Algorítimos e Estrutura de Dados III
O Problema da Viagem do Torneio

Pablo Cecilio Oliveira
Alexander Cristian

1 Introdução

O Problema da Viagem do Torneio Viajante (*Traveling Tournament Problem*) pode ser considerado como um problema de otimização matemática que envolve a criação de uma escala de jogos viável para um torneio. Essa viabilidade pode ser definida de maneira geral por dois outros problemas. O primeiro é uma questão de viabilidade, o padrão de jogos em casa e longe deve ser suficientemente variado, de modo a evitar padrões de estadia e deslocamento excessivos. O segundo é uma questão de deslocamento, onde se busca minimizar o custo total de deslocamento.

Neste trabalho prático é apresentado uma solução aproximada para o TTP do Campeonato Brasileiro de Futebol.

A escala de jogos do Campeonato Brasileiro de Futebol consiste em definir uma escala de jogos viável entre n times divididos em $n-1$ rodadas k no primeiro turno, onde cada time joga com cada um dos outros sem repetições, e sem que existam mais que três jogos consecutivos fora de casa. Para o returno na rodada $k+(n-1)$, os jogos são espelhados do primeiro turno, porem com o mando de campo invertido.

2 Implementação

Para a construção de uma solução inicial, utiliza-se o método de três fases proposto por Ribeiro e Urrutia[3], e posteriormente, é adotado o método LNS (*Large Neighborhood Search*), que utiliza de busca em profundidade para se obter melhores resultados nas soluções aproximadas.

2.1 Método de três fases para geração de uma solução inicial

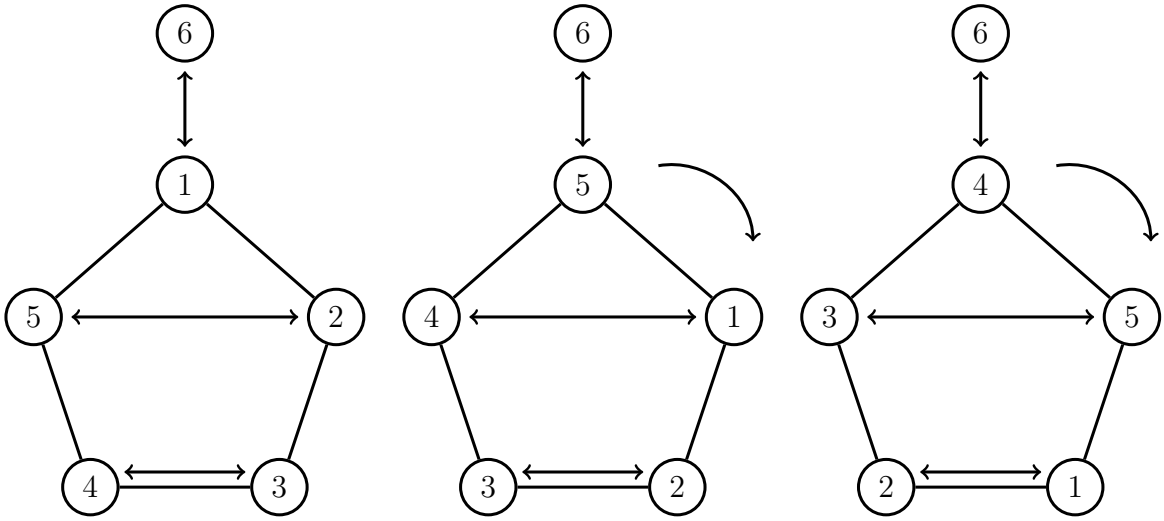
2.1.1 Primeira fase: Método do Polígono

A primeira fase da solução inicial consiste em uma fatoração que utiliza o Método do Polígono. Nesse método, sendo n o número de times participantes do campeonato, forma-se um polígono com $n-1$ nós numerados e um n-ésimo time é atribuído a um nó fora desse polígono.

A cada rodada, os times localizados nos nós $k = 2, 3, ..., n/2$ jogam contra os times localizados nos nós $n+1-k$, o n-ésimo time localizado fora do polígono joga contra o

time localizado no primeiro nó do polígono. Para as $n-1$ rodadas seguintes, o polígono é movido em sentido horário para determinar os novos jogos. A Figura 2.1, demonstra a fatoração para as três primeiras rodadas de um campeonato com 6 times.

Figura 2.1: Método do Polígono para três primeiras rodadas com n = 6



Fonte: Mine [2]

Tabela 2.1: Solução obtida pelo Método do Polígono

	1	2	3	4	5	6	7	8	9	10
Time 1	6	4	2	5	3	6	4	2	5	3
Time 2	5	3	1	4	6	5	3	1	4	6
Time 3	4	2	5	6	1	4	2	5	6	1
Time 4	3	1	6	2	5	3	1	6	2	5
Time 5	2	6	3	1	4	2	6	3	1	4
Time 6	1	5	4	3	2	1	5	4	3	2

Fonte: autores

A Tabela 2.1 mostra os resultados obtidos pelo Método do Polígono para 6 times nas primeiras $n-1$ rodadas e espelhado para as $n-1$ rodadas seguintes no MDRR[1]. O mesmo método pode ser aplicado aos 20 times requeridos nesse trabalho prático, para esse número de times ou qualquer outro número par de times, um polígono com $n-1$ vértices é utilizado.

2.1.2 Segunda fase: Associando Times

A segunda fase consiste na associação entre os times reais e os times abstratos. Para representar times abstratos usam-se as letras correspondentes aos números gerados pelo método do polígono, já os times reais são representados por números, correspondentes a ordem de seus índices.

Para essa fase, primeiro obtêm-se uma matriz quadrada de $n \times n$ oponentes consecutivos gerada partir da escala criada pelo Método do Polígono.

[1]Mirrored Double Round Robin

São considerados como "oponentes consecutivos" no caso em que um time T_k possuir confrontos entre $T_i \times T_k$ e $T_j \times T_k$ em uma ou mais rodadas consecutivas, não importando a ordem dos jogos. Como exemplo, observa-se na Tabela 2.2 que os times B e E são oponentes consecutivos por 6 vezes de outros times.

Tabela 2.2: Oponentes consecutivos

	1	2	3	4	5	6	7	8	9	10
Time 1	F	D	**B**	**E**	C	F	D	**B**	**E**	C
Time 2	E	C	A	D	F	E	C	A	D	F
Time 3	D	**B**	**E**	F	A	D	**B**	**E**	F	A
Time 4	C	A	F	**B**	**E**	C	A	F	**B**	**E**
Time 5	B	F	C	A	D	B	F	C	A	D
Time 6	A	E	D	C	B	A	E	D	C	B

Fonte: autores

A Tabela 2.3 demonstra a matriz quadrada de oponentes consecutivos dos jogos entre os times abstratos. Nessa matriz podemos observar que além de B e E serem oponentes consecutivos por 6 vezes, outros times também são consecutivos por uma ou mais vezes, como exemplo, C e A também são consecutivos por 6 vezes.

Tabela 2.3: Matriz de oponentes consecutivos para $n = 6$

	A	B	C	D	E	F
A	-	1	6	5	2	4
B	1	-	2	5	6	4
C	6	2	-	2	5	3
D	5	5	2	-	2	4
E	2	6	5	2	-	3
F	4	4	3	4	3	-

Fonte: autores

A associação, então, ocorre entre os pares de times reais que possuem a menor distância entre suas sedes com os pares de oponentes consecutivos de maior número de confrontos. Essa heurística é descrita a seguir.

Inicialmente, pares de times abstratos são ordenados em ordem decrescente conforme o número de oponentes consecutivos (Tabela 2.5). Times reais são ordenados em ordem crescente de distância em relação às sedes de seus oponentes (Tabela 2.4), e são associados aos times abstratos nesta ordem. Dessa forma, os times abstratos com maior número de oponentes consecutivos serão preferencialmente associados aos times reais com as distâncias entre sedes menores.

Tabela 2.4: Pares ordenados de times reais.

1	(2,6)	(Atlético Mineiro, Cruzeiro)	0 Km
2	(1,6)	(América Mineiro, Cruzeiro)	0 Km
3	(1,2)	(América Mineiro, Atlético Mineiro)	0 Km
4	(3,5)	(Atlético Paranaense, Corinthians)	415 Km
5	(2,4)	(Atlético Mineiro, Botafogo)	440 Km
6	(1,4)	(América Mineiro, Botafogo)	440 Km
7	(4,6)	(Botafogo, Cruzeiro)	441 Km
8	(4,5)	(Botafogo, Corinthians)	444 Km
9	(1,5)	(América Mineiro, Corinthians)	585 Km
10	(2,5)	(Atlético Mineiro, Corinthians)	585 Km
11	(5,6)	(Corinthians, Cruzeiro)	587 Km
12	(3,4)	(Atlético Paranaense, Botafogo)	849 Km
13	(1,3)	(América Mineiro, Atlético Paranaense)	1000 Km
14	(2,3)	(Atlético Mineiro, Atlético Paranaense)	1000 Km
15	(3,6)	(Atlético Paranaense, Cruzeiro)	1001 Km

Fonte: autores

Tabela 2.5: Pares ordenados de times abstratos.

1	(E,B)	6	11	(D,F)	4	21	(D,E)	2
2	(C,A)	6	12	(F,A)	4	22	(C,D)	2
3	(B,E)	6	13	(F,D)	4	23	(C,B)	2
4	(A,C)	6	14	(B,F)	4	24	(B,C)	2
5	(E,C)	5	15	(A,F)	4	25	(E,D)	2
6	(D,B)	5	16	(F,B)	4	26	(A,E)	2
7	(D,A)	5	17	(E,F)	3	27	(E,A)	2
8	(C,E)	5	18	(F,C)	3	28	(D,C)	2
9	(B,D)	5	19	(F,E)	3	29	(B,A)	1
10	(A,D)	5	20	(C,F)	3	30	(A,B)	1

Fonte: autores

O pseudo-código para a heurística dessa associação é mostrado no Algorítimo 1.

Algoritmo 1: Associa times reais à times abstratos

EntradaEntrada **repita**
  Seja T_1 o próximo time real a ser associado a um time abstrato
  e seja T_2 o time, cuja cidade é mais próxima à cidade de T_1.
  se T_2 *já foi associado* **então**
    Encontre o primeiro par (a, b) de times abstratos, tal que a esteja
    associado ao time T_2 e b ainda não esteja associado a nenhum time real.
    $b \leftarrow T_1$
  senão
    Encontre o primeiro par (a, b) de times abstratos, tal
    que nenhum deles esteja associado a um time real.
    $a \leftarrow T_1$
  fim
até *todos os times reais sejam associados aos times abstratos*;

Fonte: Mine [2]

Essa heurística de associação tem como objetivo minimizar o deslocamento de um time T_k, isso baseia-se na ideia de que se existirem confrontos $T_i \times T_k$ e $T_j \times T_k$ consecutivos, a distância percorrida por T_k seria menor caso as sedes de seus oponentes consecutivos forem próximas.

2.1.3 Terceira fase: Mando de Campo

Na terceira e última fase para gerar uma solução inicial, são estabelecidos os mandos de campo. Considera-se para isso a restrição de três jogos consecutivos dentro ou fora de casa, assim sendo, para gerar uma escala viável deve-se alternar o máximo possível o mando de campo.

Os mandos de campo da primeira rodada são determinados aleatoriamente, e nas rodadas seguintes até a última do primeiro turno $(n-1)$, uma heurística é utilizada para determinar o mando de um jogo entre os times T_1 e T_2.

Para que a heurística possa ser usada, é preciso definir N_i como a quantidade total de jogos consecutivos dentro ou fora de casa que um time T_i jogou em rodadas anteriores, e a partir desse valor, o Algorítimo 2 é aplicado.

Algoritmo 2: Determina o mando de campo dos jogos

Determine, de forma aleatória, o mando de campo dos jogos da primeira rodada.
para cada *rodada* $k = 2, 3, ..., (n-1)$ *do primeiro turno* **faça**
  se $N_2 > N_1$ **então**
    se *Se* T_2 *jogou seu último jogo em casa* **então** a realização do jogo será na casa do time T_1;
    se *Se* T_2 *jogou seu último jogo fora de casa* **então** a realização do jogo será na casa do time T_2;
  fim
  se $N_2 < N_1$ **então**
    se *Se* T_1 *jogou seu último jogo em casa* **então** a realização do jogo será na casa do time T_2;
    se *Se* T_1 *jogou seu último jogo fora de casa* **então** a realização do jogo será na casa do time T_1;
  fim
  se $N_2 = N_1$ **então**
    se T_1 *jogou seu último jogo em casa e o time* T_2 *jogou seu último jogo fora de casa* **então** a realização do jogo será na casa do time T_2;
    se T_1 *jogou seu último jogo fora de casa e o time* T_2 *jogou seu último jogo em casa* **então** a realização do jogo será na casa do time T_1;
    senão o mando de campo é determinado aleatoriamente;
  fim
fim
Determinar os jogos do segundo turno espelhados ao primeiro turno.
Caso a escala ainda permanecer inviável, reinicie o algoritmo.

Fonte: Mine [2]

2.2 LNS da solução inicial

Após serem obtidos os resultados de custos da solução inicial, uma busca em profundidade é realizada utilizando-se do movimento de troca de mandos de campo.

Para isso, a escala obtida inicialmente é submetida a um número constante de interações de mudança de mando. Essas interações tem como objetivo alcançar melhores resultados por meio de mudanças do custo total da escala em cada interação.

A melhor solução que for encontrada nessas interações é definida então como a solução aproximada para o problema.

3 Análise de Complexidade

A análise considerada o pior caso, e sua complexidade é definida por tempo.

As Funções utilizadas pelo programa as quais um número fixo de instruções, sejam eles k, são executas independente de n, essas classificam-se como $\mathcal{O}(1)$ ou de complexidade constante e foram ignoradas na complexidade total do programa.

A partir do escopo principal do programa, a análise é dividida pelas seguintes partes:

1. Alocando os dados: $2 \times \underbrace{n+1}_{getlines()} + 2 \times \underbrace{n(n+1)}_{allocTable()} = 2n^2 + 4n + 1$
2. Obtendo os dados: $\underbrace{n+2}_{getCLUB()} + \underbrace{n+1}_{getCITY()} + \underbrace{n+2}_{getDIST()} = 3n + 5$
3. Usando o método do polígono e imprimindo os resultados:

$$\underbrace{n - 1 + (n-1)(\frac{n+2}{2})}_{initPolygon()} + \underbrace{n(n+1)}_{printEscala()} = \frac{3n^2 + 5n - 4}{2}$$

4. Associando os clubes: $\underbrace{n^3 + 5n^2 + 8n + 2}_{associaClub()}$
5. Copiando a escala gerada pelo polígono e definindo os mandos de campo:

$$\underbrace{n(2n-1)}_{copyTable()} + \underbrace{\frac{n^3 + 6n^2 + 12n}{4}}_{setmando()} + \underbrace{n(n+1)}_{printEscala()} = \frac{n^3 + 18n^2 + 12n}{4}$$

6. Busca por profundidade:

$$2 \times \underbrace{n(n+1)}_{allocTable()} + \underbrace{n(2n-1)}_{copyTable()} + \underbrace{n+1}_{freeMemory()} + ITER(2\underbrace{n(2n-1)}_{copyTable()} + \underbrace{\frac{n^3 + 6n^2 + 12n}{4}}_{setmando()})$$

$$= \frac{16n^2 + 8n + 4 + ITER(n^3 + 22n^2 + 8n)}{4}$$

- Define-se $ITER$ como um número constante de interações.

7. Liberando a memoria e imprimindo os resultados:

$$4 \times \underbrace{n+1}_{freeMemory()} + \underbrace{n(n+1)}_{printTabela()} + \underbrace{n+1}_{printTravel()} = n^2 + 6n + 5$$

Simplificando a soma das funções chamadas pelo escopo principal do programa e eliminado as constantes, a complexidade total do programa é definida pela equação: $n^3 + 37n^2 + 45n \leqslant cn^3 \rightarrow \mathcal{O}(n^3)$.

4 Lista das Funções

4.1 Funções dos Arquivos

`getlines()`	Retorna o numero de linhas validas de um arquivo.	$\mathcal{O}(n)$
`getCLUB()`	Carrega os clubes a partir do arquivo de clubes, salva os dados em *struct array*.	$\mathcal{O}(n)$
`getCITY()`	Carrega as cidades a partir do arquivo de cidades, salva os dados em um *struct array*.	$\mathcal{O}(n)$
`getDIST()`	Carrega as distancias a partir do arquivo de distancias, salva os dados em uma matriz alocada.	$\mathcal{O}(n)$
`errFile()`	Retorna a mensagem de erro caso um arquivo não for encontrado.	$\mathcal{O}(1)$

4.2 Funções da Lista Encadeada Simples

`create()`	Inicializa uma Lista Encadeada Simples	$\mathcal{O}(1)$
`atP()`	Dada a posição, retorna um nó da Lista.	$\mathcal{O}(n)$
`LLmin()`	Retorna um nó com o menor valor de um dado.	$\mathcal{O}(n)$
`LLmax()`	Retorna um nó com o maior valor de um dado.	$\mathcal{O}(n)$
`LLidx()`	Retorna a posição de um nó.	$\mathcal{O}(n)$
`LLpsh()`	Insere os dados a partir da posição cabeça.	$\mathcal{O}(1)$
`LLpop()`	Remove os dados a partir da posição cabeça.	$\mathcal{O}(1)$
`LLdel()`	Remove um nó, dado sua posição na Lista.	$\mathcal{O}(n)$
`LLins()`	Insere um nó, dado sua posição na Lista.	$\mathcal{O}(n)$
`LLchg()`	Troca a posição de dois nós.	$\mathcal{O}(n)$
`LLdup()`	Remove os nós com dados duplicados.	$\mathcal{O}(n)$
`LLinc()`	Ordena a Lista em ordem crescente.	$\mathcal{O}(n)$
`LLdec()`	Ordena a Lista em ordem decrescente.	$\mathcal{O}(n)$
`LLprt()`	Retorna a impressão da Lista.	$\mathcal{O}(n)$
`LLclr()`	Libera os dados da Lista da memoria.	$\mathcal{O}(n)$
`isEmpty()`	Retorna se uma Lista está ou não vazia.	$\mathcal{O}(1)$

4.3 Funções de Retorno de Impressão

`printTabela()`	Retorna a impressão formatada da escala.	$\mathcal{O}(n^2)$
`printEscala()`	Retorna a impressão não formatada da escala..	$\mathcal{O}(n^2)$
`printTravel()`	Retorna a impressão das distancias percorrida por cada um dos times.	$\mathcal{O}(n)$
`printMatrix()`	Retorna a impressão de uma matriz $n \times n$.	$\mathcal{O}(n^2)$
`printCLUB()`	Retorna a impressão não formatada do *struct array* de clubes.	$\mathcal{O}(n)$
`printCITY()`	Retorna a impressão não formatada do *struct array* de cidades.	$\mathcal{O}(n)$

4.4 Funções da Solução inicial

`initPolygon()`	One-factorization para gerar uma tabela inicial valida.	$\mathcal{O}(n^2)$
`associaClub()`	Associa os times reais aos times abstratos conforme o pseudo-código descrito no Algorítimo 1.	$\mathcal{O}(n^3)$
`changeClube()`	Altera o índice dos clubes conforme a associação realizada em `associaClub()`.	$\mathcal{O}(n)$
`buildCalvin()`	Cria os pares de times reais para `associaClub()`.	$\mathcal{O}(n^2)$
`buildHarold()`	Cria os pares de times abstratos para `associaClub()`.	$\mathcal{O}(n^2)$
`buildLinked()`	Cria uma lista simplesmente encadeada para o controle dos times associados na `associaClub()`.	$\mathcal{O}(n)$
`islinked()`	Verifica se um time real já esta associado.	$\mathcal{O}(n)$
`associar()`	Associa times reais com abstratos na lista criada por `buildlinked()`.	$\mathcal{O}(n)$
`areClubs()`	Verifica se todos os times já foram associados.	$\mathcal{O}(n)$
`findAT1()`	Função booleana de retorno para `associaClub()`.	$\mathcal{O}(1)$
`findBT1()`	Função booleana de retorno para `associaClub()`.	$\mathcal{O}(1)$

4.5 Funções de Mando de Campo

`setmando()`	Determina o mando de campo dos times conforme o pseudo-código descrito no Algorítimo 2.	$\mathcal{O}(n^3)$
`sorteia()`	Determina randomicamente os mandos de campo da primeira rodada.	$\mathcal{O}(n)$
`CSTMando()`	Verifica se a condição de três jogos consecutivos dentro ou fora de casa foi respeitada.	$\mathcal{O}(n^2)$
`espelha()`	Espelha os mandos gerados no primeiro turno para o returno.	$\mathcal{O}(n^2)$
`randint()`	Gera um inteiro randomicamente conforme o *seed* do sistema.	$\mathcal{O}(1)$

4.6 Funções Gerais

`allocTable()`	Aloca uma matriz de ponteiros.	$O(n^2)$
`freeMemory()`	Libera a memoria alocada por `allocTable()`.	$O(n)$

`copyTable()`	Copia dados entre matrizes.	$O(n^2)$
`shiftArray()`	Movimenta os dados em um array (*shift*).	$O(1)$
`viagem()`	Retorna a distancia entre as sedes dos times.	$O(1)$
`custos()`	Calcula e retorna os custos totais de uma escala de jogos gerada, assim como o custo de deslocamento individual dos times.	$O(n^2)$
`timeresult()`	Retorna o tempo de execução até determinado ponto.	$O(1)$
`wait()`	Função de espera (pausa o tempo de execução por n segundos).	$\mathcal{O}(n)$
`ask()`	Espera por uma entrada fornecida pelo usuário.	$\mathcal{O}(n)$

5 Análise de Resultados

Tabela 5.1: Análise dos resultados.

S_0	S_k	k	Tentativas	Tempo	Diferença %	Ganho
837635	784889	3	19698	1s 903ms	6.30%	52746
848148	790269	10	18519	1s 795ms	6.82%	57879
814842	772594	9	19458	1s 892ms	5.18%	42248
864708	782358	5	18773	1s 823ms	9.52%	82350
858557	780693	8	19692	1s 910ms	9.07%	77864
858637	755076	9	19651	1s 912ms	12.06%	103561
840565	781085	4	20540	1s 984ms	7.08%	59480
825463	781733	6	18880	1s 830ms	5.30%	43730
848581	772612	9	19195	1s 862ms	8.95%	75969
844549	770172	8	18408	1s 791ms	8.81%	74377
865979	779945	6	20467	1s 985ms	9.93%	86034
857004	767938	5	19898	1s 928ms	10.39%	89066
795849	774157	3	19754	1s 914ms	2.73%	21692
862298	769219	8	18952	1s 846ms	10.79%	93079
842242	777349	4	19309	1s 868ms	7.70%	64893
844337	776005				8.042%	68331.6

Fonte: autores

Mediante a análise de resultados obtidos, pode-se perceber a vantagem do uso de um método heurístico como o de LNS (Large Neighborhood Search), que é baseado no princípio de destruição e reconstrução de rotas. Onde utilizamos a primeira tabela gerada pelo método de 3 fases, a reorganizando K vezes, de forma a construir uma melhor solução.

Na tabela acima temos então que S(0) é o valor inicial obtido, referente a distância em km que os times irão percorrer durante a realização do campeonato, e S(K) o valor da distância prevista após a aplicação do método heurístico.

Portanto a abordagem proporciona uma melhora significativa na resolução do problema, onde em média existe uma redução de aproximadamente 8.05% na distância total a ser percorrida pelos times.

Contudo, sabendo que normalmente os clubes de futebol fretam aviões para transportar seus jogadores; e tendo em mente que o querosene de aviação (QAV)[**?**] possui o

preço atual de R$3,30 o litro, e que um avião adequado para acomodar um time de futebol, viaja a 880km por hora utilizando em média 8,09 litros por quilômetro[**?**]. Podemos concluir que com o uso dessa heurística, a redução de 68.332km de viagem durante o campeonato resultariam na economia de R$1.824.259 e 78 horas de viagem.

Desta forma pode-se traduzir para o cenário das competições brasileiras que não só seriam reduzidos os de gastos dos clubes, mas também os jogadores sofreriam de um desgaste menor entre as partidas.

Referências

[1] Kelly Easton, George Nemhauser, and Michael Trick. Solving the traveling tournament problem. https://mat.gsia.cmu.edu/TOURN/ttp_ipcp.pdf. [Acesso em: 09-Outubro-2018].

[2] Marcio Mine. Programação de jogos de competições esportivas. http://www.decom.ufop.br/prof/marcone/projects/ttp/Publications/RT_FAPEMIG_Mine_2006.pdf. [Acesso em: 07-Outubro-2018].

[3] Celso Ribeiro and Sebastiáan Urrutia. Heuristics for the mirrored traveling tournament problem. https://pdfs.semanticscholar.org/89c1/3241f875a17238d5300ab42435aa6c0fbf66.pdf. [Acesso em: 08-Outubro-2018].

[4] Michael Trick. Challenge traveling tournament instances. https://mat.gsia.cmu.edu/TOURN/. [Acesso em: 09-Outubro-2018].

O histórico do desenvolvimento desse trabalho se encontra online em: https://github.com/Durfan/ufsj-aeds3-tp2.