Organização de um campeonato de futebol

Adrielle Andrade Carvalho
Instituto de Computação
Universidade Federal da Bahia
Salvador, Brasil
adrielle.andrade@ufba.br

Fernando Franco de Lacerda Neto
Instituto de Computação
Universidade Federal da Bahia
Salvador, Brasil
fernando.franco@ufba.br

Lucas de Azevedo de Oliveira
Instituto de Computação
Universidade Federal da Bahia
Salvador, Brasil
lucasazevedo@ufba.br

Esse projeto busca estudar e propor uma solução original para um problema NP-completo, sendo o problema escolhido a organização de juízes para jogos de futebol da Copa do Mundo. A técnica utilizada para determinar o escalonamento da agenda de trabalho baseia-se em um algoritmo guloso.

Palavras-chave: NP-completo, futebol, árbitros, algoritmo guloso.

I. Definição

Em uma partida de futebol da Copa do Mundo são necessários ao menos 5 árbitros, sendo eles:

- 1 árbitro principal, o qual apita o jogo;
- 2 árbitros assistentes, também chamados de bandeirinhas;
- 1 árbitro reserva, que deve substituir um dos três acima em caso de impossibilidade no cumprimento de sua função;
- 1 ou mais árbitros de vídeo, conhecidos como VAR, que ficam em uma área externa para revisão de lances.

Durante a fase de grupos, principalmente na terceira rodada, ocorrem jogos simultâneos e, dependendo do total de árbitros disponíveis naquele momento, pode-se tornar um desafio montar um planejamento coerente com os eventos, mesmo que não sejam incluídas questões como melhor custo e deslocamento.

Nesse sentido, o problema do escalonamento de árbitros para jogos de futebol, assim como outros problemas de escalonamento, se encontra na classe dos problemas NP-completos. Tal problema consiste em organizar os juízes escalados para cada partida, considerando seus horários de trabalho e sua disponibilidade no momento do jogo. Acrescenta-se que todos os juízes estão qualificados para todas as possíveis funções.

Instância: Inteiros *j*, *x, y, h* e n, um vetor com os *j* horários de jogos e uma n-tupla, sendo que cada um de seus elementos é um vetor de *h* intervalos nos quais cada um dos n juízes podem trabalhar.

Questão: ∃ um planejamento tal que em todos os *j* jogos tenha pelos menos 5 árbitros sendo que cada árbitro precisa participar de pelo menos *x* jogos e no máximo *y* jogos?

II. Justificativa

Com o objetivo de demonstrar o pertencimento desse problema à classe dos problemas NP-completos, deve-se reduzir um outro problema já conhecido por pertencer a essa classe ao problema novo. No caso do escalonamento de árbitros, existe um problema muito semelhante, chamado escalonamento de funcionários. Tal problema é descrito em [1], como segue:

Instância: Inteiros m e k, um conjunto C de m-tuplas, cada uma com k 1's e $m - k$ 0's, os quais representam agendas de trabalho dos funcionários, uma m-tupla de requerimentos $\overline{R}$ de inteiros não negativos e uma quantidade n de funcionários.

Questão: ∃ um planejamento $f: C \to Z_0^+$, tal que $\Sigma_{\bar{c}\epsilon C} f(\bar{c}) \leq n$ e $\Sigma_{\bar{c}\epsilon C} f(\bar{c}) \cdot \bar{c} \geq \overline{R}$?

Nesse momento, deve-se construir uma instância baseada nele para o problema dos juízes:

Instância da redução: Para cada funcionário do problema do escalonamento, cria-se cinco árbitros para o problema dos juízes com escalas de horário idênticas. Sendo o número de juízes cinco vezes o número de funcionários, e o requerimento de árbitros cinco vezes $\overline{R}$.

Resposta para o problema dos juízes é sim:

$\Sigma_{\bar{c}\epsilon C} f(\bar{c}) \leq n$ e $\Sigma_{\bar{c}\epsilon C} f(\bar{c}) \cdot \bar{c} \geq \overline{R} \Rightarrow$ ∃ planejamento tal que todos os jogos tenha pelo menos 5 árbitros, escolhendo exatamente os cinco árbitros "cópias" criados a partir do problema do escalonamento. Como o problema dos funcionários permite que existam funcionários escalados para o mesmo turno, cumpre-se também a questão de jogos simultâneos. Além disso, a quantidade mínima *x* de jogos por árbitro é o $min\{$*quantidade de funcionários em um turno*$\}$ e o *y* é o $max\{$*quantidade de funcionários em um turno*$\}$.

III. Revisão Bibliográfica

Existem diversos problemas de escalonamento de funcionários que se assemelham ao problema dos juízes. Neles, costumeiramente, há um número n de trabalhadores com diferentes horários de trabalho, os quais podem ser organizados em n-tuplas de conjuntos com seus turnos. É possível acrescentar restrições como regras da legislação trabalhista ou redução de custos. No problema dos juízes, assume-se que os custos são os mesmos independentemente da forma que os árbitros são escolhidos e todas as agendas na n-tupla já seguem a legislação.

Em [2], foi estudado o escalonamento de enfermeiros em um hospital, considerando restrições como reduzir custos, atender as preferências dos funcionários e obedecer a legislação trabalhista. Segundo o artigo, o problema do escalonamento de enfermeiros é NP-difícil, entretanto, justifica-se sua análise devido a semelhança com o problema dos juízes.

A ideia encontrada pelos autores é modelar o problema dos enfermeiros como um Problema de Atribuição Multinível e solucioná-lo através de sucessivas resoluções do Problema de Atribuição. Na resolução dos Problemas de Atribuição foi utilizada uma combinação do procedimento conhecido como shortest augmenting path e o método húngaro, um algoritmo desenvolvido por Carpaneto e Toth. A questão dos custos possui um grande peso nessa análise, tornando-o um problema focado em reduzir custos.

Dessa forma, é construída uma matriz de custos de tamanho $n \times n$, na qual cada elemento a_{ij} representa o custo de um enfermeiro i receber num dia k uma tarefa j. Ao todo devem existir n enfermeiros. Após a resolução desse Problema de Atribuição, deve-se gerar uma outra matriz de mesmo formato para o dia seguinte e, caso ocorra alguma violação, uma penalidade é aplicada. Em seguida, ocorre uma fase de melhoria, contendo cortes e recombinações das matrizes até chegar no resultado final.

O algoritmo proposto obteve resultados melhores do que aqueles encontrados na base de dados de referência, obtendo melhor eficiência em problemas de grande porte, ou seja, com tamanho mais realista. A complexidade das técnicas utilizadas é de $O(n^3)$, sendo considerado eficiente por poder ser executado em um curto período de tempo e obter resultados superiores.

Em [3], é analisado um problema bastante semelhante a [2]. Agora, busca-se encontrar um escalonamento para médicos, considerando a qualificação, respeitando regras sindicais e preferências pessoais. Tal problema é bastante complexo devido aos conflitos gerados por essas restrições, se enquadrando na classe de problemas NP-difíceis.

Dessa maneira, o procedimento adotado é dividido em duas partes, assim como em [2]. Inicialmente é proposta uma técnica para obter soluções iniciais, chamada *relax-and-fix* e, posteriormente, uma técnica de melhoria, nomeada *fix-and-optimize*.

O método *relax-and fix* tem como principal objetivo obter uma solução rápida, a qual deverá ser melhorada pelo *fix-and-optimize*. Nessa primeira etapa, é realizado um particionamento do conjunto de variáveis originais de forma repetida e esses subproblemas são resolvidos. Para isso, foi testado o particionamento por semanas, seguindo a estratégia *forward,* e o particionamento por turnos, no qual em cada iteração é escolhido um número de turnos para ser integralizado e fixado após a resolução.

Em seguida, dá-se início a fase de melhorias. O *fix-and–optimize* funciona como um algoritmo de busca local, realizando partições e fixando variáveis. Esse algoritmo chega ao fim quando o limite de tempo é alcançado ou não são encontradas melhorias após um certo número de iterações. Da mesma forma que no passo anterior, foram explorados particionamentos por semana e por turno.

Por fim, concluiu-se que o método proposto obteve resultados de qualidade melhor do que o método antigo, todavia, não houve melhoria quanto a qualidade de escalas e os limites inferiores de ambas as soluções são bastante fracos.

IV. METODOLOGIA

A técnica escolhida será baseada em algoritmos gulosos. Esse método, segundo [4], é "míope" e, portanto, faz escolhas levando em conta somente a iteração atual. Dessa forma, é possível que não seja feita a melhor escolha para um determinado problema. A escolha desse modelo de algoritmo faz sentido devido a busca realizada não ter retornado resultados que utilizassem essa técnica, apesar de poder ser considerado um problema de organização, o qual busca o melhor agendamento que atende determinados requisitos.

A otimalidade da resposta para o problema pode ser descrita como obter uma matriz que atende todas condições iniciais, logo, todos os juízes trabalham ao menos a quantidade mínima de partidas, nenhum deles ultrapassa o número máximo de jogos por juiz e todos os jogos possuem pelo menos 5 árbitros.

Por motivo de familiaridade dos autores, a linguagem escolhida para implementação do algoritmo foi C++. A ideia do algoritmo para esse trabalho foca em escolher sempre o juiz com menos horários disponíveis e que ainda precise da maior quantidade de jogos para atender ao número mínimo de partidas por juiz. Acrescenta-se que o juiz escolhido não pode ter alcançado o número máximo de jogos que cada árbitro pode participar. A escolha pode ser considerada gulosa devido a esses dados serem modificados a cada iteração, logo, deve-se se considerar apenas a corrente. Desse modo, define-se a entrada da seguinte maneira:

Entrada: n árbitros, h horários que os árbitros podem trabalhar, j horários dos jogos, x quantidade mínima de jogos por juiz, y quantidade máxima de jogos por juiz, uma matriz binária $[n \times h]$ na qual cada linha é um árbitro e cada coluna um horário que o árbitro pode participar. A matriz deve ser preenchida com 0's caso o juiz não esteja disponível nesse horário e 1's caso esteja.

É preciso também que essa entrada atenda a alguns requisitos, tais como:

- $5 \times j \geq n \times x$;
- $SL \geq x$, tal que SL representa a soma de cada linha;
- $SC \geq 5 * JH$, tal que SC representa a soma de cada coluna e JH o número de jogos naquela hora.

Nesse sentido, o primeiro passo após o recebimento das entradas deve ser a criação de uma matriz ordenada de forma crescente, baseada na quantidade de jogos disponíveis para cada árbitro, dividido pela quantidade de jogos que esse mesmo árbitro precisa participar, com o objetivo de obter uma melhor otimização do tempo.

Dessa forma, o algoritmo seleciona aquele com menor "densidade" disponível, marca que ele está escalado para determinado jogo, recalcula sua densidade, representada no novo cálculo por $\frac{quantidade\ disponível - 1}{quantidade\ para\ participar - 1}$, e o insere novamente na lista ordenada, seguindo sua nova densidade. É importante ressaltar que esse juiz não pode estar escalado para um outro jogo no mesmo horário

Esse procedimento deve ser realizado repetidamente até todos os jogos possuírem o número mínimo de juízes. Nesse momento, será finalizado o escalonamento e o algoritmo deverá retornar essa agenda, no formato de uma matriz na qual cada linha representa um jogo e nas colunas a partir da tem-se o ID de cada juiz, sendo representado por sua posição na matriz de horários.

Desse modo, o algoritmo é composto por:

- Função "sortFunction": uma função que realiza uma ordenação padrão na matriz;
- Função "densidade": uma função que calcula a matriz densidade, cujos elementos são o quociente da divisão entre quantidade de jogos que um juiz

pode participar e quantos jogos ele precisa para alcançar o número mínimo de partidas;
- Função principal: uma função na qual realiza-se a chamada para outras funções e é capaz de realizar as escolhas gulosas, montando a matriz de resultados.

V. Resultados

Realizando a análise assintótica, é possível perceber que o algoritmo possui uma complexidade assintótica $O(n \log n + j + (n \times h) + (j \times n \times n \log n))$. Nesse momento, será apresentada uma análise dos casos de teste, visando mostrar com clareza o funcionamento do código e suas limitações.

A. Caso de teste 1

Nesse caso de teste, existem condições simples, visando mostrar o funcionamento do algoritmo e a solução ótima encontrada.

Entrada: 8 juízes, que trabalham em 4 horários, 4 jogos, sendo os seus horários 0, 1, 2, 3, e cada árbitro deve participar de ao menos 2 partidas e em no máximo 3 partidas. A matriz de horários deve ser da seguinte forma:

1 1 1 0

1 0 0 1

1 1 1 0

0 1 1 0

1 0 1 1

0 1 0 1

1 1 0 1

0 0 1 1

Primeiramente, o algoritmo procura 5 juízes para formar o time de arbitragem de cada jogo, logo, escolhe-se o primeiro árbitro na lista ordenada de "densidades", tal que cada densidade é igual a $\frac{quantidade\ disponível}{quantidade\ para\ participar}$, que tem disponibilidade no horário do jogo, dando preferência para árbitros que ainda não cumpriram com o mínimo de jogos requeridos. Caso, todos os árbitros já tenham cumprido com o número de partidas mínimas, escolhe-se o primeiro árbitro disponível para aquele horário de jogo.

Nesse caso de teste, temos que o algoritmo escolhe na seguinte ordem:

Jogo 1 - Árbitros 2,1,3,5 e 7

Jogo 2 - Árbitros 4,6,1,3 e 7

Jogo 3 - Árbitros 4,8,1,3 e 5

Jogo 4 - Árbitros 2,6,8,5 e 7

No qual os árbitros são numerados de 1 a 8, e escalados na ordem da esquerda para a direita e de cima para baixo. Concluindo, portanto, a solução para esse caso.

B. Caso de teste 2

Agora, busca-se mostrar um caso que não cumpre algum dos pré-requisitos, portanto, deve falhar.

Entrada: 8 juízes, que trabalham em 4 horários distintos, 5 jogos, sendo os seus horários 0, 1, 2, 3, 1, e cada árbitro deve participar de ao menos 2 partidas e em no máximo 3 partidas. A matriz de horários deve possuir o seguinte formato:

1 1 1 0

1 0 0 1

1 1 1 0

0 1 0 0

1 0 1 1

0 1 0 1

1 1 0 1

0 0 1 1

É possível perceber que esse caso falha por 2 motivos. Em primeiro lugar, nota-se que o quarto árbitro não pode participar de dois jogos. Além disso, como existem dois jogos na hora 1, seria necessário pelo menos 10 árbitros que pudessem participar naquele horário, o que não é possível. Em termos práticos, o que garante a falha desse caso é que não existe forma de escolher cinco árbitros para o terceiro jogo, e nem escolher dois jogos para o árbitro de número quatro trabalhar.

C. Caso de teste 3

Nesse caso de teste, busca-se exibir um outro exemplo no qual ocorre uma falha.

Entrada: 6 juízes, que trabalham em 5 horários distintos, 5 jogos, sendo os seus horários 0, 1, 2, 3, 4, e cada árbitro deve participar de ao menos 2 partidas e no máximo 4 partidas. A matriz de horários deve possuir o seguinte formato:

1 1 1 1 0

1 1 1 0 1

1 1 0 1 1

1 0 1 1 1

0 1 1 1 1

1 1 1 1 1

Claramente, para existir solução nesse caso seria necessário que o último árbitro participasse de todos os 5 jogos, o que não é possível já que cada juiz pode participar de no máximo 4 partidas, então fazendo as escolhas não será possível encontrar uma combinação para o último jogo.

D. Caso de teste 4

Nesse caso de teste, mostra-se um caso que o algoritmo guloso falha, todavia, existe uma solução.

Entrada: 10 juízes, que trabalham em 5 horários distintos, 5 jogos, sendo os seus horários 0, 1, 2, 3, 4 e cada árbitro deve participar de ao menos 2 partidas e no máximo 3 partidas. A matriz de horários deve possuir o seguinte formato:

1 1 1 1 1

1 0 0 0 1

0 0 1 1 0

0 1 1 1 0

0 0 1 1 0

1 1 0 0 1

0 0 1 1 0

1 1 0 0 1

0 0 1 1 0

1 1 0 0 1

A saída do algoritmo para esse problema é:

Não existe juízes o suficiente para o jogo número 5

Jogo na hora : 0 Juizes escolhidos: 1 5 7 9 0

Jogo na hora : 1 Juízes escolhidos: 3 9 7 5 0

Jogo na hora : 2 Juízes escolhidos: 8 3 0 2 4

Jogo na hora : 3 Juízes escolhidos: 3 2 4 6 8

Jogo na hora : 4 Juízes escolhidos: 9 7 5 1 -1

Tal que o -1 significa que não foi encontrado árbitro disponível para aquele jogo. No entanto, o algoritmo falha, visto que esse caso de teste possui uma solução, sendo uma das possíveis escolhas descrita abaixo:

Jogo na hora : 0 Juizes escolhidos: 1 5 7 9 0

Jogo na hora : 1 Juízes escolhidos: 3 9 7 5 0

Jogo na hora : 2 Juízes escolhidos: 2 3 4 6 8

Jogo na hora : 3 Juízes escolhidos: 2 3 4 6 8

Jogo na hora : 4 Juízes escolhidos: 0 1 5 7 9

O algoritmo nesse caso de teste falhou por conta da escolha gulosa, como pode-se perceber o juiz 6 precisa participar dos jogos nos horários 2 e 3, e o algoritmo não o escalou, visto que não foi considerado a melhor escolha daquele momento.

VI. Conclusões

É evidente, portanto, que esse algoritmo, apesar de ter sido pensado com foco na Copa do Mundo, pode ser utilizado em outros campeonatos, como a Liga dos Campeões ou o Brasileirão. Ademais, é possível resolver outros problemas de escalonamento de funcionários, basta que os custos sejam iguais e não exista distinção entre os cargos, visto que o número de juízes por partida pode ser lido como número de funcionários por turno e o número máximo de jogos por juiz pode ser interpretado como número máximo de horas trabalhadas sem violar as leis trabalhistas.

Tendo em vista que o melhor algoritmo analisado na Revisão de Literatura possuía complexidade assintótica $O(n^3)$ e que o algoritmo proposto tem uma complexidade de $O(n \log n + j + (n \times h) + (j \times n \times n \log n))$, conclui-se que esse algoritmo é mais eficiente em determinados cenários, mais especificamente, quando o número de jogos e horários trabalhados por cada árbitro for $O(n)$. No entanto, é preciso ressaltar que a comparação não é completamente justa visto que o algoritmo apresentado em [2] busca resolver um problema semelhante, contudo, nele os custos são levados em consideração e possuem um peso elevado na determinação da escolha pelo algoritmo.

O método de ordenação, que é chamado sempre que um juiz é escolhido, utilizado foi o *sort* padrão do C++, que sempre tem complexidade assintótica de $O(n \log n)$, todavia, pensando em melhorias e otimização de código, seria possível utilizar o *sort* na primeira iteração e após isso utilizar o Insertion Sort para inserir o elemento atualizado, visto que como ele insere apenas um elemento e percorre um vetor de tamanho *n,* sua complexidade assintótica é indicada por $O(n)$.

Outrossim, é possível definir escolha ótima diferente para o código ou, possivelmente, considerar as duas escolhas ótimas como uma lista de prioridade, sendo essa nova escolha ótima encontrada através da seleção de jogos com menos opções de árbitros. Como exemplo, considerando essa nova escolha, seria possível encontrar a solução para o caso de teste D.

VII. Referências

[1] M. R. Garey, D. S. Johnson, Computers and intractability, 1st ed., United States, 1979.

[2] D. B. Rizzato, E. L. Melo, A. A. Constantino, Automação e otimização do processo de escalonamento de enfermeiros em hospital, III SIMEPRO, pp. 1-5, 2009.

[3] V. A. P. A. Devesse, Métodos de solução para o problema de escalonamento de médicos, Tese de Mestrado em Ciência da Computação e Matemática Computacional, USP, 2016.

[4] P. Feofiloff, Método guloso. Disponível em: <https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/guloso.html>. Acesso em: 26 de novembro de 2022.