Documentação TP1 - Um Novo Sistema de Seleção Unificada

Isadora Alves de Salles

16/04/2019

1 Introdução

Este trabalho teve como objetivo criar uma modificação no Sisu de tal forma que no final do processo todos os candidatos obtenham a maior satisfação possível, ou seja, sejam alocados da melhor forma. Sendo assim, o resultado esperado baseia-se no resultado que o algoritmo de Gale-Shapley retorna pois, o mesmo otimiza o emparelhamento estável para quem fazendo a proposta, que nesse caso serão os candidatos.

Cada candidato terá uma lista de prioridades e cada universidade terá uma nota de corte, e esses são os requisitos que temos que atender. Algumas restrições para o problema:

- As notas de corte e o número de vagas das universidades devem ser respeitados.
- Se há um candidato A que está desalocado mas há outro candidato B com nota inferior a de A alocado em uma das universidades listadas por A, isso configura um par instável e não deve ocorrer.
- Se dois candidatos tiraram a mesma nota e A está alocado em X e B está alocado em Y, porém A prefere Y e B prefere X, isso também configura um par instável e não deve ocorrer.
- Qualquer outra configuração em que exista um par instável não deve ocorrer.
- As listas de preferências das universidades são sempre a mesma, visto que todas elas preferem alunos com notas maiores. O que diferencia cada uma delas é a nota de corte.
- Um candidato não pode ser alocado para uma universidade que não está em sua lista de prioridades.
- Quando todos os aspectos forem iguais, o candidato com menor id terá prioridade.

Então, dado a lista de candidatos com suas notas e suas respectivas lista de preferências de universidades e a lista de universidades com número de vagas e nota de corte, a tarefa é conseguir o melhor casamento estável, seguindo as restrições ditas acima.

2 Solução do Problema

Primeiramente, optei por não criar um vetor para armazenar a lista de candidatos que aplicaram para cada universidade ordenados de forma decrescente por nota, pois é um tanto quanto redundante já que ao caminhar pela lista de prioridade dos candidatos saberemos para quais universidades ele aplicou, e nesse caso basta conferir se a nota do candidato é adequada as restrições de nota de corte das universidades que ele deseja e se há vagas. Porém agindo dessa forma, foi necessário ordenar as notas de todos os candidatos de forma decrescente antes de iniciar a alocação, para estabelecer uma ordem de leitura dos candidatos que facilite a alocação e manter a ordem de preferência das universidades.

Pensando em uma forma de otimizar o algoritmo de Gale-Shapley e tendo em vista que os candidatos com as maiores notas serão alocados com maior certeza do que os candidatos com as menores notas, optei por ordenar as notas do ENEM, em ordem decrescente, antes de realizar o emparelhamento estável. Assim, foi possível alocar os candidatos que tiraram as maiores notas primeiro, visto que nenhum outro candidato que vier depois dele causará um par instável pois as universidades preferem notas maiores. Dessa forma, consegui evitar o processo de trocas que ocorre no algoritmo de Gale-Shapley. Quando um candidato é alocado em uma universidade o par já será parte da solução final ótima.

Para conseguir implementar dessa forma foi usado uma tabela hash de tamanho 1000 (que é a nota máxima que um candidato pode obter) em que a hashKey será a nota dos candidatos, e para cada célula da tabela teremos uma lista encadeada com o id dos candidatos que tiraram a nota em questão. Assim, criei uma tabela hash usando o método de tratamento por colisão com lista encadeada, em que uma colisão ocorre quando mais de um aluno tira exatamente a mesma nota. Simultaneamente ao preenchimento da tabela, um vetor foi criado para armazenar as notas, sem repetição, e no fim da leitura de entradas o mesmo será ordenado usando o algoritmo quickSort, e assim os candidatos serão encontrados através do caminhamento nesse vetor e uma busca em tempo constante na tabela.

Para os candidatos foi alocado um vetor de listas encadeadas, em que o index do vetor representa o candidato e as listas contém as universidades que o candidato prefere. A cada iteração no vetor de notas temos um hashGet com o qual obtemos a lista de candidatos que tiraram aquela nota, e para cada candidato avalia-se a primeira preferência, se o candidato cumprir com o requisito da nota de corte e ainda existirem vagas, o mesmo é alocado na universidade. Depois de analisar a primeira preferência de cada um dos candidatos com notas iguais, repete-se o processo e analisamos a segunda preferência dos candidatos que não foram alocados na sua primeira preferência, isso será repetido até que todos os candidatos que tiraram essa nota sejam alocados ou atingirmos o final das listas de preferências deles sem que seja possível alocar. E repetimos o processo pra a próxima iteração no vetor de notas. Realizando o procedimento dessa forma evita-se que hajam pares instáveis quando dois alunos tiram a mesma nota. Veja o exemplo de como será a saída do algoritmo num caso que poderia surgir um par instável quando dois candidatos tiram a mesma nota:

```
ARQUIVO CONTENDO OS CANDIDATOS

3
1 105
1
2 95
1 2
1 95
2


ARQUIVO CONTENDO AS UNIVERSIDADES

2
1 90
1 90


Grupos com alocacao
1 1
3 2
Candidatos nao alocados
2
```

Figura 1: Exemplo de teste que poderia causar par instável.

Como visto no exemplo acima, os candidatos 2 e 3 tiraram a mesma nota e ambos aplicaram para a universidade 2 que tem apenas uma vaga, porém o candidato 3 colocou a universidade 2 como a primeira em sua lista de prioridades, então ele será alocado a ela e o candidato 2 ficará desalocado.

Resumindo, da forma que o algoritmo foi implementado primeiro percorre-se a primeira preferência de todos os candidatos que tiraram a nota X, depois olhamos a segunda preferência dos candidatos que tiraram a nota X mas não foram alocados na primeira universidade de preferência. Com isso, não é necessário que voltemos atrás para corrigir a alocação de algum candidato que está resultando em pares instáveis, pois estamos olhando para a preferência da faculdade (ordem de notas decrescente, nota de corte e quantidade de vagas) e a dos candidatos (ordem de universidades) simultaneamente. A ideia é que, temos mais certeza de que os candidatos que tiraram as maiores notas serão alocados em uma das universidades para as quais eles aplicaram do que os candidatos com menores notas, assim caminhar pelos candidatos através da ordem decrescente das notas, ao invés de caminhar por ordem do id, irá poupar o esforço da correção na alocação, pois as vagas nas universidades já serão preenchidas de forma adequada (maiores notas) e de forma a priorizar as preferências dos alunos. Além disso, como os candidatos são inseridos em ordem na tabela hash, a restrição de que se todos os aspectos dos candidatos forem iguais o candidato com menor id terá prioridade também será considerada, pois as células da tabela estarão preenchidas pelos alunos em ordem pelo id.

A seguir um pseudocódigo da função que realiza o casamento estável entre as universidades e os candidatos:

```
function StableMatching(hash){
        for all notas in vector_notas do {
                busca = hashGet(nota);
                while(busca != NULL){
                        for all elements in busca do{
                                if not NULL{
                                        if is possible alloc in the university{
                                                alloc;
                                                hashRemove;
                                        }
                                        else
                                                remove university from list of preferences;
                                }
                                else
                                        hashRemove;
                        busca = hashGet(nota);
                }
        }
}
```

Figura 2: Pseudocódigo para a função de casamento estável.

Pode-se afirmar que o algoritmo implementado é um algoritmo guloso pois parte-se do princípio que a cada etapa fazemos uma escolha localmente ótima, em busca de um ótimo global que será nossa solução final.

3 Análise de Complexidade Assintótica

Denotaremos por n o número de universidades e m o número de candidatos.

3.1 Complexidade Assintótica de Tempo

O programa executa três partes principais: lê as entradas, inicializa e ordena o vetor de notas e realiza o casamento estável entre as universidades e os candidatos. Vamos então analisar as complexidades separadamente.

3.1.1 Leitura dos dados: O(m+n)

Nesta etapa cada uma das n universidades é lida e em seguida cada um dos m candidatos, em tempo O(1). Assim a complexidade é O(m+n).

3.1.2 Vetor com as notas: O(mlogm)

Um vetor de tamanho igual ao número de candidatos é criado, e inicializado com zeros, tendo custo O(m). Simultaneamente a leitura do arquivo de candidatos esse vetor é preenchido com as notas, sem valores repetidos, no pior caso cada um dos candidatos tira uma nota diferente e teremos que ordenar um vetor contendo m valores. Então, usando o algoritmo quickSort para ordenar as notas de forma decrescente temos que a complexidade desta etapa é O(mlogm).

3.1.3 Função de casamento estável: O(n*m)

No inicio da função o vetor que armazena as alocações dos candidatos é inicializado com zeros, tendo custo O(m).

A função que realiza o casamento estável possui 3 loops aninhados, porém não possui complexidade cúbica. Isso ocorre pois, o primeiro loop é o que percorre o vetor de notas e faz-se uma busca O(1) na tabela hash, que foi feito apenas para facilitar o acesso e a alocação dos candidatos ordenados pela maior nota. Assim, para cada iteração no vetor de notas teremos uma certa quantidade de candidatos, e para cada candidato uma certa quantidade de universidades para as quais ele aplicou. Logo, teremos que percorrer na verdade por todos os candidatos e todas as preferências deles. O pior caso ocorre quando todos os candidatos aplicam para todas as universidades, assim a complexidade será O(n*m).

A complexidade final dessa função será o max(O(m), O(n*m)) = O(n*m).

Assim a complexidade assintótica de tempo do programa é O(m+n) + O(mlogm) + O(n*m) = O(n*m).

3.2 Complexidade Assintótica de Espaço

Além de variáveis simples, 3 vetores são armazenados em memória. Sendo que um deles é o vetor que armazena os resultados do casamento estável, ele possui m posições que representam cada um dos candidatos, portanto seu custo de espaço é O(m).

Os outros dois vetores representam a universidade e a lista de preferência dos candidatos. Sendo o primeiro um vetor de structs que contém o número de vagas e a nota de corte de cada uma das universidades representadas pelo index, sua custo de espaço é O(2n) pois ele armazena em n posições duas variáveis. O segundo é um vetor de listas encadeadas contendo as prioridades dos candidatos representados também pelo index no vetor, este ocupa um espaço na memória de O(m*n), pois no pior caso cada uma das m posições do vetor irá armazenar uma lista de n universidades.

Por fim, foi implementada uma tabela hash com 1000 posições, sendo que cada uma das células irá armazenar uma lista encadeada contendo os candidatos que tiraram a nota representada pela hashKey. Assim a custo será O(1000+m), pois será alocada uma tabela de 1000 posições e teremos m candidatos para alocar dentro dessa tabela.

Assim a complexidade assintótica de espaço do programa é O(m) + O(2n) + O(m*n) + O(1000+n) = O(m*n).

4 Análise Experimental

Com o intuito de tirar algumas conclusões sobre o novo sistema construído, analisei as métricas de satisfação dos candidatos e a taxa de alocação, para tentar entender como estas se comportam a medida que há um aumento no tamanho da lista de preferências dos candidatos. Para isso criei um arquivo de teste para as universidades, o qual contém 15 universidades com 28 vagas no total, e em cada execução do programa esse arquivo continua sendo o mesmo, o que varia é o arquivo de entrada dos candidatos. Neste caso, a entrada do candidato só será modificada quanto à lista de preferências, as notas e quantidades de candidatos permanecem os mesmos, pois dessa forma é possível observar o quanto o aumento da lista de preferências pode influenciar na satisfação do candidato e na taxa de alocação. A seguir um exemplo dos arquivos de entrada usados para realizar essa análise e a saída encontrada:

```
ARQUIVO CONTENDO OS CANDIDATOS
20
2 85
1 2
2 95
1 2
2 90
2 4
2 100
15 6
2 105
5 8
2 120
13 11
2 100
5 7
2 95
8 9
2 95
9 8
2 80
1 2
2 65
1 4
2 80
5 9
2 95
15 3
2 80
9 10
2 85
9 2
2 75
1 5
2 95
14 12
2 80
2 5
2 85
4 1
2 95
1 2
```

```
ARQUIVO CONTENDO AS UNIVERSIDADES
15
1 70
2 80
3 90
2 90
1 85
3 105
1 85
2 100
1 105
2 70
3 75
2 80
1 95
3 95
1 100

-----------------------------------

Grupos com alocacao
2 1
3 2
4 15
5 5
6 13
7 7
13 3
14 10
17 14
20 2
Candidatos nao alocados
1
8
9
10
11
12
15
16
18
19
```

Figura 3: Exemplo de teste.

O exemplo anterior é usado na primeira linha da Tabela 1. Como podemos observar na tabela, com o aumento do tamanho da lista de preferências temos mais candidatos alocados, e dessa forma também aumentaremos o grau de satisfação dos candidatos e a taxa de alocação. Assim, podemos concluir que permitir que a lista de prioridades seja maior faz com que a taxa de alocação aumente, pois o candidato terá mais chances de entrar na faculdade, proporcionalmente ao aumento do tamanho da lista. Da mesma forma que isso também influencia positivamente no gruau de satisfação dos candidatos, dado que quanto mais candidatos alocados espera-se que maior seja o grau de satisfação dos candidatos.

Tabela 1: Análise da satisfação e alocação dos candidatos a medida que aumentamos o tamanho da lista de preferências.

Entrada	Quant. de Preferências	Grau de Satisfação	Vagas Alocadas	Taxa de Alocação
in1	2	0,4262	10	0,3571
in2	3	0,4762	12	0,4286
in3	4	0,4977	13	0,4643
in4	5	0,5171	14	0,5
in5	6	0,5249	15	0,5357
in6	7	0,5682	17	0,6071

Agora, para o mesmo exemplo dado na Figura 3 , apenas alterei a quantidade de vagas total fornecida, mantendo a entrada do candidato inalterada sempre. A seguir a análise de como a taxa de alocação e o grau de satisfação dos candidatos irão se comportar:

Tabela 2: Análise da satisfação e alocação dos candidatos a medida que aumentamos a quantidade de vagas total fornecidas.

Entrada	Vagas Fornecidas	Grau de Satisfação	Vagas Alocadas	Taxa de Alocação
in7	28	0,4262	10	0,3571
in8	32	0,5446	12	0,3750
in9	35	0,5446	12	0,3428
in10	38	0,6762	15	0,3947
in11	41	0,6762	15	0,3658
in12	44	0,6762	15	0,3409

A métrica de grau de satisfação do candidato não leva em conta o quão vazia está a universidade, então aumentar a quantidade de vagas fornecidas não fará com que haja diminuição no grau de satisfação, apenas aumento ou se manter o mesmo. Como pode ser observado na Tabela 2 o aumento do número de vagas fornecidas pelas faculdades pode ou não aumentar o grau de satisfação. Isso ocorre pois, como não houve alteração nas listas de preferência dos candidatos alguns candidatos não são alocados simplesmente por não possuir uma nota superior a nota de corte das universidades para as quais ele aplicou. Assim, vagas irão sobrar nas universidades pois não há alunos que preencham o pré-requisito de nota.

Já a taxa de alocação sempre irá sofrer variação, devido ao fato de que a mesma é obtida pela razão de vagas alocadas por número total de vagas fornecidas, assim, quando não há um aumento no número de candidatos alocados com o aumento das vagas fornecidas a taxa de alocação irá diminuir. Dessa forma, não podemos assumir que quanto maior o número de vagas maior a taxa de alocação.

Note que o grau de satisfação cresceu mais com o aumento das vagas fornecidas do que com o aumento da lista de preferências, isso se deve ao fato de que esta métrica tende a ser maior quando os candidatos são alocados em suas faculdades "mais preferidas", ou seja, as primeiras em sua lista de prioridade, quando aumentamos o número de vagas, mais candidatos poderão ser alocados nas suas faculdades preferidas e isso afetará o grau de satisfação de maneira relevante. Outra observação que deve ser feita é que, se o número de vagas de todas as universidades fosse

aumentado simultaneamente, será esperado um grau de satisfação superior pois todos aqueles candidatos que se encaixam no requisito de nota mas não na quantidade de vagas seriam alocados.

Agora temos uma tabela com a diminuição da nota de corte média das universidades para observar como isso afeta as duas métricas analisada anteriormente, além da média da taxa de preenchimento das universidades. As análises continuam sendo feitas mantendo todas as outras variáveis inalteradas, neste caso mudando apenas a nota de corte das universidades.

Tabela 3: Análise da satisfação, alocação dos candidatos e média da taxa de preenchimento das universidades a medida que diminuímos a nota de corte.

Entrada	Nota de Corte	Satisfação	Taxa de Alocação	Média da Taxa de Preenchimento
in13	88	0,4262	0,3571	0,8111
in14	84	0,4762	0,3929	0,8444
in15	82	0,5262	0,4286	0,8444
in16	77	0,5262	0,4286	0,8444
in17	74	0,5262	0,4286	0,8444
in18	71	0,5762	0,4643	0,8444

Como pode ser observado na Tabela 3 a diminuição da nota de corte pode ter um efeito positivo nas três métricas, porém em determinado momento a diminuição da nota passa a não importar, isso ocorre pois deixa de existir candidatos que tiraram notas próximas as novas notas de corte que ainda poderiam ser alocados. Vale lembrar que a lista de prioridades dos candidatos e a quantidade de vagas das universidades não estão se alterando. Assim, em determinado momento a variação apenas da nota de corte passa a não modificar as métricas. Caso fossem tratadas as variações de todos os parâmetros juntos, seria possível dizer qual combinação resultaria em valores melhores para as métricas.

Resumindo, a partir dessas analises o que pode-se afirmar é que o aumento da lista de preferências dos candidatos tende a ter uma influência positiva sobre o grau de satisfação dos candidatos e a taxa de alocação, visto que mais candidatos serão alocados. Já o aumento do número de vagas fornecidas também tende a ter um bom resultado para o grau de satisfação, já que com mais vagas há mais possibilidades de que os candidatos sejam alocados. Porém esse aumento pode causar uma diminuição na taxa de alocação e/ou na média da taxa de preenchimento já que podemos alcançar casos em que teremos mais vagas do que candidatos realizando a prova. Sobre a nota mínima a diminuição da mesma trará resultados positivos para todos as métricas, até certo ponto no qual irá estabilizar pois não haverá mais candidatos que podem ser alocados devido ao fato de que os outros parâmetros (vagas e lista de preferências) estão inalterados.

Para a próxima análise um programa foi criado para gerar aleatoriamente 10 casos de teste sobre os quais irei observar o comportamento das três métricas já citadas em relação à quantidade de vagas ofertadas, nota de corte média e vagas ocupadas. Na Tabela 4 estão listados os dados necessários para cada um dos 10 testes. Já na Tabela 5 estão listados os valores correspondentes a cada uma das métricas para cada teste.

Tanto a métrica de taxa de alocação quanto a de média da taxa de preenchimento se comportam de forma a depender do número de vagas fornecidas. Em todos os 10 testes aleatórios feitos, o número de vagas fornecidas é superior ao número de candidatos que realizaram a prova, dessa forma essas duas métricas vão tender a ter valores menores, pois estará sobrando vagas nas universidades. Assim, no caso de poucas pessoas realizando a prova, a diminuição da quantidade de vagas será favorável para o aumento dessas duas métricas. No caso de muitas pessoas realizando a prova, intuitivamente, o aumento da quantidade de vagas aumentará a quantidade de vagas alocadas e pode trazer aumento para essas duas métricas além de um aumento no grau de satisfação dos candidatos. A seguir está a análise os parâmetros com base nos resultados obtidos nos 10 testes:

Tabela 4: Listagem dos parâmetros gerados aleatoriamente para os 10 casos de teste.

Teste	Universidades	Candidatos	Vagas fornecidas	Nota de corte média	Vagas ocupadas
1	10	37	876	57	27
2	7	17	790	51	14
3	26	180	2277	104	49
4	14	23	617	61	21
5	27	319	1490	92	12
6	17	143	964	55	129
7	14	311	634	68	273
8	29	29	705	80	26
9	31	190	920	94	170
10	57	178	1762	98	168

Tabela 5: Análise da taxa de alocação, média da taxa de preenchimento de vagas nas universidades e do grau de satisfação dos candidatos em 10 casos de teste gerados aleatoriamente.

Teste	Taxa de Alocação	Média da taxa de preenchimento	Grau de satisfação
1	0,0308	0,1162	0,5091
2	0,0177	0,0407	0,5954
3	0,0215	0,4144	0,1004
4	0,0340	0,0825	0,6065
5	0,0081	0,1352	0,0135
6	0,1338	0,2155	0,6337
7	0,4306	0,6142	0,6047
8	0,0369	0,1357	0,5214
9	0,1848	0,2311	0,5239
10	0,0953	0,1478	0,5159

Intuitivamente, quanto menor a nota de corte das universidades maior será a taxa de preenchimento das mesmas pois mais alunos estarão dentro dos requisitos de nota desejados. O mesmo efeito pode ser observado com a diminuição da quantidade de vagas fornecidas, isso ocorre pois dado que as universidades estão ofertando menos vagas, as mesmas não terão tantas vagas sobrando, o que pode ocorrer quando a nota de corte é alta e tem-se muitas vagas. Assim, nas tabelas podemos ver que o caso em que a média da taxa de preenchimento é maior ocorre no caso 7, quando temos um dos números mais altos de candidatos realizando a prova, um dos mais baixos números de vagas ofertadas e uma nota de corte não muito alta. Neste mesmo caso de teste tivemos também a maior taxa de alocação e o maior grau de satisfação do candidato. Todas essas 3 métricas se correlacionam com a quantidade de vagas ocupadas, e para se obter um maior número de vagas ocupadas temos que tratar os outros parâmetros do problema. Como foi mostrado nos testes anteriores o aumento da lista de preferências trás bons resultados para o aumento de vagas ocupadas.

Diferentemente da análise anterior, em que fixamos alguns parâmetros e só alteramos aquele que queríamos analisar a influência, nesta foram utilizados casos de testes aleatórios, dessa forma não pode-se afirmar qual dos parâmetros está afetando as métricas no final, mas podemos dizer o que esperamos do resultado dada uma combinação desses parâmetros. Sendo assim, para ficar mais fácil de visualizar a influência da nota e da quantidade de vagas oferecidas sobre a métrica da média da taxa de preenchimento de vagas, plotei os dois gráficos de dispersão a seguir:

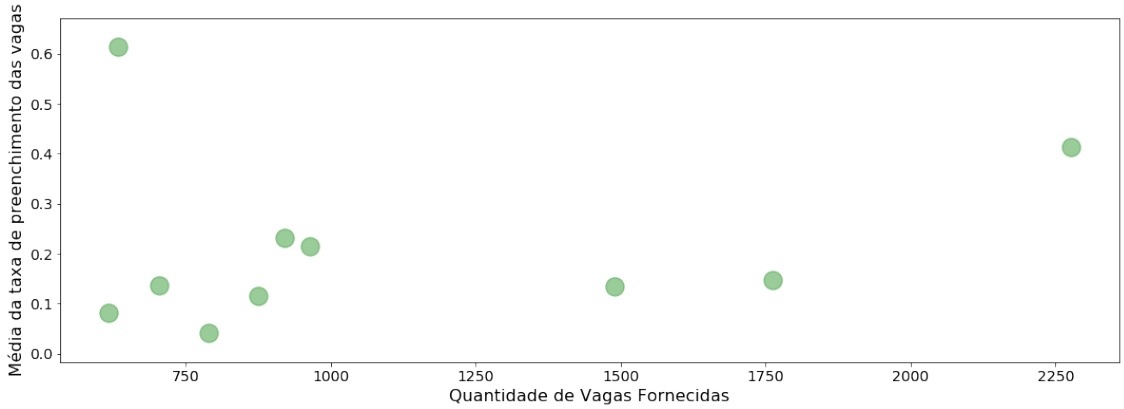

Figura 4: Gráfico de dispersão da média da taxa de preenchimentos dado a variação na quantidade de vagas fornecidas.

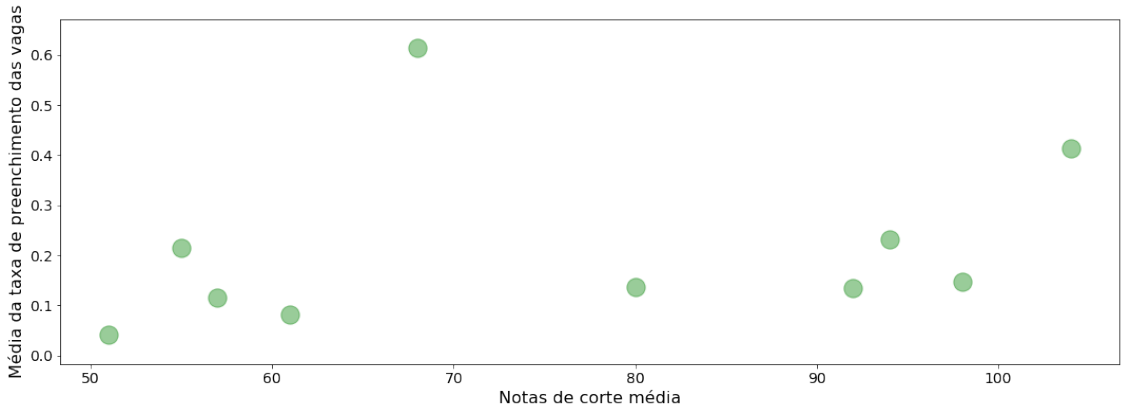

Figura 5: Gráfico de dispersão da média da taxa de preenchimentos dado a variação na nota de corte média das universidades.

A informação dada por esses gráficos é que a combinação de uma baixa quantidade de vagas fornecidas e uma nota de corte não muito alta fazem com que a média da taxa de preenchimento seja mais positiva, neste caso. Isso ocorre pois, com uma nota de corte baixa mais alunos conseguem atingir os requisitos necessários para serem alocados na universidade, ao mesmo tempo que uma quantidade baixa de vagas faz com que o preenchimento das mesmas ocorra mais facilmente, quanto menos vagas menor a chance de sobrarem muitas vagas vazias. E o teste em que tivemos maior otimização desta métrica, como dito anteriormente, foi o teste 7, o qual pode ser visto nos dois diagramas de dispersão como sendo o ponto que corresponde à média superior a 0,6.

5 Conclusão

Neste trabalho o problema Um Novo Sistema de Seleção Unificada foi resolvido com custo O(m*n), modelando o problema através do uso de listas e tabela hash e fazendo o uso dos princípios de um casamento estável, levando em conta todas as possibilidades de pares instáveis que poderiam surgir. O método de ordenar as notas de forma decrescente antes de percorrer pelos alunos foi escolhido para evitar a ocorrência de inúmeras trocas de universidade na alocação de candidatos, o que ocorreria caso eu preenchesse as universidades em ordem crescente pelo id do candidato.