Relatório Parcial do Projeto de Supercomputação

Thiago Hampl de Pierri Rocha

Introdução

Ao se deparar com um problema, algumas questões são intuitivamente pensadas, como “Quanto tempo vou demorar para resolver?”, ou “Qual o procedimento certo para chegar a um resultado?”, no entanto sempre existiu uma certeza que, inconscientemente, guia e motiva todas as pessoas a resolverem tais problemas: existe, ao menos, uma solução. E se, por algum motivo, um problema tiver mais de uma solução? Nesse caso, o objetivo principal muda já que passa a ser importante buscar, não só uma solução possível, mas também a melhor solução que maximize alguma propriedade.

Nessa conjuntura, o fato de ter mais de uma solução aumenta bastante a complexidade da implementação de um algoritmo que tenta buscar a melhor solução para um problema principalmente porque a pergunta motivadora muda de “Qual o procedimento certo para chegar a um resultado?” para “Qual o procedimento para encontrar o melhor resultado?”. Sutilmente, a diferença das perguntas abre portas para um mundo completamente diferente de solução de problemas em que não existe certeza de que o melhor resultado encontrado pelo algoritmo é a melhor solução de todas, chamada de global, mas sim a melhor que ela foi capaz de computar a partir do que ela estudou e mapeou.

Tecnologias robustas como Redes Neurais trabalham com esses tipos de problema em que é atribuído um score para cada solução encontrada. Elas trabalham com mapeamento contínuo, em que para buscar o melhor resultado basta mapear um plano de soluções e progredir em direção ao vetor gradiente a fim de encontrar o máximo global da variável resposta no plano. No entanto, qual seria o procedimento para encontrar a melhor solução para um problema discreto, em que não é possível mapear vizinhos imediatos a uma solução e utilizar a busca pelo gradiente?

Um desses problemas é o da Maratona de Filmes. Assim como outros problemas de múltiplas soluções, ele possui um enunciado relativamente simples e intuitivo, mas por ser discreto é mais complexo encontrar a melhor solução do que para problemas contínuos de Redes Neurais. Derivado do problema da Mochila Binária, a Maratona de Filmes resume-se em passar a maior quantidade do dia assistindo filmes, sabendo que existem duas restrições principais: apenas um filme pode ser visto por vez e existe um número máximo de filmes que podem ser assistidos dentro de uma categoria. Os Inputs variam a quantidade de filmes e a quantidade de categorias que eles disponibilizam para serem processados.

Nesses tipos de problemas discretos, o único procedimento que garante a solução ótima é a força bruta: considerando um input, testar todas as possibilidades de agenda de filme e retornar a que assistir mais filme sou a que maximizar o tempo de tela (depende do que é interessante tratar como resposta principal). No entanto, a complexidade de tempo de uma heurística de força bruta escala a um ponto em que, mesmo computacionalmente, com um input de tamanho pequeno já chega a ser inviável esperar pela resposta do algoritmo e, por isso, para resolver a Maratona de Filmes foram desenvolvidas duas heurísticas: Gulosa e Gulosa Aleatória.

Desenvolvimento

A resolução do projeto pode ser dividida em três etapas de desenvolvimento de código: heurística gulosa, heurística gulosa aleatória e o interpretador em Python. As duas primeiras são, efetivamente, as implementações de procedimentos para buscar soluções, utilizando a linguagem C++. O último tem como função principal gerar resultados das variáveis respostas interessantes considerando as duas heurísticas e salvar as execuções do código em uma planilha de dados.

Como o objetivo principal do projeto é analisar as soluções que cada heurística gerou, variar os arquivos de input é vital para observar como cada implementação reage com a mudança de ambiente. Nesse caso, um arquivo de input pode variar a quantidade de filmes e a quantidade de categorias que ele oferece a uma heurística. O número de filmes começará em 5 e terminará em 10000 com incremento de 5 filmes. Já o número de categorias varia de 1 a 20 categorias, com o incremento padrão de 1. Considerando que cada possibilidade de quantidade de filmes cruza com as 20 possibilidades de categoria, o número de input total gerados foi de 40000.

//Talvez exemplo de input

Para as implementações, existem alguns trechos de código que não mudam independentemente da heurística: tratamento do input, mapeamento dos horários disponíveis e a lógica do gerador de output.

Tratamento do Input

Código-fonte:

```
    int n_filme = 0;
    int n_cat = 0;

    vector<filme> todos_filmes;
    vector<int> max_categorias;

    // Ler número de filmes e categorias do input
    cin >> n_filme >> n_cat;

    todos_filmes.reserve(n_filme);
    int this_max_cat=0;

    // Ler máximo de filmes para cada categoria e armazenar em um vetor
    for(int i = 0; i < n_cat; i++){
        cin >> this_max_cat;
        max_categorias.push_back(this_max_cat);
    }

    int this_h_inicio=0;
    int this_h_fim;
    int this_categoria=0;

    // Ler horário de início, fim e categoria de cada filme
    for(int i = 0; i < n_filme; i++){
        cin >> this_h_inicio >> this_h_fim >> this_categoria;
        if (this_h_fim<this_h_inicio){
            // Tratamento especial para os filmes de "madrugada" para que o horário
final sempre seja maior que o inicial
            this_h_fim += 24;
        }
```

```
        todos_filmes.push_back({i, this_h_inicio, this_h_fim, this_categoria,
this_h_fim - this_h_inicio});
    }
```

O Input foi processado conforme as informações são dispostas, então a primeira informação coletada é o número de filmes e o número de categorias do arquivo de texto. Com essas duas informações, foi possível fazer laços de looping que capturavam, respectivamente, os máximos para cada categoria e as características dos filmes, guardando em vetores a serem verificados depois. A duração do filme também foi armazenada (último parâmetro do struct), para facilitar gerar o tempo de tela total da solução posteriormente.

Mapeamento dos Horários Disponíveis

Código-fonte:

```
// Função que verifica se o filme pode ser assistido
bool verify_agenda(map<int, bool> agenda, filme this_filme){
    // Se iniciar e acabar no mesmo horário, bastar retornar o invertido do booleando
da posição
    if (this_filme.h_inicio == this_filme.h_fim)
        return !agenda[this_filme.h_inicio];

    // Retornar Falso se algum horário do filme ter booleano true (já foi preenchido)
    for (int i=this_filme.h_inicio; i<this_filme.h_fim; i++){
        if (agenda[i]==true){
            return false;
        }
    }
    return true;
}

// Função que preenche os horários do filme na agenda
map<int, bool> fill_agenda(map<int, bool> agenda, filme this_filme){
    // Se iniciar e acabar no mesmo horário, preencher o horário para true
    if (this_filme.h_inicio == this_filme.h_fim)
        agenda[this_filme.h_inicio] = true;
    else{
        // Preencher todos os horários para true
        for (int i=this_filme.h_inicio; i<this_filme.h_fim; i++){
            agenda[i] = true;
        }
    }

    return agenda;
}
```

Uma das restrições do problema é que apenas um filme pode ser visto por vez. Em outros termos, isso significa que não podem ter dois filmes em um mesmo horário entre os filmes assistidos. Para fazer essa verificação, foi declarado um HashMap que mapeia um inteiro para um booleano. O inteiro representa um horário do dia e o booleano se aquele horário já está preenchido por algum filme. Se um filme adicionado começar as 8 e terminar as 11, a agenda deve preencher os horários 8, 9 e 10 com true. Dessa forma, começar as 8 e terminar as 11 implica, em teoria, que o filme durou até 10:59 e um filme que começar as 11 pode ser adicionado à solução.

Dessa forma, foram criadas duas funções para gerir esse HashMap denominado de “agenda” no código, a que faz a verificação da restrição do filme e a que atualiza a agenda se o filme for adicionado à solução. A única ressalva em relação a essas funções é para os casos em que um filme começa e termina no mesmo horário. Nesse caso, com a convenção usada de que um filme utiliza do horário de início até o final menos um, foi necessária uma verificação unicamente para esses casos em que o horário de início é verificado ou preenchido.

A primeira função retorna um booleano que, se true, significa que os horários do filme não estão preenchidos no Map e o filme pode ser incluído. Para os filmes que começam e terminam no mesmo horário, basta inverter o booleano do horário de início. Se ele for false, quer dizer que não há nenhum filme naquele horário e, por isso, a função retorna true. Se o Map do horário estiver com true, quer dizer que algum filme já preencheu o horário e, por isso, a função deve retornar false. Já para todos os outros filmes com duração maior que zero, é necessário verificar todos os horários até o final menos um e, se houver ao menos um horário que possua valor true, o filme não pode ser adicionado. A lógica da segunda função é análoga à primeira, mas com o objetivo agora de substituir os horários do filme por true na agenda.

Gerador de Output

Código-fonte:

```
// Função responsável por receber os Filmes Escolhidos e gerar um txt com as
variáveis resposta
void generateOutput(vector<filme> mochila, int n_filme, int n_cat){
    ofstream outputFile;

outputFile.open("../outputs/output_gul_"+to_string(n_filme)+"_"+to_string(n_cat)); //
nome do arquivo a partir do input
    outputFile << n_filme << " " << n_cat << endl;

    double tempo_tela=0;
    int n_mochila = 0;

    // Contando filmes dentro do vetor
    for(auto& this_film : mochila){
        tempo_tela +=this_film.duracao;
        n_mochila += 1;
    }

    double media = tempo_tela/(double)n_mochila; // Calculando média de tempos dos
filmes assistidos
    outputFile << n_mochila << " " << tempo_tela << " " << media << endl; //
Escrevendo as variáveis no arquivo
    outputFile.close();
}
```

As variáveis respostas que devem ser extraídas de uma execução são: quantidade de filmes assistidos, tempo de tela total que os filmes preencheram na agenda e a média da duração dos filmes. Dessa forma, a função generateOutput tem como objetivo escrever um arquivo de texto dessas variáveis resposta para cada execução das heurísticas (para cada input). Dessa forma, como foram gerados 40000 inputs, na execução das duas heurísticas devem ser gerados 80000 arquivos de output.

Heurística Gulosa

Código-fonte:

```
//ordenar
    sort(todos_filmes.begin(), todos_filmes.end(), heuristica_por_final);

    vector<filme> mochila;
    map<int, bool> agenda; // O valor padrão do booleano é false, por isso basta
declarar a variável

    for(auto& this_filme : todos_filmes){

        if (verify_agenda(agenda, this_filme) &&
max_categorias[this_filme.categoria-1] - 1 >= 0){
            max_categorias[this_filme.categoria-1] -= 1;
            agenda = fill_agenda(agenda, this_filme);
            mochila.push_back(this_filme);

        }

    }
    //output_visual(mochila);
    generateOutput(mochila, n_filme, n_cat);

    cout << '\n';
```

Ambas as heurísticas possuem estrutura similar, onde o vetor de filmes é ordenado a partir do horário fina ldos filmes e existe um looping principal responsável por preencher o vetor “mochila”, dos filmes escolhidos para a solução. Na gulosa, definida acima, basta verificar se o filme passa pelas restrições do problema (os horários do filme estão disponíveis na agenda e o máximo da categoria do filme não foi atingido). Como no input as categorias começam em 1, é necessário subtrair 1 no index do vetor, para que a categoria 1 corresponda à primeira posição do vetor. Por fim, se um filme for adicionado à mochila, dois parâmetros devem ser atualizados: o vetor de máximo de filmes por categoria e a agenda de filmes.

Profiling Valgrind:

-- User-annotated source: filmes_guloso.cpp

Ir

-- line 29 ----------

```
. struct filme {
.       int id;
.       int h_inicio;
.       int h_fim;
```

```
  .       int categoria;
  .       int duracao;
  . };
  .
 27  bool heuristica_por_final(filme a, filme b){
 81       return a.h_fim < b.h_fim; // ordenando pelo horário de término do filme
 27  }
  .
   .  // Função responsável por receber os Filmes Escolhidos e gerar um txt com as
variáveis resposta
 14  void generateOutput(vector<filme> mochila, int n_filme, int n_cat){
  .       ofstream outputFile;
  .
outputFile.open("../outputs/output_gul_"+to_string(n_filme)+"_"+to_string(n_cat));       //
nome do arquivo a partir do input
  8       outputFile << n_filme << " " << n_cat << endl;
1,596  => ???:0x000000000010a4d0 (2x)
  .
  3       double tempo_tela=0;
  .       int n_mochila = 0;
  .
  .       // Contando filmes dentro do vetor
 14       for(auto& this_film : mochila){
  8       tempo_tela +=this_film.duracao;
  .       n_mochila += 1;
  .       }
  .
  3       double media = tempo_tela/(double)n_mochila; // Calculando média de tempos
dos filmes assistidos
  3       outputFile << n_mochila << " " << tempo_tela << " " << media << endl; //
Escrevendo as variáveis no arquivo
 84  => ???:0x000000000010a4d0 (1x)
  .       outputFile.close();
 11  }
```

```
 .
 .  void output_visual(vector<filme> mochila){
 .        cout<<"\n\n\n";
 .
 .        cout << "|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|" << endl;
 .        cout << "|00\t01\t02\t03\t04\t05\t06\t07\t08\t09\t10\t11\t12\t13\t14\t15\t16\t17\t18\t19\t20\t21\t22\t23\t|" << endl;
 .        int agora = 0;
 .        int numero_de_espacos = 0;
-- line 67 ----------------------------------------
-- line 107 ----------------------------------------
 .        cout << "|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|" << endl;
 .
 .        for(auto& this_film : mochila){
 .        cout << this_film.h_inicio << " " << this_film.h_fim << " " << this_film.id << '\n';
 .        }
 .  }
 .
 .  // Função que verifica se o filme pode ser assistido
90  bool verify_agenda(map<int, bool> agenda, filme this_filme){
 .        // Se iniciar e acabar no mesmo horário, bastar retornar o invertido do booleando da posição
40        if (this_filme.h_inicio == this_filme.h_fim)
 .        return !agenda[this_filme.h_inicio];
 .
 .        // Retornar Falso se algum horário do filme ter booleano true (já foi preenchido)
80        for (int i=this_filme.h_inicio; i<this_filme.h_fim; i++){
28        if (agenda[i]==true){
12               return false;
 .        }
```

```
.       }
4       return true;
80  }
.
.   // Função que preenche os horários do filme na agenda
24  map<int, bool> fill_agenda(map<int, bool> agenda, filme this_filme){
.       // Se iniciar e acabar no mesmo horário, preencher o horário para true
8       if (this_filme.h_inicio == this_filme.h_fim)
.       agenda[this_filme.h_inicio] = true;
.       else{
.       // Preencher todos os horários para true
18      for (int i=this_filme.h_inicio; i<this_filme.h_fim; i++){
3               agenda[i] = true;
.       }
.       }
.
.       return agenda;
20  }
.
.   //
12  int main() {
.
1       int n_filme = 0;
1       int n_cat = 0;
.
.       vector<filme> todos_filmes;
.       vector<int> max_categorias;
.
.       // Ler número de filmes e categorias do input
6       cin >> n_filme >> n_cat;
7,726   => ???:0x000000000010a330 (2x)
.
```

```
  1      todos_filmes.reserve(n_filme);
  1      int this_max_cat=0;
  .
  .      // Ler máximo de filmes para cada categoria e armazenar em um vetor
 18      for(int i = 0; i < n_cat; i++){
 12      cin >> this_max_cat;
3,992  => ???:0x000000000010a330 (4x)
  .      max_categorias.push_back(this_max_cat);
  .      }
  .
  1      int this_h_inicio=0;
  .      int this_h_fim;
  1      int this_categoria=0;
  .
  .      // Ler horário de início, fim e categoria de cada filme
 67      for(int i = 0; i < n_filme; i++){
 95      cin >> this_h_inicio >> this_h_fim >> this_categoria;
35,745  => ???:0x000000000010a330 (30x)
 40      if (this_h_fim<this_h_inicio){
  .              // Tratamento especial para os filmes de "madrugada" para que o horário final sempre seja maior que o inicial
  .              this_h_fim += 24;
  .      }
 40      todos_filmes.push_back({i, this_h_inicio, this_h_fim, this_categoria, this_h_fim - this_h_inicio});
  .      }
  .
  .      //ordenar
  .      sort(todos_filmes.begin(), todos_filmes.end(), heuristica_por_final);
  .
  .      vector<filme> mochila;
  .      map<int, bool> agenda; // O valor padrão do booleano é false, por isso basta declarar a variável
```

```
 .
 39     for(auto& this_filme : todos_filmes){
 .
158             if (verify_agenda(agenda, this_filme) && max_categorias[this_filme.categoria-1] - 1 >= 0){
2,554       => filmes_guloso.cpp:verify_agenda(std::map<int, bool, std::less<int>, std::allocator<std::pair<int const, bool> > >, filme) (10x)
 8                  max_categorias[this_filme.categoria-1] -= 1;
 18                 agenda = fill_agenda(agenda, this_filme);
602         => filmes_guloso.cpp:fill_agenda(std::map<int, bool, std::less<int>, std::allocator<std::pair<int const, bool> > >, filme) (2x)
 .                  mochila.push_back(this_filme);
 .
 .      }
 .
 .      }
 .      //ordenando para imprimir
 .      //sort(mochila.begin(), mochila.end(), [](auto& i, auto&j){return i.id < j.id;});
 .
 .      //output_visual(mochila);
 6      generateOutput(mochila, n_filme, n_cat);
21,113   => filmes_guloso.cpp:generateOutput(std::vector<filme, std::allocator<filme> >, int, int) (1x)
 .
 .      cout << '\n';
 .      return 0;
 15 }
  .  // gerar arquivo de output com n_filmes, tempo de tela, tempo de execução, média da duração dos filmes,
 .
 .  // Formato
 .  // n_filmes_input   n_cat_input
 .  // tempo_de_tela   média_filmes
 .
```

Ir

1,146 events annotated

Os pontos que consomem mais memória são nas leituras do input e na chamada da verificação da agenda, sendo que este último pode ser alvo de otimização a partir de outro tipo de verificação da restrição. Além disso, a função que gera o output tem o pior uso de memória.

Heurística Gulosa Aleatória

Código-fonte:

```
    //ordenar
    sort(todos_filmes.begin(), todos_filmes.end(), heuristica_por_final);

    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
    default_random_engine generator(seed);
    uniform_int_distribution<int> distribution(1,4);

    int i = 1;
    int n_filme_copy = n_filme;

    for(auto& this_filme : todos_filmes){
        int prob = distribution(generator); // gera número

        if (verify_agenda(agenda, this_filme) &&
max_categorias[this_filme.categoria-1] - 1 >= 0){
            max_categorias[this_filme.categoria-1] -= 1;
            agenda = fill_agenda(agenda, this_filme);
            mochila.push_back(this_filme);

        }
        if (prob==4 && i<n_filme_copy){
            uniform_int_distribution<int> distribution_id(i, n_filme_copy-1);
            int id_aleat = distribution_id(generator);
            filme filme_aleat = todos_filmes[id_aleat];

            if (verify_agenda(agenda, filme_aleat) &&
max_categorias[filme_aleat.categoria-1] - 1 >= 0){
                max_categorias[filme_aleat.categoria-1] -= 1;
                agenda = fill_agenda(agenda, filme_aleat);
                mochila.push_back(filme_aleat);
                todos_filmes.erase(todos_filmes.begin()+id_aleat-1);
                n_filme_copy = n_filme_copy - 1;

            }

        }
        i++;

    }
```

```
    //output_visual(mochila);
    generateOuput(mochila, n_filme, n_cat);
```

A heurística Gulosa aleatória deriva da anterior, em que a lógica inicial é muito similar. A diferença é que, além de verificar o filme sendo iterado pelo laço principal, deve haver uma chance de 25% para que um filme aleatório seja escolhido e adicionado à mochila. Para isso, é sorteado um valor de 1 a 4 e, se o valor for 4, o código deve executar a particularidade do código aleatório.

Algumas variáveis são importantes para controlar o vetor de possibilidades de filmes que podem ser sorteados. O inteiro i é um contador de iterações mais um. Ele serve como valor de início distribuição uniforme que sorteia o id do filme a ser adicionado. Coo há sempre um filme sendo analisado antes do bloco aleatório do código, o sorteio pode acontecer com esse incremento de um. Além disso, foi feita uma cópia da quantidade de filmes totais do input para ser manipulada e subtraída quando um filme for escolhido. Toda vez que um filme aleatório é adicionado à mochila, é necessário apagá-lo do vetor de todos os filmes, para que ele não seja iterado e verificado novamente. Como o tamanho do vetor total diminui, é necessário controlar o máximo da distribuição uniforme que sorteia um id, para que ele não escolha um filme que não está mais presente.

Para o gerador de output, é importante gerar um arquivo de texto correspondente ao input e, por isso, nessa etapa o núemro de filmes total deve ser o mesmo recebido pelo input. Por isso, é importante criar a variável cópia a ser manipulada. De certa forma, para descobrir a quantidade de filmes aleatórios escolhidos, bastaria subtrair o número de filmes pela sua cópia. Como isso é uma resposta aleatória, não faria sentido estudá-la em relação aos valores de input e, por isso, não foi armazenada.

Profiling vallgrind:

```
--------------------------------------------------------------------------------
-- User-annotated source: filmes_guloso_aleat.cpp
--------------------------------------------------------------------------------
Ir


-- line 29 ----------------------------------------
 .  struct filme {
 .      int id;
 .      int h_inicio;
 .      int h_fim;
 .      int categoria;
 .      int duracao;
 .  };
 .
```

```
 27  bool heuristica_por_final(filme a, filme b){
 81      return a.h_fim < b.h_fim; // ordenando pelo horário de término do filme
 27  }
  .
  .  bool heuristica_por_duracao(filme a, filme b){
  .      return a.duracao < b.duracao; // ordenando pela duração do filme
  .  }
  .
 14  void generateOuput(vector<filme> mochila, int n_filme, int n_cat){
  .      ofstream outputFile;
  .
outputFile.open("../outputs/output_aleat_"+to_string(n_filme)+"_"+to_string(n_cat));
  8      outputFile << n_filme << " " << n_cat << endl;
1,596  => ???:0x000000000010a4f0 (2x)
  .
  3      double tempo_tela=0;
  .
  .      int n_mochila = 0;
 14      for(auto& this_film : mochila){
  8      tempo_tela +=this_film.duracao;
  .      n_mochila += 1;
  .      }
  3      double media = tempo_tela/(double)n_mochila;
  3      outputFile << n_mochila << " " << tempo_tela << " " << media << endl;
 84  => ???:0x000000000010a4f0 (1x)
  .      outputFile.close();
  .
 11  }
  .
  .  void output_visual(vector<filme> mochila){
  .      cout<<"\n\n\n";
  .
```

```
 .      cout                                                                                                                                                        <<
"|--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------|" << endl;
 .      cout                                                                                                                                                        <<
"|00\t01\t02\t03\t04\t05\t06\t07\t08\t09\t10\t11\t12\t13\t14\t15\t16\t17\t18\t19\t20\t21\t22
\t23\t|" << endl;
 .      int agora = 0;
 .      int numero_de_espacos = 0;
-- line 69 ---------------------------------------
-- line 108 ---------------------------------------
 .      }
 .      cout                                                                                                                                                        <<
"|--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------|" << endl;
 .
 .      for(auto& this_film : mochila){
 .      cout << this_film.h_inicio << " " << this_film.h_fim << " " << this_film.id << '\n';
 .      }
 .  }
 .
135  bool verify_agenda(map<int, bool> agenda, filme this_filme){
 60     if (this_filme.h_inicio == this_filme.h_fim)
 .      return !agenda[this_filme.h_inicio];
 .
115     for (int i=this_filme.h_inicio; i<this_filme.h_fim; i++){
 42     if (agenda[i]==true){
 20             return false;
 .      }
 .      }
 5      return true;
120  }
 .
 24  map<int, bool> fill_agenda(map<int, bool> agenda, filme this_filme){
 8      if (this_filme.h_inicio == this_filme.h_fim)
```

```
.       agenda[this_filme.h_inicio] = true;
.       else{
18      for (int i=this_filme.h_inicio; i<this_filme.h_fim; i++){
3               agenda[i] = true;
.       }
.       }
.       map<int, bool>::iterator it;
.
.       return agenda;
20  }
.
12  int main() {
.
.       map<int, bool> agenda;
.
1       int n_filme = 0;
1       int n_cat = 0;
.       vector<filme> mochila;
.       vector<filme> todos_filmes;
.       vector<int> max_categorias;
.
9       cin >> n_filme >> n_cat;
7,726   => ???:0x000000000010a350 (2x)
.
1       todos_filmes.reserve(n_filme);
1       int this_max_cat=0;
19      for(int i = 0; i < n_cat; i++){
16      cin >> this_max_cat;
3,992   => ???:0x000000000010a350 (4x)
.       max_categorias.push_back(this_max_cat);
.       }
.
```

```
  1       int this_h_inicio=0;
  .       int this_h_fim;
  1       int this_categoria=0;
  .
 67       for(int i = 0; i < n_filme; i++){
120                cin >> this_h_inicio >> this_h_fim >> this_categoria;
35,745  => ???:0x000000000010a350 (30x)
 40       if (this_h_fim<this_h_inicio){
  .                this_h_fim += 24;
  .       }
 70       todos_filmes.push_back({i, this_h_inicio, this_h_fim, this_categoria, this_h_fim -
this_h_inicio});
  .       }
  .
  .       //ordenar
  .       sort(todos_filmes.begin(), todos_filmes.end(), heuristica_por_final);
  .
  1       unsigned seed = chrono::system_clock::now().time_since_epoch().count();
937  => ???:0x000000000010a2c0 (1x)
  1       default_random_engine generator(seed);
  .       uniform_int_distribution<int> distribution(1,4);
  .
  5       int i = 1;
  2       int n_filme_copy = n_filme;
  .
 25       for(auto& this_filme : todos_filmes){
  .       int prob = distribution(generator); // gera número
  .
158                if (verify_agenda(agenda, this_filme) &&
max_categorias[this_filme.categoria-1] - 1 >= 0){
2,407    =>  filmes_guloso_aleat.cpp:verify_agenda(std::map<int, bool, std::less<int>,
std::allocator<std::pair<int const, bool> > >, filme) (10x)
  8                max_categorias[this_filme.categoria-1] -= 1;
```

```
 18                 agenda = fill_agenda(agenda, this_filme);
602        =>    filmes_guloso_aleat.cpp:fill_agenda(std::map<int,    bool,    std::less<int>,
std::allocator<std::pair<int const, bool> > >, filme) (2x)
  .                mochila.push_back(this_filme);
  .
  .        }
 40        if (prob==4 && i<n_filme_copy){
 10                 uniform_int_distribution<int> distribution_id(i, n_filme_copy-1);
  .                 int id_aleat = distribution_id(generator);
 30                 filme filme_aleat = todos_filmes[id_aleat];
  .
 82                 if              (verify_agenda(agenda,              filme_aleat)              &&
max_categorias[filme_aleat.categoria-1] - 1 >= 0){
883        =>    filmes_guloso_aleat.cpp:verify_agenda(std::map<int,    bool,    std::less<int>,
std::allocator<std::pair<int const, bool> > >, filme) (5x)
  .                max_categorias[filme_aleat.categoria-1] -= 1;
  .                agenda = fill_agenda(agenda, filme_aleat);
  .                mochila.push_back(filme_aleat);
  .                todos_filmes.erase(todos_filmes.begin()+id_aleat-1);
  .                n_filme_copy = n_filme_copy - 1;
  .
  .                }
  .
  .        }
 20        i++;
  .
  .        }
  .
  .        //output_visual(mochila);
  5        generateOuput(mochila, n_filme, n_cat);
21,113                 =>          filmes_guloso_aleat.cpp:generateOuput(std::vector<filme,
std::allocator<filme> >, int, int) (1x)
  .
  .        cout << '\n';
```

. return 0;

15 }

. // gerar arquivo de output com n_filmes, tempo de tela, tempo de execução, média da duração dos filmes,

.

. // Formato

. // n_filmes_input n_cat_input

. // tempo_de_tela média_filmes

.

Ir

1,548 events annotated

O uso de memória é bem semelhante, sendo ruim em leituras de variáveis do input, a chamada da verificação da agenda e na chamada do gerador de output. Além disso, a criação da seed a partir do tempo gera uma complexidade um pouco alta, beirando os 1000 irs. Para esse projeto em específico, não era necessário que um mesmo código e ijput gerassem dois outputs diferentes, então uma possível otimização seria trocar a geraçã oda seed pelo tempo de execuçã opor uma constante.

Interpretador Python

O código em Python foi separado em dois arquivos: o que roda as heurísticas e gera uma base de dados normalizada e o que interpreta a base e gera os gráficos resposta.

Código-fonte generate_csv.py:

```
import pandas as pd
import subprocess
import time
from os import system
from os import listdir
from os.path import isfile, join
import matplotlib.pyplot as plt
from natsort import os_sorted

# roda os arquivos, cria um pandas dataframe com n_filme do input, n_categoria e o
tempo de execucao

def run_inputs(dataframe_geral, heuristica, ex):
    input_path = '/home/user/SupercompProjeto/inputs'
    input_files = os_sorted([f for f in listdir(input_path) if
isfile(join(input_path, f))])
```

```
    for input_f in input_files:
        system('clear')
        print(f'> Running Inputs For {heuristica}')
        print(f'> {input_f}')
        n_filmes = int(input_f.split('_')[1])
        n_cat = int(input_f.split('_')[2])

        with open(f'{input_path}/{input_f}') as f:
            start = time.perf_counter()
            proc = subprocess.run([ex], input=f.read(), text=True,
capture_output=True)
            end = time.perf_counter()

        this_line = pd.DataFrame({'heur':[heuristica],
                                  'n_filmes':[n_filmes],
                                  'n_categorias':[n_cat],
                                  'tempo_ex':[end-start],
                                  'n_assistidos':[None],
                                  'tempo_tela':[None],
                                  'media':[None]})

        dataframe_geral = pd.concat([dataframe_geral, this_line])


    return dataframe_geral


# le os outputs e adiciona as 3 variáveis às linhas correspondentes de
filme_categoria

def process_outputs(dataframe_geral):
    output_path = '/home/user/SupercompProjeto/outputs'
    output_files = os_sorted([f for f in listdir(output_path) if
isfile(join(output_path, f))])

    for output_f in output_files:
        system('clear')
        print(f'> Processing Outputs')
        print(f'> {output_f}')
        heuristica = output_f.split('_')[1]
        n_filmes = int(output_f.split('_')[2])
        n_cat = int(output_f.split('_')[3])

        with open(f'{output_path}/{output_f}') as f:
            file_lines = f.readlines()

        n_assistidos = int(file_lines[1].split(' ')[0])
        tempo_tela = int(file_lines[1].split(' ')[1])
        media = float(file_lines[1].split(' ')[2][:-1])

        dataframe_geral.loc[(dataframe_geral.heur ==
heuristica)&(dataframe_geral.n_filmes == n_filmes)&(dataframe_geral.n_categorias ==
n_cat), 'n_assistidos'] = n_assistidos
        dataframe_geral.loc[(dataframe_geral.heur ==
heuristica)&(dataframe_geral.n_filmes == n_filmes)&(dataframe_geral.n_categorias ==
n_cat), 'tempo_tela'] = tempo_tela
        dataframe_geral.loc[(dataframe_geral.heur ==
heuristica)&(dataframe_geral.n_filmes == n_filmes)&(dataframe_geral.n_categorias ==
n_cat), 'media'] = media
```

```
# main chama as primerias funcoes e depois cria um gráfico 3d para cada variavel
resposta n_filmes_assistidos, tempo_tela e média

def main():
    dataframe_geral = pd.DataFrame()

    heuristicas =
{'aleat':'/home/user/SupercompProjeto/guloso_aleatorio/filmes_guloso_aleat',
                  'gul':'/home/user/SupercompProjeto/guloso/filmes_guloso',
                  }

    for heuristica, ex in heuristicas.items():
        dataframe_geral = pd.concat([dataframe_geral, run_inputs(dataframe_geral,
heuristica, ex)])

    process_outputs(dataframe_geral)

    dataframe_geral.to_csv('all_programs.csv')

if __name__ == '__main__':
    main()
```

O código roda cada executável com todos os inputs e depois processa todos os outputs gerados para construir um dataframe em Pandas. Ele foi estruturado para que cada linha seja uma execução com um input e, por isso, como são 40000 inputs para duas heurísticas, foram gerados 80000 arquivos de output. Abaixo há um exemplo de algumas linhas da base criada:

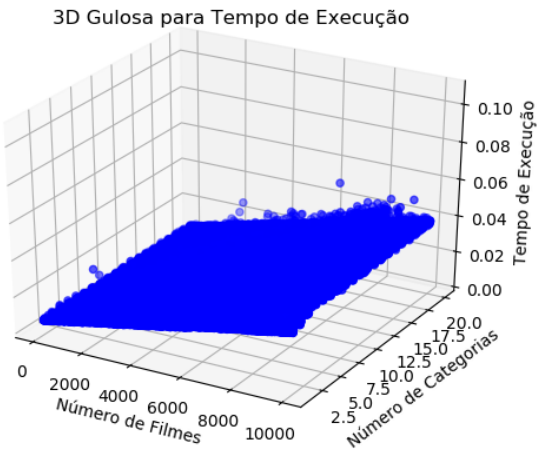
heur	n_filmes	n_categorias	tempo_ex	n_assistidos	tempo_tela	media
aleat	10000	16	0.060601942997891456	8	23	2.875
aleat	10000	17	0.06416208302835003	10	26	2.6
aleat	10000	18	0.07163409801432863	12	24	2.0
aleat	10000	19	0.060832635033875704	10	23	2.3
aleat	10000	20	0.06397613696753979	11	25	2.27273
gul	5	1	0.0061840449925512075	1	4	4.0
gul	5	2	0.006181764998473227	2	6	3.0
gul	5	3	0.008440171019174159	2	7	3.5
gul	5	4	0.008154988987371325	3	10	3.33333

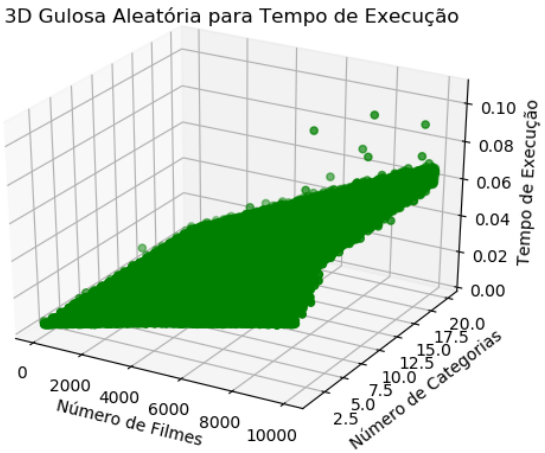
Código-fonte generate_graphs.py:

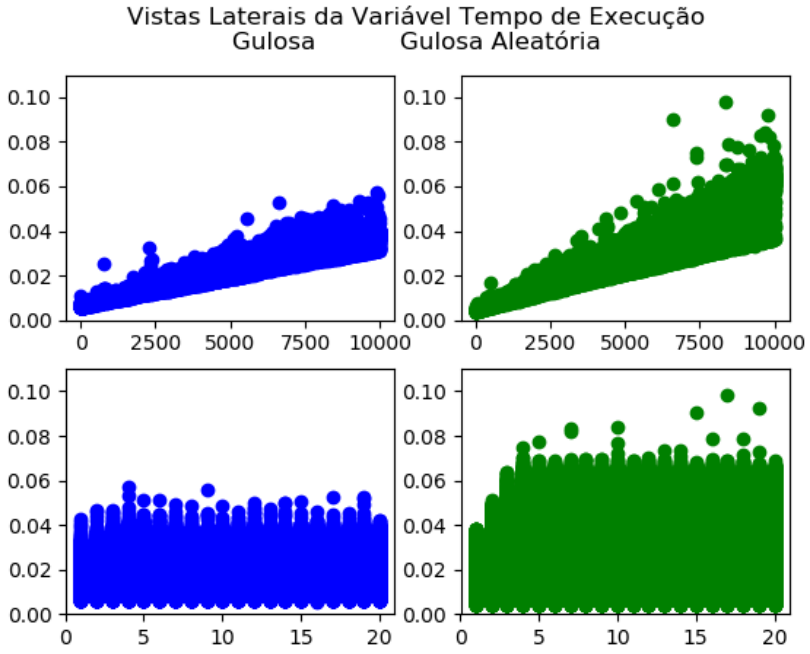
Resultados

Os resultados de cada heurísticas estão apresentados por variável resposta, em três dimensões para se observar a influência das duas variáveis de input na resposta, e depois com cortes bidimensionais para comparar valores. Para esses gráficos bidimensionais, a primeira linha do gráfico corresponde ao número de filmes e a segunda, ao número de categorias.

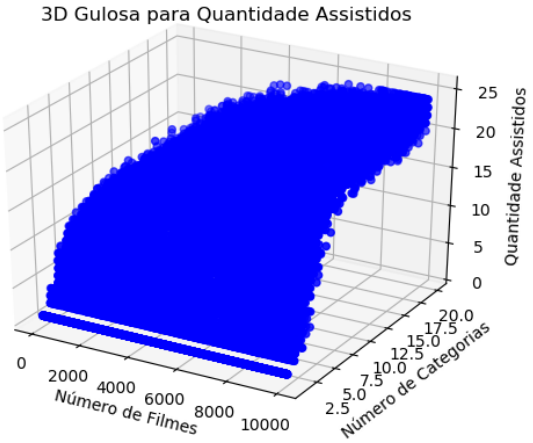
Tempo de Execução

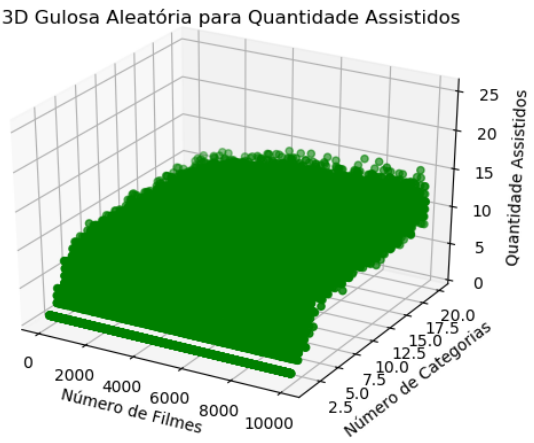


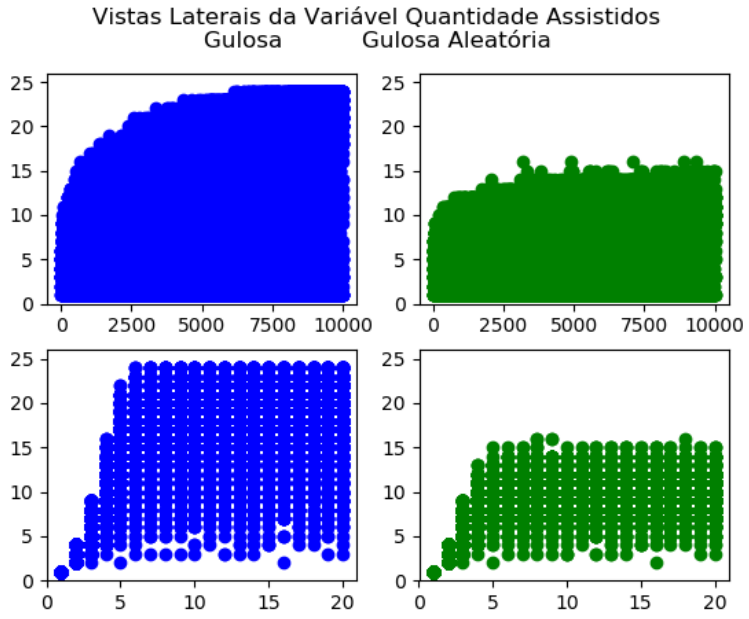




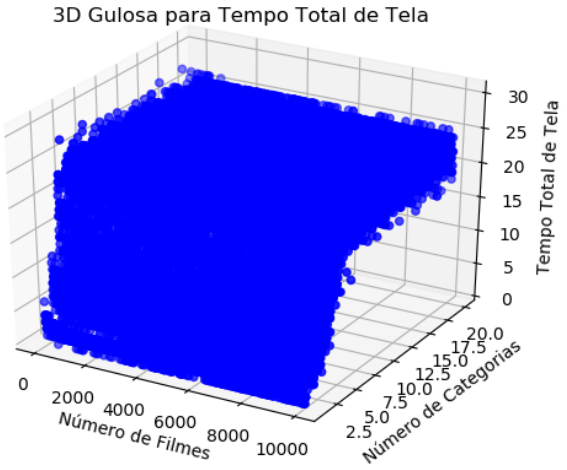
Quantidade de Filmes Assistidos

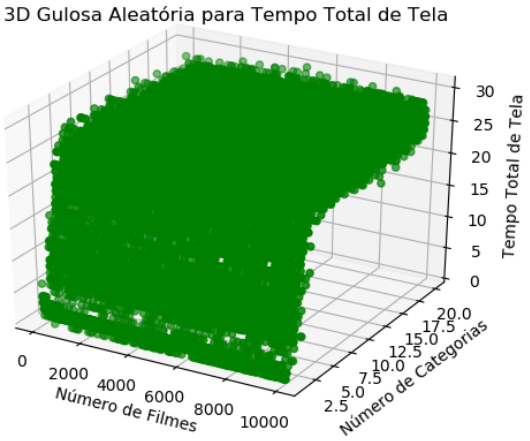


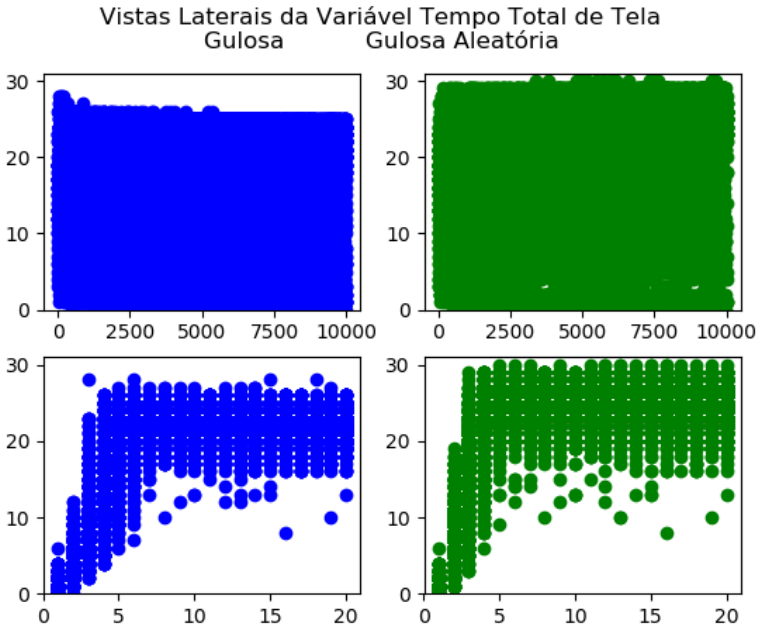




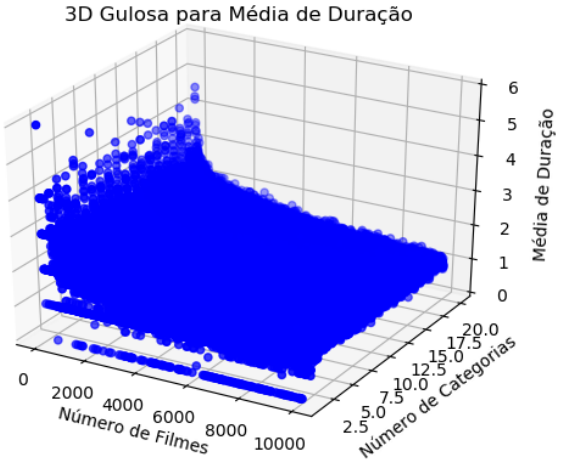
Tempo Total de Tela

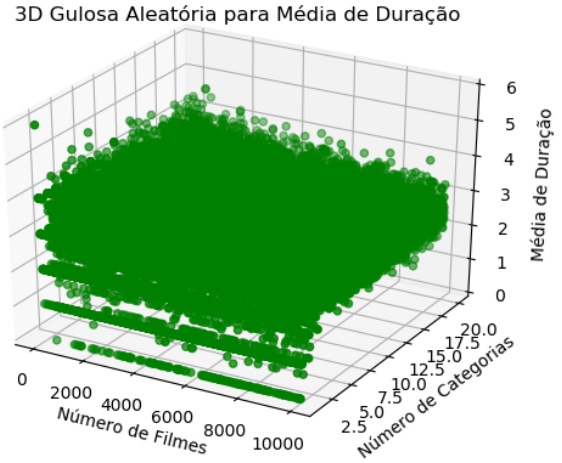


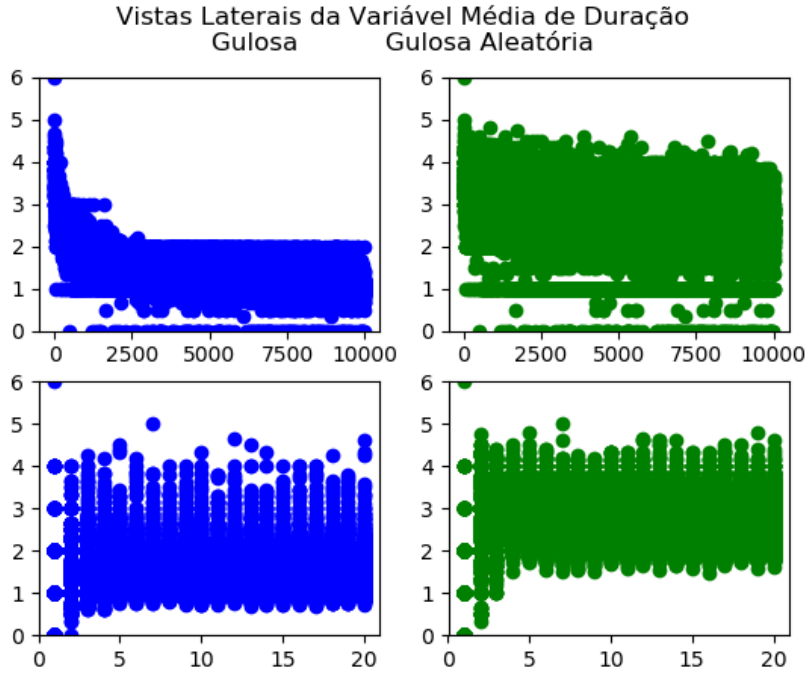




Média de Duração dos Filmes Selecionados







Conclusão

Para cada variável resposta, é possível inferir algo sobre como o input interferiu nos gráficos. O tempo de execução parece ter sido o que gerou o resultado mais diferente entre as heurísticas, sendo diretamente proporcional ao crescimento do número de filmes do input. Isso é bem justificável, já que o código da heurística aleatória é derivado da gulosa com algumas alterações dentro do looping principal. Se o código aumenta a complexidade, o tempo de execução aumenta.

Já para a quantidade de filmes assistidos, variável resposta importante para a solução, as duas curvas parece logarítmicas preenchidas, em que para valores pequenos, existe uma menor porcentagem de pontos com valor resposta alto. Para a heurística gulosa aleatória isso é menos observado, mas para a aleatória a curva é mais acentuada e, inclusive, esta possui um máximo de resposta maior, chegando a 25 filmes assistidos. O caráter de Exploration acabou diminuindo a quantidade de filmes assistidos no dia, talvez por aleatoriamente escolher filmes com maior duração que preenchem mais horários da agenda.

Já o tempo de tela total, variável resposta mais importante do problema, possuí um resultado interessante. Para ambas as heurísticas, a quantidade de filmes não afetou a variável, mas o número de categorias influencia num crescimento rápido para valores de categoria menores, com uma estabilidade entre 5 e 10 categorias. Ambas as heurísticas alcançaram resultados semelhantes de tempo de duração, mas a aleatória parece ter preenchido bem mais os maiores valores de tempo de tela, enquanto a gulosa beira bem pouco as 30 horas. Dessa forma, os melhores resultados estão em inputs com mais de 5 categorias da heurística gulosa aleatória.

Por último, a média dos filmes assistidos possui a primeira tendência negativa dos gráficos, em que quanto menor o número de filmes do input, maior a média de filmes. Isso surge, provavelmente, da maior probabilidade de o gerador de input escrever arquivos com filmes de menor duração se o arquivo é maior (já que existem mais filmes num geral, existem mais filmes com duração baixa). Como a heurística ordena os filmes por horário de término dos filmes, é difícil que um filme com uma duração muito alta consiga se encaixar na agenda, já que ele a preenche demais, dando mais espaço para as muitas com duração baixa. Essa tendência ocorre nas duas heurística, enquanto que a quantidade de categorias tenta desenhar uma curva larga logarítmica mas está tão distribuída que é difícil tirar grandes conclusões.

Num geral, a heurística aleatória demora mais pra executar porém, atinge valores maiores na variável de Tempo de Tela, que era a ser maximizada no problema. Se, na elaboração das heurísticas, o objetivo principal seria o maior número de filmes em um dia, a heurística gulosa atinge valores maiores, tendo soluções com médias de filmes menores.