de preferencias do primeiro grupo
14     n_pessoas_g2: o numero de pessoas no segundo grupo
15
16     """
17
18     organiza_propostas = []
19     solteiro = []
20
21     # Preenche organiza propostas com listas vazias na
22     # mesma quantidade que existem pessoas que recebem propostas
23     for preferencia in range(n_pessoas_g2):
24
25         organiza_propostas.append([])
26
27     # Para cada pessoa que lanca propostas, recebe sua pessoa mais
       preferida
28     # e apensa a lista de propostas recebidas de sua pessoa mais
       preferida
29     # o numero correspondente a ela
30     for pessoa in range(len(lista_preferencias)):
31
32         mais_preferido = lista_preferencias[pessoa][0]
33
34         if mais_preferido == 0:
35
36             solteiro.append(pessoa + 1)
37
38         else:
39             organiza_propostas[mais_preferido - 1].append(pessoa +
       1)
40
41     return organiza_propostas, solteiro
```

A segunda função forma os casamentos entre membros de ambos os grupos e registra as rejeições daqueles que não foram escolhidos. Para isso, primeiramente é criada uma lista para armazenar o pareamento dos membros dos grupos e a lista de booleanos que identifica quem foi rejeitado nessa iteração do algorítimo. Depois, para cada proposta feita vê-se qual é a proposta mais atraente e é feito o pareamento. Caso a pessoa do grupo 2 prefira ficar sozinho, também é registrado esse fato. Todas as pessoas rejeitadas tem sua rejeição registrada no vetor booleano. Por fim, as pessoas solteiras do grupo 1 são registradas na lista de pareamentos.

```
1
2  def casamentos_e_rejeicoes(organiza_prop, solteiros, lista_pref2,
       n_pessoas_g1):
3
4      """
5      Recebe o vetor de propostas e o vetor de preferencias do
6      segundo grupo e retorna os pareamentos preliminares entre
7      os grupos e a lista de quais pessoas do primeiro grupo
8      foram rejeitadas
9      ---------------------------------
10
11     organiza_prop: lista das propostas criada pela funcao
       organiza_propostas
12     solteiros: lista de pessoas solteiras
13     lista_pref2: a lista de preferencias dos membros do segundo
       grupo
14     n_pessoas_g1: numero de pessoas no grupo 1
15
16     """
17
18     casamento_prel = []
19     nova_lista_rej = [False for i in range(n_pessoas_g1)]
20
21     # Para cada lista de propostas
22     for proposta in range(len(organiza_prop)):
23
24         # Busca na lista de preferencias da pessoa que
25         # recebeu as propostas, partindo das pessoas
26         # mais preferidas para as menos preferidas
27         for preferencia in lista_pref2[proposta]:
28
29             # Faz o casamento preliminar entre a melhor
30             # proposta e a pessoa que recebeu as propostas
31             if preferencia in organiza_prop[proposta]:
32
33                 casamento_prel.append([preferencia, proposta + 1])
34
35                 # As demais opcoes que nao a melhor proposta
36                 # sao rejeitadas
37                 for pessoa in organiza_prop[proposta]:
38
39                     if pessoa != preferencia:
40
41                         nova_lista_rej[pessoa - 1] = True
42
43                 break
```

```
44
45              # Se a pessoa prefere ficar solteira
46              elif preferencia == 0:
47
48                  casamento_prel.append([preferencia, proposta + 1])
49
50                  for pessoa in organiza_prop[proposta]:
51
52                      nova_lista_rej[pessoa - 1] = True
53
54                  break
55
56      # Append das pessoas solteiras na lista de casamentos
        preliminares
57      if len(solteiros) != 0:
58
59          for solteiro in solteiros:
60
61              casamento_prel.append([solteiro, 0])
62
63      return casamento_prel, nova_lista_rej
```

A função atualiza preferências recebe uma lista de preferencias a ser atualizada e o vetor de booleanos que indica quem foi rejeitado na presente iteração. Todos aqueles que foram rejeitados tem o primeiro elemento de sua ordenação de preferências removido dessa ordenação.

```
1  def atualiza_preferencias(lista_preferencias, lista_rej):
2
3      """
4      Com base em quem foi rejeitado na ultima iteracao
5      atualiza as preferencias das pessoas do primeiro grupo
6      ---------------------------------
7
8      lista_preferencias: a lista de preferencias a ser atualizada
9      lista_rej: a lista de quem foi rejeitado na ultima iteracao
10
11     """
12
13     # Para cada pessoa da lista de rejeitados
14     for i in range(len(lista_rej)):
15
16         # Se a pessoa tiver sido rejeitada remove
17         # o primeiro elemento da sua lista de preferencias
18         if lista_rej[i]:
19
20             lista_preferencias[i].pop(0)
21
22     return lista_preferencias
```

A função iteração repete as funções acima descritas até que de uma iteração para outra não haja alteração nos pareamentos feitos. Desse modo, a função iteração executa o algorítimo de solução do problema de *matching* até que o equilíbrio seja encontrado.

```
1  def iteracao(vec_grupo1, vec_grupo2, n_igual = True):
2
```

```
    """
    Recebe os vetores de preferencias de cada um dos grupos e
    retorna a configuracao final dos pareamentos entre os
    grupos respeitando as restricoes
    ---------------------------------

    vec_grupo1: lista de preferencias do primeiro grupo
    vec_grupo2: lista de preferencias do segundo grupo
    n_igual: booleano se cada grupo tem o mesmo numero de membros

    """

    # Armazena a quantidade de pessoas no segundo grupo
    if n_igual:
        n_pes2 = len(vec_grupo1[0])
        n_pes1 = n_pes2
    else:
        n_pes2 = len(vec_grupo1[0]) - 1
        n_pes1 = len(vec_grupo2[0]) - 1

    # Lista criada para armazenar os casamentos preliminares
    # da iteracao anterior
    casa_prel_anterior = []

    # Cria listas de preferencias, de pessoas em cada grupo e de
    # rejeicoes no primeiro grupo
    lp1, lp2, lg1, lg2, rej = cria(vec_grupo1, vec_grupo2)

    # Organiza as primeiras propostas de casamento
    org_prop, sol = organiza_propostas(lp1, n_pes2)

    # Organiza os primeiros casamentos preliminares e rejeicoes
    casa_prel, rej = casamentos_e_rejeicoes(org_prop, sol, lp2,
    n_pes1)

    # Atualiza preferencias com base nas primeiras rejeicoes
    lp1 = atualiza_preferencias(lp1, rej)

    # Enquanto os casamentos preliminares se alterarem
    # de uma iteracao para outra, continua formando
    # novos pares
    while casa_prel != casa_prel_anterior:

        casa_prel_anterior = casa_prel
        org_prop, sol = organiza_propostas(lp1, n_pes2)
        casa_prel, rej = casamentos_e_rejeicoes(org_prop, sol, lp2,
     n_pes1)
        lp1 = atualiza_preferencias(lp1, rej)

    #nome_casamento = nomeia_casamentos(casamentos, nomes_grupo1,
    nomes_grupo2)

    return casa_prel
```

Por fim, a função nomeia_casamentos recebe os pares em equilíbrio encontrados pela função iteração e também vetores contendo os nomes dos membros de cada grupo. A função transforma a lista de pares em equilíbrio em uma lista

que contém os nomes dos membros, ao invés de apenas conter os identificadores numéricos.

```
def nomeia_casamentos(casamentos, nomes_grupo1, nomes_grupo2):

    """
    Transforma a lista de casamentos identificados por
    identificadores
    numericos em uma lista de casamentos identificados pelos nomes
    dos
    membros de cada grupo
    ---------------------------------

    casamentos: lista de casamentos identificados pelo
    identificador numerico
    nomes_grupo1: lista de strings com os nomes dos membros do
    grupo1
    nomes_grupo2: lista de strings com os nomes dos membros do
    grupo2

    """

    casamentos_nomeados = []

    for par in casamentos:

        casamentos_nomeados.append([nomes_grupo1[par[0] - 1],
                                    nomes_grupo2[par[1] - 1]])

    return casamentos_nomeados
```