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

Pablo Cecilio Oliveira
Alexander Cristian

1 Introdução

Otimização é o processo de melhorar algo. Basicamente é a ferramenta de matemática que nós confiamos para obter respostas aos problemas. Em matemática e ciência da computação, um problema de otimização é o problema de encontrar a melhor solução de todas as soluções viáveis. A terminologia "melhor" solução implica que há mais de uma solução e as elas não são de igual valor. Esses problemas de otimização no domínio científico e da vida real encontram soluções em pesquisa algorítmica que mostra o uso de métodos clássicos, métodos heurísticos e baseados na natureza métodos para calcular a solução.

Na solução de problemas difíceis onde não há soluções exatas eficientes, abordagens heurísticas como Simulated Annealing, Busca Tabu, GRASP (Greedy Randomized Adaptive Search Procedure), VNS (Variable Neighborhood Search), VND (Variable Neighborhood Descent), o ILS (Iterated Local Search) e o Método de Reconexão por Caminhos (Path Relinking) se tornam alternativas populares.

Então neste trabalho nós apresentamos um algoritmo que faz uso de uma dessas abordagens, para resolver um problema de otimização conhecido como Traveling Tournament Problem.

1.1 Especificação do problema

O Problema da viagem do torneio (Traveling Tournament Problem) é um problema de agendamento de esportes que abstrai as questões importantes na criação horários, onde a viagem em equipe é uma questão importante. Ligas esportivas profissionais existem em todo o mundo. Essas ligas são muitas vezes de grande importância econômica devido às enormes receitas geradas pela venda de bilhetes e direitos de transmissão para os jogos. Portanto, o planejamento dessas ligas é de suma importância. Um aspecto importante é a geração de um calendário para os torneios que especifica a ordem em que as equipes jogam entre si durante a temporada e o local de cada jogo.

Portanto consideramos um torneio de n times, onde n é um número par. Em um torneio simple round-robin (SRR), cada time joga com cada um dos demais exatamente

uma vez, nas n - 1 rodadas pré-estabelecidas. O jogo entre os times i e j é representado pelo par não ordenado i, j. Há n=2 jogos em cada rodada. Cada time joga exatamente uma vez por rodada. Em um torneio double round-robin (DRR), cada time joga com cada um dos outros duas vezes, uma em casa e outra fora de casa. Um torneio mirrored double round-robin (MDRR) é um torneio SRR nas primeiras n – 1 rodadas, seguido das mesmas rodadas iniciais mas invertendo a sua localização. Uma viagem é uma sequência de jogos fora de casa. Enquanto uma estadia é uma sequência de jogos em casa. Assumimos que cada time tem um estádio para seu uso na sua cidade natal. As distâncias entre as cidades são conhecidas. Cada time parte de sua cidade no início do torneio e retorna à sua cidade no fim do torneio, se já não estiver lá. Sempre que duas rodadas de um time forem fora de casa, ele não retorna à sua cidade natal entre elas.

O problema da viagem do torneio (Traveling Tournament Problem) é definido como a seguir. Dados n times e as distâncias entre as suas cidades de origem, o problema consiste em definir uma sequência DRR de tal forma que nenhum time joga mais de três jogos fora de casa, não há repetições (ou seja, dois jogos consecutivos entre os mesmos times em localidades diferentes), e a soma das distâncias percorridas pelos times é minimizada. O problema da viagem do torneio espelhada tem uma restrição adicional: os jogos da rodada k são os mesmos da rodada k + (n- 1), k = 1; . . . ; n - 1, com a localização invertida.

1.2 Solução proposta

Para a resolução deste problema utilizamos o método de GRASP (Greedy Randomized Adaptive Search Procedures) que é uma meta-heurística constituída por heurísticas construtivas e busca local. Consiste de múltiplas aplicações de busca local, cada uma iniciando de uma solução diferente. As soluções iniciais são geradas por algum tipo de construção randômica gulosa ou algum esquema de perturbação, esse método é constituído basicamente de duas fases: uma fase de construção e uma fase de busca local, cujo objetivo é convergir à solução encontrada na fase de construção para um ótimo local.

Na segunda fase ocorre o refinamento da solução gerada pela fase de construção, aplicando um método de busca local. Quanto melhor é a qualidade da solução gerada pela heurística de construção, maior é a velocidade de convergência desta solução para um ótimo local.

2 Implementação

Para a construção de uma solução inicial, utiliza-se o método de três fases proposto por Ribeiro e Urrutia (2004a)[1].

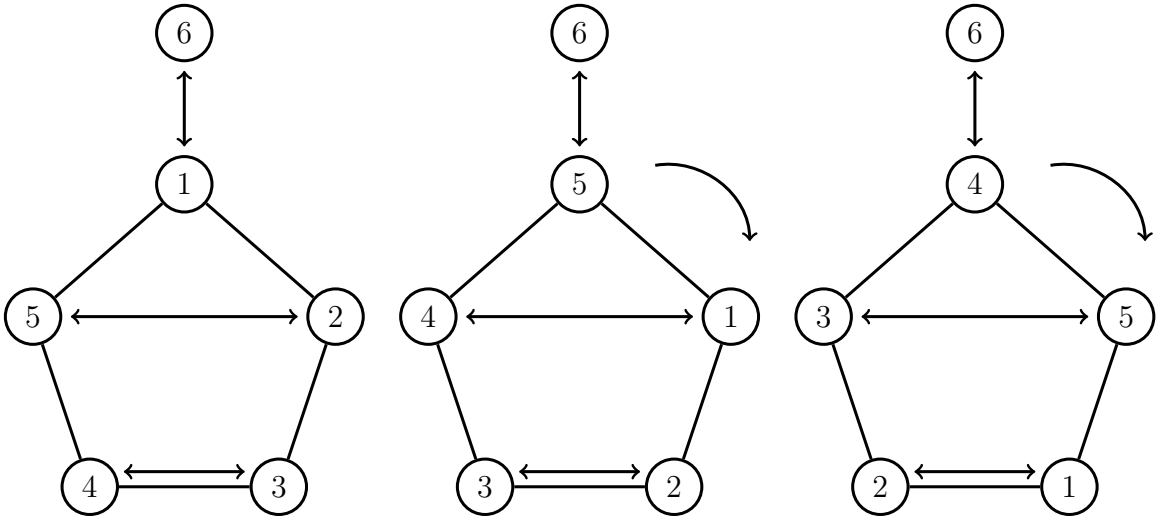
2.1 Método de três fases para geração de uma solução inicial

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 1, demonstra a fatoração para as três primeiras rodadas de um campeonato com 6 times.

Figura 1: Método do Polígono para três primeiras rodadas com n = 6



Fonte: Mine [2]

Tabela 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 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. 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.

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 que os times B e E são oponentes consecutivos por 6 vezes de outros times.

Tabela 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 Figura 2 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.

Figura 2: Matriz de oponentes consecutivos para $n = 6$

$$\begin{bmatrix} 0 & 1 & 6 & 5 & 2 & 4 \\ 1 & 0 & 2 & 5 & 6 & 4 \\ 6 & 2 & 0 & 2 & 5 & 3 \\ 5 & 5 & 2 & 0 & 2 & 4 \\ 2 & 6 & 5 & 2 & 0 & 3 \\ 4 & 4 & 3 & 4 & 3 & 0 \end{bmatrix}$$

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 4). Times reais são ordenados em ordem crescente de distância em relação às sedes de seus oponentes (Tabela 3), 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 3: 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 4: 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

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 associação tem como objetivo minimizar o deslocamento de um time T_k, baseando-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 suas sedes 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]

3 Análise de complexidade

Para as Funções utilizadas pelo programa as quais um número fixo de instruções, sejam eles k, são executas independente de n, essas classificam-se como $O(1)$ ou de complexidade constante. Dentre as listadas nas seções seguintes, se encontram:

- Funções que retornam uma condição booleana;
- Inicialização da Lista Simplesmente Encadeada;
- Inserção ou remoção de um nó a partir do nó "cabeça" de uma Lista Simplesmente Encadeada;
- Funções de retorno de impressão para mensagens de erro, ou para o retorno de impressão do tempo decorrido até determinado ponto relativo ao código;
- A Função que realiza a copia de um trecho de memória, ou a que retorna um conteúdo endereçado diretamente para uma área de memoria;

- A Função que gera um números randômicos através do seed fornecido pelo sistema.

Para as Funções analisadas como sendo de complexidade linear, ou $O(n)$, tais que n mais um número fixo de instruções são menores do que cn ($f(n) = (n+1) \Rightarrow (n+1) \leqslant cn$). Essas compõe a maioria do programa e são listadas de forma geral a seguir:

- A Função que retorna um o numero de linhas de um arquivo;
- Funções que carregam os dados dos arquivos para estruturas de dados;
- Funções que carregam os dados dos arquivos para estruturas de dados;
- Funções de uso geral da Lista Simplesmente Encadeada:
  - Funções que retorna um nó que contem um menor valor de um dado, ou a que retorna um nó que contem o maior valor de um dado;
  - Função que retorna a posição de um nó na lista;
  - Funções que carregam os dados dos arquivos para estruturas de dados;
  - Funções que removem ou inserem um nó da Lista conforme sua posição;
  - Função que troca as posições de dois nós;
  - Função que remove nós com dados duplicados;
  - Funções que ordenam a lista em ordem crescente ou decrescente;
  - Função que retorna a impressão da Lista;
  - Função que libera os dados da Lista da memoria
- Funções de uso geral da Lista Simplesmente Encadeada:

3.1 Funções de arquivos para os dados

Tabela 5: Funções de arquivos para os dados.

Função	Finalidade	Complexidade
$getlines()$	Retorna o numero de linhas validas de um arquivo.	$O(n)$
$getCLUB()$	Carrega os clubes a partir do arquivo de clubes, salva os dados em array struct.	$O(n)$
$getCITY()$	Carrega as cidades a partir do arquivo de cidades, salva os dados em um array struct.	$O(n)$
$getDIST()$	Carrega as distancias a partir do arquivo de distancias, salva os dados em uma matriz alocada.	$O(n)$
$errFile()$	Retorna a msg de erro caso o arquivo não for encontrado.	$O(1)$

Fonte: autores

3.2 Funções da Lista Encadeada Simples

Tabela 6: Funções da Lista Encadeada Simples.

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

Fonte: autores

3.3 Funções de Retorno de Impressão

Tabela 7: Funções da Solução inicial.

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

Fonte: autores

3.4 Funções da Solução inicial

Tabela 8: Funções da Solução inicial.

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

Fonte: autores

3.5 Funções de uso comum ao programa

Tabela 9: Funções de uso comum ao programa.

Função	Finalidade	Complexidade
$allocTable()$	Aloca uma matriz de ponteiros.	$O(n)$
$freeMemory()$	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).	$O(n)$
$ask()$	Espera por um input fornecido pelo usuário.	$O(n)$

Fonte: autores

4 Análise de resultados

Tabela 10: 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				8.09%	68332

Fonte: autores

5 Considerações finais

Durante o desenvolvimento vários métodos foram testados com o objetivo de encontrar a maneira mais eficiente de solucionar o problema da viagem do torneio. //comentários sobre as dificuldades no desenvolvimento do código

Referências

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

[2] Marcio Tadayuki 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].

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