Trabalho Prático 1: Um Novo Sistema de Seleção Unificada

Thiago Felicíssimo César

1 Introdução e Apresentação do Problema

Neste trabalho prático, implementa-se um algorítimo de alocação de candidatos em vagas de universidades. O algoritmo recebe duas listas, uma delas contendo os candidatos e outra contendo as universidades, e gera uma solução que atenda as restrições impostas tanto pelas universidades quando pelos candidatos. Por exemplo, cada candidato apresenta uma lista de universidades desejadas - na ordem de preferências dele -, de modo que o algoritmo não pode alocar o candidato em uma universidade que não esteja nessa lista. Da mesma forma, as universidades apresentam números de vagas finitos e notas de corte, que são fatores que devem ser respeitados na alocação. Em outras palavras, se uma universidade dispõe de um certo número de vagas, não se pode alocar nela mais alunos que vagas disponíveis, assim como não se pode alocar em uma dada universidade um aluno cuja note é menor que a de corte. Ademais, universidades não tem listas de preferências, pois para elas o único critério para saber se um aluno é preferível a outro é a nota do aluno (o critério de desempate para candidatos com a mesma nota é alocar o de menor índice).

A solução apresentada também tem a característica de ser ótima para os candidatos. Ou seja, se um determinado candidato foi alocado para a i-ésima posição na sua lista de preferências, não pode existir uma outra solução que atenda a todos os critérios impostos aqui e na qual o mesmo candidato seja alocado para a j-ésima posição da sua lista de preferências, com $j < i$. Uma última restrição imposta diz respeito a estabilidade da solução. Mais especificamente, a solução atende às condições a seguir.

1. Dois estudantes não podem desejar trocar de universidades entre eles;
2. Um candidato alocado em uma determinada universidade não pode ter uma nota menor que um candidato não alocado que tenha tal universidade em sua lista de opções;
3. Se uma determinada universidade tem vagas livres, um aluno que atende a nota de corte e que tem tal universidade na lista de preferências não pode deixar de ser alocado para ela para ir para uma universidade que ele prefere menos;
4. Um candidato não pode ficar sem vaga se uma de suas opções tem uma vaga livre e ele atende a nota de corte.

2 Solução Proposta

Primeiramente, observamos que o problema atacado neste trabalho prático consiste em uma instância do Problema do Casamento Perfeito. Desta forma, já é possível notar que o algoritmo de GS (Gale-Shapley) é um candidato para resolver o problema. Todavia, é necessário precisar alguns pontos de como o algorítimo deve ser usado.

No algoritmo original de GS, o grupo 1 propõe na ordem de suas preferências para os membros do grupo 2. Sempre que um membro não pareado do grupo 2 recebe uma proposição, ele aceita e, caso posteriormente receba uma proposta melhor - ou seja, uma proposta de um membro do grupo 1 que está mais alto na sua lista de preferências -, ele descarta o seu par e passa a ser pareado com esse novo membro. Dessa forma, o membro do grupo 1 que estava pareado mas agora está sozinho volta a propor para os membros do grupo 2, do ponto de partida de onde ele parou - em outras palavras, se ele foi descartado pelo i-ésimo membro de sua lista de preferências, ele agora propõe para a $i + 1$-ésimo membro.

Notamos algumas diferenças entre o problema original e o abordado aqui. Como cada universidade tem uma nota de corte que deve ser atendida para que exista a possibilidade de o aluno ser alocado para ela, é necessário uma modificação no algoritmo para que essa condição seja atendida. Além disso, como as universidades dispõem de múltiplas vagas, ela pode aceitar múltiplos candidatos. Uma outra implicação desse fato é que, sempre que as vagas de uma universidade estiverem cheias, a universidade deve levar em consideração toda sua lista de candidatos admitidos na hora de considerar uma proposição.

Como já mencionado, a solução deve ser ótima para os candidatos. Esse requerimento é facilmente atendido pelo algorítimo de GS, uma vez que o algoritmo produz a melhor solução para o grupo que propõe. Com isso, basta que, na implementação usada, os candidatos sejam os que propõem.

3 Implementação

Agora que o algoritmo utilizado já foi discutido de forma clara em alto nível, é necessário precisar os detalhes de como ele foi implementado. Todavia, antes de falar propriamente do algoritmo e de como as decisões são tomadas, faz-se necessário discutir as estruturas de dados implementadas. Para fins de evitar repetividade, pode-se tomar como regra geral que todas as células de listas e pilhas serão sempre alocados no *heap*, com exceção das cabeças. Dessa forma, não é necessário repetir essa dado todas as vezes.

3.1 Estruturas de Dados

3.1.1 Candidatos

A Figura 1 ilustra a implementação da estrutura de dados que armazena os dados dos candidatos, no seu estado inicial - ou seja, antes da execução do algoritmo. Nela, os dois primeiros estudantes são representados, assim como parte da lista de prioridades do primeiro.

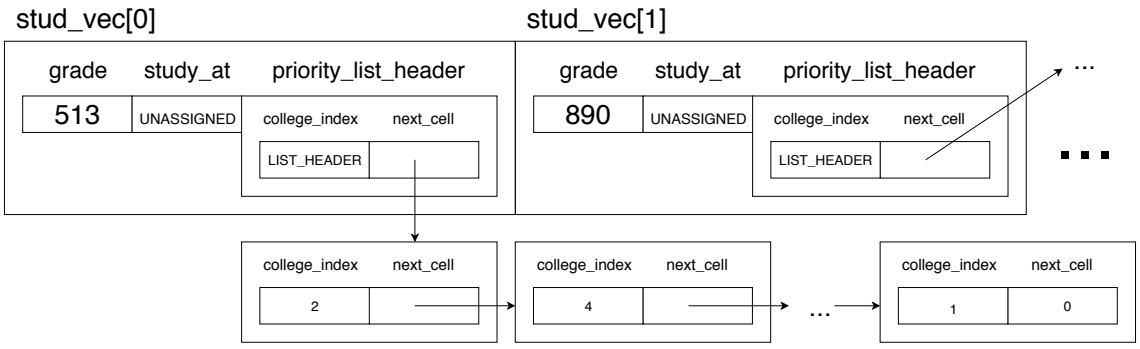

Figura 1: Estrutura de dados dos candidatos.

Como pode ser observado na imagem, os estudantes são armazenados em um arranjo (uma vez que o número de estudantes é dado e é invariante durante o código) no qual cada posição tem um *struct*, que contém a nota do aluno, a universidade para qual ele foi alocado e a cabeça da sua lista de prioridades (cada célula da lista, incluindo a cabeça, também é um *struct*).

A lista de prioridades é construída de maneira tal que as opções de maiores preferência estejam no topo. Dessa forma, a medida em que o algoritmo roda, os elementos da lista de prioridades são tirados sempre do topo. É interessante destacar também que, uma vez em que a lista é construída, nenhum elemento pode ser adicionado nela, ela só pode diminuir. Isso é uma consequência do fato de que, quando um estudante propõe para uma determinada universidade, essa universidade sai para sempre de sua lista. Dessa forma, o número de universidades para as quais ele pode propor sempre diminui. Como será visto, as universidades também serão armazenadas em um arranjo, o que possibilita que, para representar as universidades na lista de preferências, seja necessário apenas um inteiro.

Também é importante destacar que a variável *study_ at* vai sendo atualizada ao longo do código, podendo em um momento assumir o índice correspondente a uma universidade, mas em outro momento retornar a condição inicial de desalocação. Isso acontece pois um aluno pode ser alocado para uma universidade e depois ser desalocado para a entrada de um aluno cuja nota é maior.

3.1.2 Universidades

A Figura 2 ilustra a implementação da estrutura de dados das universidades, no estado final do algoritmo. Assim como na Figura 1, apenas os dois primeiros elementos e a lista correspondendo ao primeiro elemento são representados.

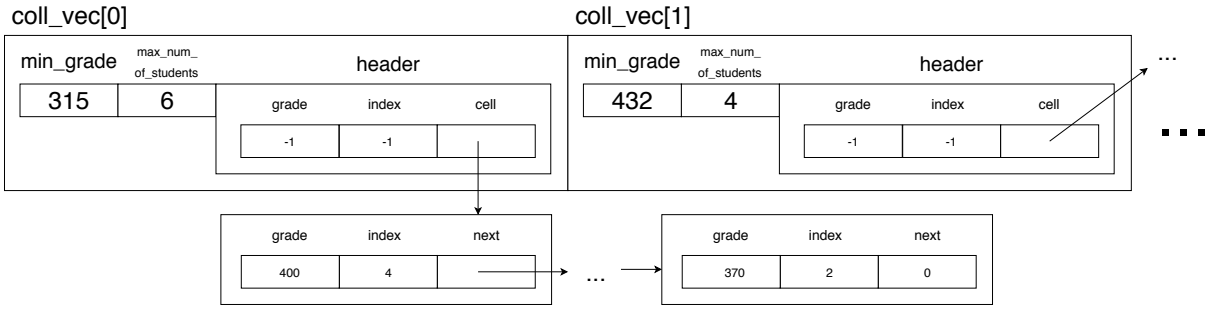

Figura 2: Estrutura de dados das universidades.

Pode ser notado que a estrutura de dados das universidades é bastante similar a estrutura dos candidatos. As universidades são guardadas em um arranjo, em que cada posição é um *struct* que guarda o número de vagas, a cabeça da lista de estudantes e a nota de corte da universidade. As células da lista de estudantes contém o índice do estudante e sua nota (é conveniente guardar também a nota de cada estudante alocado, pois esse dado é frequentemente utilizado durante o processo de decisão sobre aceitar ou não a proposta de um novo estudante).

Ao contrário da lista de preferência dos candidatos, a lista de estudantes alocados começa vazia e cresce ao longo do algorítimo, sendo seu tamanho limitado pelo número de vagas na universidade. Uma vez que a lista chega no seu tamanho máximo, novas células não podem ser criadas, e a partir desse ponto um candidato só pode ser alocado se um estudante já alocado for descartado pela universidade. Uma vez que o algoritmo de inserção não é tão trivial, ele será discutido de forma mais detalhada aqui. Para isso, o Algoritmo 1, que realiza essa operação, é exibido a seguir.

Algorithm 1 Inserção de estudantes na lista da universidade

para *i = 1, 2, ... numero_de_vagas* **faça**
  se *a i-ésima célula não existe* **então**
    cria a i-ésima célula e coloca C nela
    retorna C foi alocado sem retirar ninguém
  fim
  senão se *a nota do candidato na posição i for menor que a nota de C, ou as notas forem iguais mas o índice de C for menor* **então**
    troca C com o estudante na posição i, agora o estudante que estava na posição i passa a ser o C
  fim
fim
retorna C foi rejeitado ou desalocado pela universidade

Considera-se a situação em que um candidato C propõe para uma universidade e esta precisa tomar uma decisão. Para isso, o algoritmo itera sobre a lista de candidatos. Para cada i, entre 1 e o número de vagas, o algoritmo verifica se a i-ésima célula existe e, caso ela não exista, cria-se a célula, colocando C nela. O algoritmo então finaliza e retorna uma sinalização que a alocação foi feita sem remover ninguém. Todavia, se nessa iteração a célula na posição i já existir, o algorítimo verifica se para a universidade é preferível trocar o candidato C pelo estudante correspondente a célula i. Caso positivo, os dois são trocados e o estudante que estava na posição i passa a ser o novo C.

Todavia, isso não quer dizer que o novo C será necessariamente descartado pela universidade. O algoritmo ainda continuará iterando sobre a lista, tentando achar uma posição livre para o novo C ou um estudante pior. Com isso, percebe-se que o candidato C pode mudar de identidade várias vezes durante o algoritmo. Percebe-se também que, se a iteração termina, é porque um certo candidato teve sua proposta rejeitada ou foi descartado pela universidade. Para sinalizar isso, o programa retorna a identidade de C. Com isso, o algoritmo de GS poderá utilizar essa informação para tomar uma decisão sobre o que fazer.

3.1.3 Pilha de Proposições

A Figura 3 ilustra a Pilha de Proposições. Somente a cabeça, a primeira e a última célula são representadas.

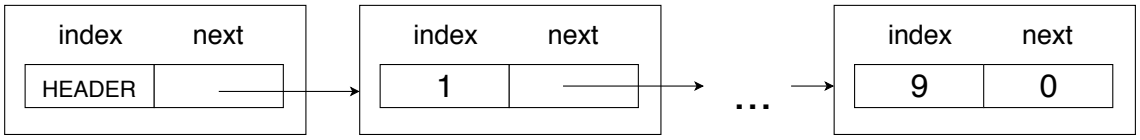

Figura 3: Pilha de Proposições.

Diferentemente das outras estruturas de dados apresentadas até agora, esta não tem como objetivo armazenar uma informação fornecida para o programa como entrada. Na verdade, a Pilha de Proposições tem como objetivo organizar quais candidatos ainda devem fazer suas propostas. Como os candidatos são indexados, somente o índice correspondente ao candidato é armazenado.

No começo do algoritmo, a pilha começa cheia com todos os candidatos e a medida em que o algoritmo vai executando os candidatos vão sendo retirados. É importante observar que um candidato pode voltar para a pilha caso ele seja desalocado de uma universidade. Sendo assim, um candidato pode sair e voltar para a pilha múltiplas vezes enquanto sua lista de preferências tiver algum universidade, mas se em um determinado ponto ele for descartado ou rejeitado pela última opção de sua lista, então ele sairá da pilha e não voltará mais, uma vez que ele não pode propor para nenhuma outra universidade.

3.2 O Algoritmo de GS

Agora que a implementação das estruturas de dados já está bem clara, entender a implementação do algoritmo de GS torna-se uma tarefa bem simples. A implementação é dada pelo Algoritmo 2 abaixo.

Algorithm 2 Implementação do algoritmo de GS

```
enquanto a pilha não estiver vazia faça
    retira a célula no topo e pega o índice i armazenado nela
    enquanto o candidato i não estiver alocado e sua lista de preferências não estiver vazia faça
        propõem para a universidade no topo da lista de preferências do candidato i
        se o candidato i foi aceito então
            atualiza as informações de i
            se algum candidato foi descartado no processo então
                atualiza as informações do candidato descartado e o insere na pilha
            fim
        fim
    fim
fim
```

O algoritmo começa retirando a célula no topo da pilha e obtendo o índice do candidato correspondente. Como a lista de preferências de cada candidato é ordenada pela preferência, o algoritmo simplesmente vai propondo para cada um das universidades na ordem da lista. Quando um candidato é aceito, suas informações são atualizadas para registrar isso (as informações da universidade são atualizadas também, mas isso é feito simultaneamente ao processo de decisão). Se no processo um outro candidato é descartado por uma universidade, as informações dele são atualizadas e ele é adicionado de volta na pilha. Mais especificamente, é a atualização da variável *study_at* que corresponde a atualização dos dados do candidato. Quando a pilha ficar vazia, todos os candidatos ou estarão alocados ou não poderão fazer mais proposições. Dessa forma, o algoritmo termina e a saída é impressa.

4 Análise de Complexidade

A análise de complexidade é feita em duas partes. Primeiramente, é feita a análise de tempo e em seguida a análise de espaço utilizado. Para fazer a análise, são levados em conta os seguintes parâmetros.

1. n, o número de estudantes;
2. m, o número de universidades;
3. k, o tamanho máximo da lista de preferência dos candidatos - ou simplesmente o tamanho, se esse número for igual para todos;
4. v, o número máximo de vagas que uma universidade pode ter.

São também apresentadas as complexidades dependendo apenas de n e m.

Por fim, é preciso fazer algumas considerações que talvez pareçam óbvias, mas que são fundamentais para a corretude dos resultados que serão obtidos. Primeiramente, assume-se que alguma universidade terá um número não nulo de vagas ($v \geq 1$) e que algum estudante terá um número não nulo de universidades em sua

lista ($k \geq 1$). Além disso, assume-se que $n \cdot k \geq m$, caso contrário a união de todas as listas de preferência nunca seria igual ao conjunto das universidades (em outras palavras, sempre existiria alguma universidade para qual nenhum estudante poderia ir).

4.1 Análise de tempo

É interessante começar essa análise pelos custos de inicialização e de finalização do programa. A inicialização de dados das universidades tem custo de $O(m)$, uma vez que o custo de inicializar uma universidade é constante (lembrando que a lista de alunos começa vazia). Todavia, ao finalizar o programa, o custo de deletar as universidades será $O(m \cdot v)$, uma vez que para cada universidade é necessário deletar todos os seus estudantes do *heap*.

A leitura dos dados dos candidatos tem custo $O(n \cdot k)$, uma vez para cada candidato a única operação que não tem custo constante é a leitura da lista de prioridades e a criação da estrutura de dados correspondente, que tem custo $O(k)$. A deleção tem o mesmo custo (não é $O(n)$, pois nem todos os candidatos estarão com a lista de prioridades totalmente vazia - no pior caso, para um candidato que foi aceito pela sua primeira opção, a deleção de sua lista custaria $O(k)$).

A criação da Pilha de Proposições terá custo de $O(n)$ - uma operação para adicionar cada estudante. A sua deleção não precisa ser considerada, pois pelo algoritmo o programa sempre termina com a pilha vazia. A impressão dos dados tem custo $O(n)$, uma vez que é impresso um conjunto de dados para cada candidato (a obtenção desses dados tem custo constante pois na *struct* de cada candidato é armazenado a universidade na qual ele foi aceito).

Indo agora para a análise da parte principal do código, é sabido que um estudante pode propor para no máximo k universidades, pois k é o tamanho máximo da lista de prioridades. Cada proposta tem um custo máximo de $O(v)$, pois no pior caso a universidade estará cheia e portanto o Algoritmo 1 executará uma iteração, que tem custo constante, para cada vaga. Como cada candidato pode propor para, no máximo, k universidades, o custo máximo das propostas de um determinado candidato é $O(v \cdot k)$ (nota-se que essa análise independe de se o candidato sair e voltar várias vezes para a pilha). Ou seja, a complexidade total da parte principal será $O(n \cdot v \cdot k)$.

Assim sendo, a complexidade total do código inteiro pode ser encontrada por meio da união das complexidades encontradas. A equação a seguir mostra como isso é feito.

$$\begin{aligned}(O(m) \cup O(m \cdot v)) \cup (O(n \cdot k) \cup O(n) \cup O(n \cdot v \cdot k)) &= O(m \cdot v) \cup O(n \cdot v \cdot k) && \text{pois } k \geq 1 \text{ e } v \geq 1 \\ &= O(n \cdot v \cdot k) && \text{pois } n \cdot k \geq m\end{aligned}$$

Caso deseje-se uma complexidade que dependa apenas de m e n, como $n \geq v$ e $m \geq k$, obtém-se que $O(n \cdot v \cdot k) \subseteq O(n^2 \cdot m)$.

4.2 Análise de espaço

Felizmente, a análise de espaço pode ser feita de maneira muito mais sucinta. Na condição inicial, a estrutura de dados dos candidatos tem complexidade de $O(n \cdot k)$, uma vez que cada estudante terá uma lista com no máximo k elementos, além de outros dados de custo de armazenamento constante. Como as listas de estudantes das universidades começam vazias, cada universidade ocupa um espaço constante no momento

inicial. Ou seja, no inicio do algoritmo, a estrutura de dados das universidades ocupa um espaço de custo $O(m)$. Por fim, a pilha de proposições começa com uma célula para cada candidato, portanto apresentando complexidade de $O(n)$.

Agora, basta notar que a cada vez que um candidato faz uma proposta, sua lista de prioridades perde uma célula. Dessa forma, mesmo que na primeira proposta ela já seja aceito, ainda assim o uso de memória não cresce, pois a nova célula criada na lista de estudantes da universidade é compensada pela célula deletada da lista de preferências. Na prática, como um estudante deve propor para várias universidades, o uso de memoria diminui, uma vez que para cada proposta uma célula de sua lista de prioridades será deletada. Dessa forma, a complexidade espacial pode ser obtida tomando a união das complexidades encontradas.

$$O(m) \cup O(n \cdot k) \cup O(n) = O(n \cdot k) \qquad \text{pois } k \geq 1 \text{ e } n \cdot k \geq m$$

Caso deseje-se uma complexidade que dependa apenas de m e n, como $m \geq k$, obtém-se que $O(n \cdot k) \subseteq O(n \cdot m)$.

5 Avaliação Estatística

Com o uso do algoritmo acima, é possível fazer uma investigação estatística de como algumas métricas da solução gerada variam de acordo com as características da entrada (os parâmetros apresentados no início da Seção 4). Para isso, são considerados três parâmetros para a análise, exibidos abaixo. A variável p_i se refere a posição da universidade na qual o candidato i foi alocado, com respeito a sua própria lista de prioridades.

$$\text{Taxa de alocação} = \frac{\text{Vagas alocadas}}{\text{Total de vagas}} \tag{1}$$

$$\text{Média de preenchimento} = \frac{1}{m} \cdot \sum_{i=1}^{m} \frac{\text{Vagas alocadas na universidade i}}{\text{Total de vagas na universidade i}} \tag{2}$$

$$\text{Taxa de satisfação} = \frac{1}{n} \cdot \sum_{i=1}^{n} \begin{cases} \frac{1}{\log_2 p_i} & \text{se o candidato i foi alocado} \\ 0 & \text{caso contrário} \end{cases} \tag{3}$$

Para a geração dos gráficos exibidos nessa seção, foram programados *scripts* para a automatização dessa tarefa. Para cada um dos pontos em um dado gráfico, são geradas 30 entradas e para cada uma delas é rodada uma versão modificada do programa desenvolvido nesse trabalho, que também calcula os parâmetros da saída gerada. Com isso, é calculado o valor médio dessas métricas para as 30 entradas. Esse processo é feito para cada ponto, variando uma determinada característica da entrada, para um total de 19 pontos, que são interpolados linearmente.

As entradas geradas para esse estudo apresentam nota de corte, número de vagas e tamanho das listas de preferências randomizados. Dessa forma, para o estudo proposto, varia-se apenas o valor máximo dessas grandezas (assim como definido na Seção 4, os parâmetros k e v são limites superiores, e não os valores das grandezas em si, que variam de acordo com os candidatos e as universidades). Observa-se que, quando não indicado, as notas de corte variam de 0 a 1000. Mais detalhes sobre a avaliação, como tabelas, bem como outros gráficos gerados, podem ser obtidos em `https://github.com/Thiagofelis/Avaliacao_Estatistica_TP1_ALG1`.

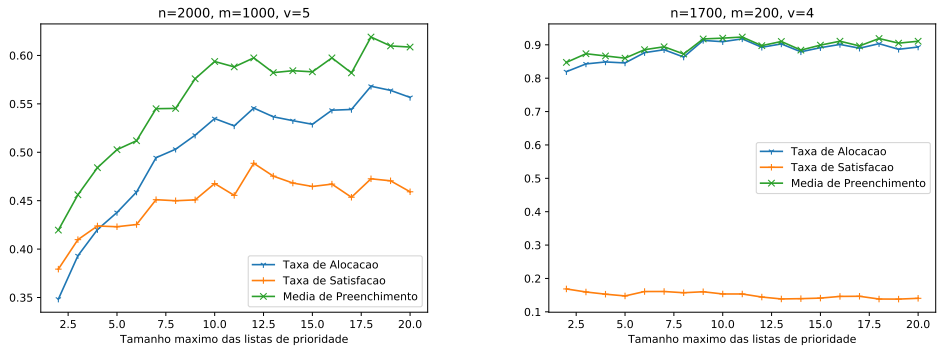

Figura 4: Impacto do tamanho máximo da lista de prioridades.

5.1 Impacto do tamanho máximo da lista de prioridades

Intuitivamente, é possível imaginar o que acontece quando os candidatos passam a poder escolher mais universidades. Um candidato que tenha mais universidades em sua lista de preferências tem menos chance de acabar desalocado, pois ele propõe para mais universidades. Isso sugere que tanto a taxa de alocação quando a média de preenchimento tenham uma tendência de crescimento positiva, uma vez que mais candidatos serão alocados e mais vagas de cada universidade serão preenchidas.

Ademais, para um determinado candidato, ter mais opções na lista de prioridades talvez se traduziria em uma maior satisfação, uma vez que para ele, ter mais opções diminui suas chances de ser rejeitado. Todavia, quanto mais opções os candidatos tiverem, mais pessoas competirão por cada vaga, o que diminui as chances de um candidato passar em uma de suas primeiras opções. Ou seja, com o aumento do tamanho da lista, alguns candidatos talvez passem a ficar mais satisfeitos por passarem a ser alocados, mas ao mesmo tempo alguns ficarão menos satisfeitos pois serão alocados para universidades piores. Isso sugere que o impacto do tamanho da lista na taxa de satisfação pode ser bem variável dependendo da situação.

O primeiro gráfico exibido na Figura 4 mostra que, quando o número total de vagas é maior que o número de candidatos, a taxa de alocação e a media de preenchimento crescem assim como previsto. Como as vagas estão em excesso com respeito ao número de candidatos, imagina-se que o impacto positivo dos estudantes que ficam mais satisfeitos por passarem a ser alocados acaba sendo maior que o impacto negativo dos estudantes que passam a ser alocados para universidades piores por falta de vaga, o que explica a tendência positiva no primeiro gráfico. Essa tendência quase não aparece no segundo gráfico (na verdade, verifica-se uma tendência suave na direção oposta, que possivelmente seja apenas ruído), no qual os estudantes são mais numerosos que as vagas.

5.2 Impacto do número máximo de vagas

Assim como feito antes com relação ao tamanho máximo da lista de prioridades, é possível fazer algumas previsões no que concerne ao impacto do crescimento do número máximo de vagas. Na medida em que as universidades oferecem mais vagas, é esperado que a taxa de satisfação cresça, pois um estudante não tão bom que quer entrar em uma determinada universidade passa a ter mais chance de entrar nela. Ao mesmo tempo, tanto a taxa de alocação quanto a média de crescimento devem apresentar tendências decrescentes, pois muito possivelmente algumas das vagas ofertadas a mais não serão preenchidas.

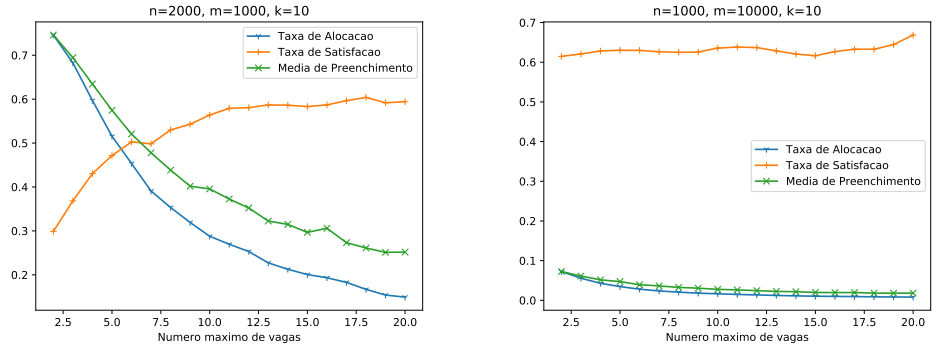

Figura 5: Impacto do número máximo de vagas.

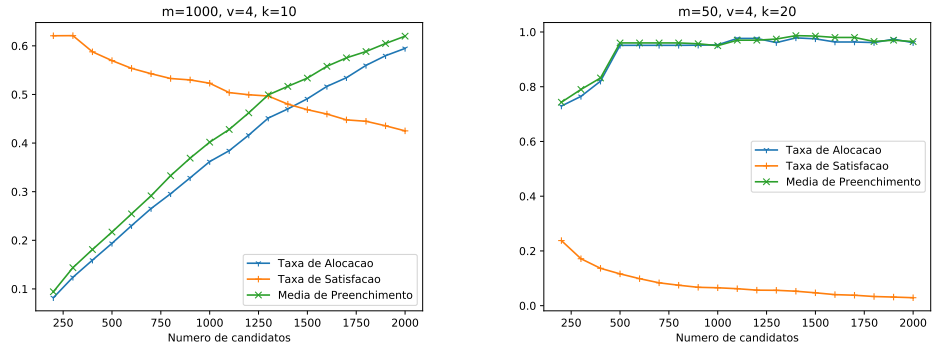

Figura 6: Impacto do número de candidatos.

De fato, como mostrado no primeiro gráfico da Figura 5, é exatamente isso que se verifica na prática. Tanto a taxa de alocação quanto a média de preenchimento decrescem a medida em que o número máximo de vagas ofertadas cresce. Por outro lado, a satisfação é crescente, confirmando a previsão do primeiro parágrafo.

Entretanto, como mostrado no segunda gráfico da Figura 5, quando o número de universidades já é muito maior que o número de alunos, a variação da oferta de vagas não produz um impacto muito alto em nenhum dos parâmetros. Isso ocorre pois os estudantes já não tem que competir muito pelas vagas, então a taxa de satisfação tende a ser alta de qualquer modo (o fato da taxa parar de crescer não quer dizer que ela tende a um, pois por mais que existam universidades em excesso, os candidatos com nota baixa ainda poderão acabar desalocados devido ao critério da nota de corte). Além disso, a maior oferta de vagas diminui tanto a taxa de alocação quanto a média de preenchimento, que de início já são tão baixas que a variação gerada acaba por ser muito suave.

5.3 Impacto do número de candidatos

O aumento do número de candidatos produz um impacto muito intuitivo no resultado. Na medida em que o número de candidatos aumenta, as universidades passam a ter mais chance de preencher suas vagas, então é esperado que tanto a taxa de alocação quando a média de preenchimento cresçam. Por outro lado, para um

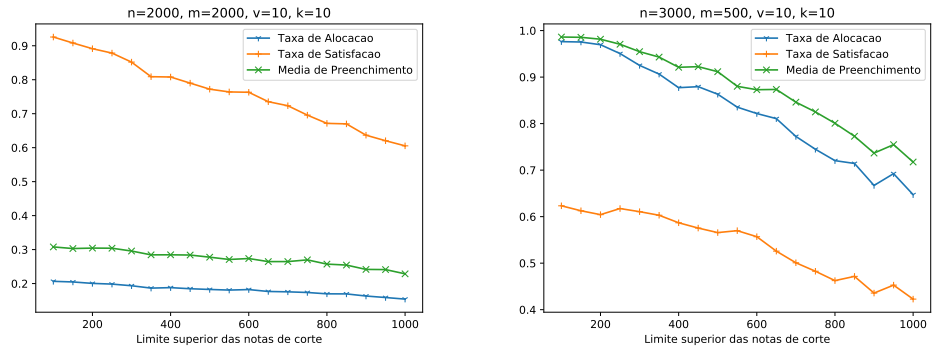

Figura 7: Impacto do limite superior das notas de corte.

determinado candidato, é melhor que o número de outros candidatos seja o menor possível, pois assim ele tem menos chance de ser rejeitado por uma universidade para a entrada de outro cuja nota é maior. Dessa forma, espera-se que a satisfação seja decrescente com respeito ao número de candidatos.

Os gráficos da Figura 6 corroboram o raciocínio dado acima. Na situação do primeiro gráfico, em que o número total de vagas é maior, percebe-se que a variação dos parâmetros é muito mais clara do que no segundo. Isso ocorre pois a partir de um certo ponto no segundo gráfico, o número de estudantes já passa a ser muito maior que o número total de vagas, o que faz com que as taxas estudadas não mudem muito. Nota-se que o número de candidatos e o número máximo de vagas são parâmetros que produzem tendências opostas na saída.

5.4 Impacto do limite superior das notas de corte

O limite superior das notas de corte também é uma característica da entrada que produz impactos muito interessantes e intuitivos na saída. Como é possível de ser prever, a medida em que universidades elevam suas notas de corte, elas correm o risco de perder futuros estudantes, ao mesmo tempo em que os candidatos passam a ter menos chances de ir para as melhores universidades de suas listas.

Isso é refletido nos dois gráficos exibidos na Figura 7. Percebe-se que a satisfação dos candidatos é muito maior quando as notas de cortes são mais baixas. O mesmo acontece com a taxa de alocação e a média de preenchimento, que apresentam uma tendência de ser maiores quando as notas de corte são menores, confirmando o argumento intuitivo apresentado no primeiro parágrafo.

6 Conclusão

O problema do casamento perfeito é extremamente geral e aparece em muitas situações práticas da vida real. Este trabalho prático permitiu que se aplicasse o raciocínio por trás da solução desse problema, que por si só é um objeto teórico e abstrato, em um projeto de um sistema que produziria impactos reais na vida das pessoas, exemplificando o reflexo prático dos problemas vistos em computação e suas possíveis aplicações.

O estudo estatístico realizado também seria de grande interesse prático na eventual implementação de um novo SISU, uma vez que permite avaliar como as alocações variam de acordo com a características da entrada.