SAMUEL JONAS DOS SANTOS FONSECA

Orientador: Marco Antonio Moreira de Carvalho

ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS APLICADO AO PLANEJAMENTO DE TORNEIOS ESPORTIVOS

Ouro Preto

Setembro de 2017

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

ALGORITMO GENÉTICO DE CHAVES ALEATÓRIAS VICIADAS APLICADO AO PLANEJAMENTO DE TORNEIOS ESPORTIVOS

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

SAMUEL JONAS DOS SANTOS FONSECA

Ouro Preto
Setembro de 2017

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

Algoritmo Genético de Chaves Aleatórias Viciadas Aplicado ao Planejamento de Torneios Esportivos

SAMUEL JONAS DOS SANTOS FONSECA

Monografia defendida e aprovada pela banca examinadora constituída por:

Dr. Marco Antonio Moreira de Carvalho – Orientador
Universidade Federal de Ouro Preto

Dr. Marcone Jamilson Freitas Souza
Universidade Federal de Ouro Preto

Msc. Felipe Lopes Melo Faria
Universidade Federal de Ouro Preto

Ouro Preto, Setembro de 2017

Resumo

A evolução dos esportes faz com que os torneios profissionais se tornem atividades econômicas em todo o mundo. Grandes eventos esportivos movimentam quantias enormes de dinheiro e atraem fãs de todos os lugares. As equipes lidam com altos custos para poder participar dos torneios, o que envolve investimento em atletas e em infraestruturas, além das viagens realizadas para enfrentar os adversários. Neste contexto, surge o Problema de Escalonamento de Torneios Esportivos, um problema NP-Difícil que consiste em gerar uma tabela de jogos minimizando a distância total viajada pelas equipes que participam de um torneio. Neste trabalho é apresentada a aplicação do método Algoritmo Genético de Chaves Aleatórias Viciadas a este problema, com algumas alterações em seus principais processos. Além disso, são apresentados também os conceitos e a aplicação de diferentes métodos de busca local e perturbação ao problema tratado. Os experimentos computacionais realizados abarcam três diferentes conjuntos de instâncias, que utilizam dados reais e artificiais. Os resultados reportados demonstram a eficiência do método para instâncias pequenas, se igualando ou aproximando dos valores de referência, e a perda da qualidade das soluções para as instâncias maiores. É apresentada também, uma análise que mostra o comportamento do método proposto, juntamente com os métodos de busca local utilizados, nos diferentes conjuntos de instâncias.

Abstract

The evolution of sports turns professional tournaments into important economic activities around the world. Major sport events move huge amounts of money and attract fans everywhere. The teams deal with high costs to participate in these tournaments, which involves investing in athletes and infrastructures, as well as the trips made to face the opponents. In this context, the Traveling Tournament Problem arises, a NP-Hard problem that consists in generating a timetable to minimize the total distance traveled by the participant teams of a tournament. In this work, we presente an application of the Biased Random-Key Genetic Algorithm to this problem, with some changes in its main processes. Besides, we also present the concepts and the application of different methods of perturbation and local search to the problem. Computational experiments involves three different instances sets, which use real and artificial data. The reported results demonstrate the efficiency of the method using small instances, matching or approaching the reference values, and also the loss of solution quality for larger instances. We also present an analysis that shows the behavior of the proposed method, together with the local search methods, for the different instances sets.

Dedico este trabalho ao meu pai Nilton, minha mãe Ervânia, minha irmã Rayanne, minha namorada Monique e a todos os meus amigos que sempre acreditaram em mim.

Agradecimentos

Agradeço a minha família, namorada e amigos por todo apoio que me deram. Agradeço também ao pessoal da Cachaça Gestor pela compreensão e ao meu orientador Marco por toda ajuda e paciência.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

O crescimento do esporte nos tempos atuais como uma prática de exercícios ou como uma indústria de entretenimento é evidente. O que antes era considerado apenas uma forma de lazer hoje exerce um papel essencial no cotidiano das pessoas, atuando diretamente na melhoria da educação e da saúde pessoal. Além disso, o esporte é um meio de estabelecer a interação entre pessoas de raças e classes econômicas distintas e reunir países de diferentes lugares do mundo, trazendo diversos benefícios sociais e econômicos.

Nesse contexto, o esporte vem se desenvolvendo também como uma forma de negócio. Muitos países disputam o direito de sediar eventos, como os Jogos Olímpicos e a Copa do Mundo de Futebol por exemplo, a fim de atrair espectadores de todo o mundo e também investimentos, com isso, desenvolvendo-se economicamente e melhorando as condições urbanas do país. Desta forma, além de promover a cultura e o turismo, sediar eventos esportivos contribui para a geração de empregos, melhoria de rodovias e o aumento da segurança dos locais que abrigam a realização dos eventos.

O envolvimento de dinheiro faz com que as ligas profissionais de esportes (tais como futebol, basquete, beisebol, hóquei) se tornem ainda mais sérias. Grandes eventos esportivos movimentam bastante dinheiro e atraem patrocinadores e interessados em investir nos atletas. No futebol, o esporte mais popular do mundo, a quantidade de dinheiro que gira em torno de patrocínio, salários, contratos e premiações é muito grande. Uma publicação feita pelo site Lance![1], mostra que o jogador mais bem pago do mundo recebe em torno de 74 milhões de euros (cerca de 260 milhões de reais) por temporada, incluindo os vencimentos brutos, rendimentos de patrocínio e os bônus recebidos pelo atleta.

Os custos envolvidos em um campeonato também são altos. O investimento em *marketing* e nos atletas são parte dos gastos que as equipes possuem. Existem também os gastos com infraestrutura (manutenção dos estádios/ginásios, médicos, funcionários, entre outros), além dos custos operacionais para poder participar dos campeonatos, como por exemplo, as viagens

[1]Disponível em http://www.lance.com.br/motionsense/messi-ronaldo-neymar-lideram-lista-mais-bem-pagos-mundo.html. Acessado em 27 de Outubro de 2016.

para enfrentar os adversários. Esta é uma das principais dificuldades enfrentadas pelos times, uma vez que consome muito tempo, dinheiro e afeta o bem-estar de todos os membros da equipe, principalmente os jogadores que são atletas de alto desempenho e precisam estar sempre em boas condições físicas.

Problemas como estes atraem pesquisadores de diversas áreas da computação, como otimização combinatória e pesquisa operacional, por exemplo. Um destes problemas é conhecido como *Traveling Tournament Problem* (TTP), que é bastante estudado na área de Escalonamento de Competições Esportivas (*Sport Scheduling*) e consiste em, dado um conjunto de n times e as distâncias entre as cidades de cada um deles, gerar a tabela ou calendário de um campeonato contendo todas as partidas entre os times. Dentre os possíveis objetivos observados do TTP estão a minimização da distância viajada pelas equipes e a minimização de *breaks*, isto é, interrupções no padrão de mando de campo (ou seja, sequência de jogos em casa e fora). Em alguns casos, são incluídos restrições e objetivos impostos pela mídia relacionados à transmissão dos jogos pela televisão, por exemplo.

Adicionalmente, o TTP está relacionado a campeonatos em que cada time joga contra todos os outros um número fixo de vezes. Comumente, os times se enfrentam exatamente duas vezes, o que define a competição como *Double Round Robin Tournament* (DRRT). Além disso, existem também competições que são definidas como *Single Round Robin Tournament* (SRRT), em que existe apenas um confronto entre as equipes.

O TTP possui diferentes versões, em que cada uma possui um conjunto de restrições específicas que as caracterizam. Um exemplo, é a versão que considera que distância entre todas as cidades é igual a 1, conhecida como *Constant Distance Traveling Tournament Problem* (CTTP). Neste trabalho será abordada a versão *Mirrored Traveling Tournament Problem* (mTTP), que considera que o torneio é dividido em dois turnos (por sua vez divididos em rodadas) onde o segundo turno é o "espelho" do primeiro. Isto significa que a sequência de jogos dos dois turnos será igual, porém, os times alternam o mando de campo.

Dentre os diversos métodos aplicáveis ao mTTP, está o Algoritmo Genético de Chaves Aleatórias Viciadas (*Biased Random-Key Genetic Algorithm*, BRKGA), proposto por Gonçalves e Resende (2011). Este é um método recente que se baseia na teoria de *Darwin* sobre a evolução das espécies, presente nos Algoritmos Genéticos, para resolver problemas de otimização combinatória. O BRKGA demonstrou ser muito eficiente em diversos casos considerados difíceis, pois utiliza de estratégias de alto nível para melhorar um conjunto de soluções.

1.1 Motivações

Há duas motivações principais para a abordagem do mTTP. Primeiramente, pela sua relevância teórica, em que o problema abordado pertence à classe NP-Difícil. Segundo, por se tratar de um problema de aplicação prática no contexto esportivo, tornando-se essencial para

o planejamento das tabelas de diversas competições como as de futebol, basquete, beisebol e hóquei, por exemplo, obtendo resultados que diminuem os gastos e o desgaste físico das equipes nas viagens realizadas durante um campeonato. Ainda, pode ser aplicado ao planejamento de transmissão de TV, rendendo lucros para as empresas que transmitem os jogos e para as equipes envolvidas. Gerar uma boa tabela de jogos pode evitar que um time viaje mais do que os outros, fazendo assim uma competição mais justa para todas as equipes.

Adicionalmente, há motivação para o emprego do método Algoritmo Genético de Chaves Aleatórias Viciadas, recentemente proposto na literatura Gonçalves e Resende (2011). Este método têm sido aplicado com sucesso em diferente problemas práticos também pertencentes à classe NP-Difícil, vide Prasetyo et al. (2015). Esta é a primeira aplicação deste método ao mTTP reportada na literatura.

1.2 Objetivos

Este trabalho consiste em desenvolver o método Algoritmo Genético de Chaves Aleatórias Viciadas para a resolução do mTTP, utilizando também diferentes métodos de busca local e perturbação. São objetivos específicos:

- Realizar pesquisas para geração de embasamento teórico e revisão bibliográfica sobre o mTTP;
- Elaborar uma heurística consistente que possa ser utilizada no problema e que permita a obtenção rápida de soluções próximas da solução ótima sem que se perca a vantagem da busca sistemática;
- Realizar pesquisas dos diferentes métodos de buscas locais que possam ser aplicados ao mTTP para a melhoria fina das soluções;
- Avaliar o método implementado considerando dados reais e também problemas teste publicamente disponíveis, realizando uma análise crítica considerando outros métodos da literatura.

1.3 Organização do Trabalho

O restante do trabalho é composto por seis capítulos, divididos da seguinte maneira: o Capítulo 2 apresenta a revisão da literatura de 1981 até 2016. A fundamentação teórica do mTTP é apresentada no Capítulo 3. No Capítulo 4 é apresentada uma implementação inicial do método proposto, além dos conceitos e aplicações dos métodos de busca local utilizados. Os experimentos realizados são apresentados no Capítulo 5. As conclusões sobre o trabalho são apresentadas no Capítulo 6.

Capítulo 2

Revisão da Literatura

Neste capítulo são brevemente descritos os principais trabalhos relacionados ao TTP e ao mTTP. Nesta revisão, os trabalhos são apresentados em ordem cronológica com exceção dos quatro últimos, que estão agrupados por apresentarem uma mesma heurística. Os trabalhos revisados estão divididos em:

- Heurísticas, Metaheurísticas e Métodos Híbridos: apresentam métodos para solução do mTTP;
- Trabalhos teóricos: definem os problemas de *Sport Scheduling* e apresentam bibliografias comentadas, a complexidade do problema e uma visão geral do TTP e do mTTP.

A maioria dos trabalhos que foram revisados utilizam as instâncias referentes ao site oficial do renomado pesquisador Michael Trick [1], que oferece diferentes tipos de instâncias para o TTP e suas versões. Esta plataforma divide as instâncias por esportes e por tipos (constantes, circulares e galáticas), além de apresentar os melhores resultados até o momento com os respectivos autores para cada uma das instâncias.

2.1 Heurísticas, Metaheurísticas e Métodos Híbridos

O trabalho proposto por Nemhauser e Trick (1998) apresenta uma heurística de três fases (*Three Phase Approach*, TPA) que consiste em: 1) gerar os padrões de jogos em casa e fora (*Home-Away Pattern, HAP*); 2) atribuir os jogos aos HAPs, resultando em uma *timetable*; 3) encontrar a solução atribuindo os times aos jogos. O trabalho é uma aplicação prática nas competições entre as universidades da *Atlantic Coast Conference* (ACC), um grupo de nove universidades dos Estados Unidos que disputam diversos esportes, sendo que o foco do trabalho é a competição de basquete.

[1] http://mat.gsia.cmu.edu/TOURN/

Ribeiro e Urrutia (2007) propõem duas heurísticas para resolver o mTTP. A primeira é uma combinação de GRASP com ILS, denominada GRILS-mTTP. A segunda é uma heurística criada pelos autores que utiliza o *Polygon Method* para gerar uma tabela de jogos para n times artificiais. Depois, estes times são associados aos times reais que participam do campeonato e, finalmente, é criado um torneio associando os estádios aos jogos entre os times reais. Os resultados dos experimentos foram comparados com os melhores conhecidos até então. A segunda heurística não obteve melhores resultados, enquanto o GRILS-mTTP conseguiu atingir resultados melhores do que os conhecidos até então. Este artigo, embora publicado em versão impressa em 2007, foi disponibilizado *online* em 2005 e desde então tem sido referenciado por outros trabalhos da literatura.

Outro artigo proposto por Urrutia e Ribeiro (2004), também utiliza a heurística GRILS-mTTP para resolver uma classe de instâncias constantes (a distância entre qualquer par de estádios é igual a 1) criadas pelos autores. Estas instâncias são usadas para definir uma conexão entre *breaks* e distâncias, em que maximizar o número de *breaks* faz com que sejam realizadas menos viagens, minimizando assim a distância viajada. Os valores obtidos nos experimentos, foram iguais aos limites inferiores para a maioria das instâncias. Segundo os autores, as instâncias testadas neste trabalho são as maiores já resolvidas até então, sendo que anteriormente a maior instância continha apenas 8 times e neste trabalho foram resolvidas instâncias com até 20 times.

O trabalho proposto por Biajoli e Lorena (2006) resolve o mTTP utilizando um método baseado em Algoritmo Genético (AG) para construir novas soluções, além de usar a metaheurística *Simulated Annealing* (SA) para melhorar a busca local nas soluções encontradas. Na realização dos experimentos, os resultados obtidos foram exatamente iguais aos melhores conhecidos até então. Por outro lado, para algumas instâncias os resultados se distanciaram muito do melhor e foram consideradas ruins pelos autores. Utilizando dados reais do campeonato brasileiro de futebol de 2003, foi observada uma redução de 51,2% no total da distância percorrida pelos times, em que antes era de 1.048.134 km e o resultado obtido foi de 511.256 km.

Van Hentenryck e Vergados (2006) propõem uma metaheurística denominada *Traveling Tournament Simulated Annealing* (TTSA), que utiliza a conhecida metaheurística *Simulated Annealing* para ajustar a função objetivo dinamicamente para equilibrar a proporção da exploração de regiões viáveis e inviáveis do espaço de busca do problema. Ainda, é utilizada a estratégia chamada *reaquecimento* para escapar de ótimos locais. O TTSA foi testado utilizando-se instâncias de torneios com e sem a restrição *mirrored* e, em alguns casos, os resultados obtidos foram melhores do que os conhecidos até então.

Rasmussen e Trick (2007) propõem um método denominado *Pattern Generating Benders Approach* (PGBA), que consiste em gerar os HAPs usando o método de *Benders*, atribuir estes HAPs aos times e obter uma tabela atribuindo os jogos para os times de acordo com

os HAPs. O PGBA foi comparado com o TPA, Nemhauser e Trick (1998), e os resultados foram parecidos para instâncias pequenas, mas para as maiores, o PGBA apresentou uma performance muito melhor que a do TPA. Para as instâncias de distância constante, os limites inferiores encontrados foram iguais aos de Urrutia e Ribeiro (2004).

Para resolver o mTTP, Cheung (2008) propõe uma heurística que utiliza a teoria dos grafos. Os vértices representam os jogos entre os times, em que dois vértices só serão adjacentes se estes jogos não puderem ocorrer simultaneamente na mesma rodada. O método consiste em fatorar o grafo em conjuntos independentes, obtendo assim todos os jogos que podem acontecer na mesma rodada. Para cada fatoração, é criada uma solução considerando a menor distância possível. Foram utilizadas instâncias com apenas 8 times, e os resultados foram equivalentes aos melhores conhecidos até então, obtidos por Ribeiro e Urrutia (2007).

O trabalho de Cheung (2009) utiliza o método de Decomposição de *Benders* para calcular os limites inferiores para o mTTP. Para todas as instâncias utilizadas no experimento computacional, os limites inferiores sofreram pequenas melhorias em relação aos que eram anteriormente os melhores conhecidos. Segundo o autor, ainda poderiam ser obtidos resultados melhores permitindo que os cálculos continuassem. Porém, a execução foi interrompida quando um ponto de retorno decrescente parecia ter sido atingido.

Goossens e Spieksma (2009) apresentam uma aplicação do mTTP no campeonato belga dos anos 2005, 2006, 2007 e 2008. A heurística proposta consiste em gerar os HAPs e atribuí-los aos times para determinar as partidas. Foram consideradas algumas restrições, como *place constraints* (que impede que um certo time jogue em casa ou fora em uma determinada rodada), *complementarity constraints* (garantem que dois oponentes não joguem em casa ao mesmo tempo) e *television constraints* (em cada rodada um número de *top teams*, os melhores nos últimos 10 anos, deve jogar uma partida fora de casa). São aplicadas penalidades quando as restrições são violadas, em que cada uma possui um nível de prioridade. O objetivo é minimizar estas penalidades. Os valores foram comparados com os obtidos manualmente pelos organizadores oficiais do campeonato belga, e, segundo os autores, o método apresentado obtém resultados aproximadamente sete vezes melhores em todos os aspectos.

Carvalho e Lorena (2012) apresentam dois modelos de programação inteira para o mTTP, em que o primeiro tem o objetivo de minimizar a distância total viajada, e o segundo, minimizar a maior distância viajada. Os modelos são baseados em detecção de conjuntos independentes em grafos (que representam os jogos que podem ser feitos simultaneamente no mesmo turno). Para as instâncias com 4 times, o resultado ótimo foi encontrado, mas para as instâncias com mais do que 6 times, o limite de tempo foi excedido. Interrompendo a execução ao encontrar a primeira solução viável, foram obtidos valores piores do que os melhores conhecidos para todas as instâncias. Ao interromper a execução de acordo com um *gap* definido, nos dois modelos foram obtidos resultados piores porém próximos do melhor conhecido, enquanto para as instâncias maiores o tempo limite foi atingido no primeiro modelo.

A heurística denominada *Biogeography Based Optimization* (BBO), Gupta et al. (2013), utiliza a heurística proposta por Ribeiro e Urrutia (2007) mas com algumas modificações para gerar uma boa programação de jogos inicial. Depois disso, é aplicada otimização baseada em biogeografia para melhorar as tabelas e encontrar o ótimo local de forma rápida. Finalmente, o SA é aplicado para melhorar as soluções encontradas. Os resultados obtidos foram piores, porém próximos aos melhores conhecidos até então, em que a maioria destes resultados se encontra no trabalho de Van Hentenryck e Vergados (2006). Em relação ao tempo de execução, o BBO-SA se mostrou mais rápido do que o método proposto por Van Hentenryck e Vergados (2006).

Os trabalhos descritos a seguir, Della Croce e Oliveri (2006), Knust e Lücking (2009), Ribeiro e Urrutia (2006) e Ribeiro e Urrutia (2012), propõem uma heurística, baseada em uma política *first-break, then-schedule* que consiste em três fases, sendo elas:

1. Gerar os HAPs;
2. Atribuir cada time a um HAP específico;
3. Gerar uma solução atribuindo os jogos aos times de acordo com os HAPs.

Della Croce e Oliveri (2006) apresentam uma aplicação prática da liga italiana de futebol (Serie A), que possui restrições adicionais de transmissão de TV e de times cabeças de chave, isto é, os melhores times do ano anterior não podem se enfrentar na primeira e na última semana do campeonato. A heurística apresentada tem o objetivo de minimizar os *breaks* e é baseada em programação linear inteira. Os resultados foram comparados com o calendário oficial da Serie A das temporadas de 2001, 2002 e 2003, em que o método proposto não violou nenhuma restrição de TV e obteve *breaks* menores em relação ao calendário oficial, que teve algumas restrições de TV não atendidas.

O trabalho de Knust e Lücking (2009) considera algumas soluções em que as fases 2 e 3 são resolvidas simultânea e separadamente. Para os experimentos, são utilizadas instâncias com e sem *place constraints* e com custos aleatórios. Estas instâncias foram criadas por Briskorn (2008). O tempo de execução apresentado é muito menor para as instâncias com esta restrição, pois alguns HAPs se tornam inviáveis e não são gerados. Para as instâncias sem *place constraints*, foi feita a comparação com Briskorn e Drexl (2009), em que os resultados obtidos foram muito parecidos.

Ribeiro e Urrutia (2006) e Ribeiro e Urrutia (2012) apresentam a aplicação do mTTP ao campeonato brasileiro dos anos de 2005 e 2006, em que os dois trabalhos possuem o objetivo de minimizar os *breaks*. Além disso, Ribeiro e Urrutia (2006) também considera transmissões de TV e tem como restrição adicional os times que são considerados elite, de acordo com o número de torcedores, participações anteriores e com os valores dos jogadores. Os resultados dos dois trabalhos foram bem melhores quando comparados aos do calendário oficial do campeonato, obtendo *breaks* bem menores para as duas instâncias.

2.2 Trabalhos Teóricos

Rasmussen e Trick (2008) apresentam uma bibliografia comentada sobre *Sports Scheduling*. Este trabalho é dividido em duas categorias, em que a primeira tem foco na minimização de *breaks* e a segunda na minimização da distância percorrida, uma vez que estes são os principais objetivos do mTTP tratados na literatura.

O trabalho de Kendall et al. (2010) tem foco em apresentar uma bibliografia comentada dos artigos que tratam do TTP e do mTTP, além de mostrar a importância dos esportes na mídia e nos negócios e o quanto os investimentos nas equipes são relevantes numa liga de esportes profissional.

Goossens e Spieksma (2012) apresentam um resumo sobre *Sport Scheduling* em ligas profissionais de futebol em 25 países da Europa na temporada de 2008-2009, mostrando o número de times de cada liga e a estrutura das tabelas dos campeonatos. Além disso, são apresentados os *breaks* obtidos na temporada em questão em cada uma das 25 ligas. O artigo também mostra uma definição de *breaks*, *canonical schedule* e efeito *carry-over* – efeitos que os times causam sobre os outros, como cansaço ou motivação pela vitória por exemplo, prejudicando ou beneficiando sua atuação nas partidas seguintes.

Ribeiro (2012) apresenta as definições dos problemas de *Sport Scheduling* e suas formulações, além de um *survey* de aplicações de métodos ótimos em ligas profissionais de diferentes esportes. O trabalho mostra também um estudo de uma aplicação prática do mTTP no campeonato brasileiro de anos anteriores.

Capítulo 3

Fundamentação Teórica

Este capítulo apresenta uma base conceitual do problema abordado neste trabalho, com suas restrições, instâncias, soluções e uma descrição matemática. Além disso, o método utilizado para a resolução do problema também é apresentado neste capítulo, em que são definidos todos os principais conceitos, as etapas que compõem o método e como ele pode ser aplicado ao problema para se alcançar os objetivos elencados anteriormente.

3.1 O *Traveling Tournament Problem*

Uma grande dificuldade presente nos campeonatos de esportes, é a elaboração de uma boa tabela de jogos. Neste contexto, tem-se o *Traveling Tournament Problem* (TTP), que está relacionado a campeonatos com caraterísticas DRRT e SRRT. Este trabalho tem foco em campeonatos que possuem dois turnos, com as equipes se enfrentando exatamente uma vez em cada turno (característica de um torneio DRRT).

Dentre as variações do TTP encontra-se a versão *Mirrored Traveling Tournament Problem*, que considera que o torneio é dividido em dois turnos, em que o segundo turno possui a mesma sequência de jogos do primeiro, porém, com os direitos de jogar em casa invertidos. Além disso, existem outras características que compõem o mTTP, sendo elas:

1. Os times jogam contra todos os outros uma vez em cada turno;
2. Cada time joga apenas uma vez em cada rodada;
3. Nenhum time pode jogar mais do que três partidas consecutivas com o mesmo padrão de mando de campo (ou seja, em casa ou fora);
4. Duas partidas com os mesmos adversários não podem ocorrer no mesmo turno.

Para resolver o mTTP, pode-se considerar apenas solucionar o primeiro turno como um torneio SRRT. Então, obtém-se o segundo turno através da solução encontrada para o primeiro,

invertendo o padrão de sequência de jogos em casa e fora dos times. Um exemplo de HAP é apresentado na Tabela 3.1, que se refere a um campeonato com quatro times brasileiros – Atlético Mineiro (CAM), Palmeiras (PAL) , Flamengo (FLA) e Grêmio (GRE) – indicando os jogos em casa (C) e fora (F) para cada time. Estes times se encontram em suas cidades no início do campeonato, saem em viagem para enfrentar os adversários e, então, retornam para suas cidades após no máximo três partidas consecutivas fora.

Tabela 3.1: Exemplo de HAPs para 4 times.

Times	Turno 1	Turno 2
CAM	F C C	C F F
PAL	F C F	C F C
FLA	C F F	F C C
GRE	C F C	F C F

Uma instância do mTTP com n times é representada por uma matriz de distâncias D_{nxn}. Cada elemento d_{ij} da matriz se refere à distância entre as cidades (ou estádios) dos times i e j. A Tabela 3.2 apresenta uma instância do mTTP de acordo com o exemplo utilizado. Pode-se observar que a distância entre os estádios de CAM e PAL é de 489 km, por exemplo, enquanto a distância entre os de FLA e GRE é de 1122 km.

Tabela 3.2: Exemplo de uma instância do mTTP.

	CAM	PAL	FLA	GRE
CAM	0	489	340	1340
PAL	489	0	358	852
FLA	340	358	0	1122
GRE	1340	852	1122	0

Além da distância, é considerada uma variável binária x_{ijk} cujo valor é definido da seguinte maneira:

$$x_{ijk} = \begin{cases} 1, & \text{se o time } i \text{ joga contra o time } j \text{ na rodada } k \\ 0, & \text{caso contrário} \end{cases}$$

Para gerar a tabela (calendário) do campeonato, considera-se um conjunto T com n times. Se n for par, o número m de rodadas será $2 \times (n-1)$, caso contrário, existirão $2 \times n$ rodadas. Considerando ainda o exemplo anterior, m será igual a 6 (onde $m = 2 \times (4-1)$). Os confrontos devem ser alocados de forma com que os times joguem exatamente uma vez em cada rodada. O calendário para este exemplo é apresentado na Tabela 3.3, em que as colunas representam as rodadas do torneio e as linhas representam os adversários dos time em cada rodada. O sinal de negativo na frente do time adversário significa que o jogo será fora de casa, como na

primeira linha por exemplo, o CAM jogará fora de casa contra o FLA na rodada 1 e depois jogará em casa contra o GRE na rodada 2.

Tabela 3.3: Exemplo de uma tabela de jogos.

Times	Rodadas					
	1	2	3	4	5	6
CAM	-FLA	GRE	PAL	FLA	-GRE	-PAL
PAL	-GRE	FLA	-CAM	GRE	-FLA	CAM
FLA	CAM	-PAL	-GRE	-CAM	PAL	GRE
GRE	PAL	-CAM	FLA	-PAL	CAM	-FLA

Uma solução do mTTP consiste em uma tabela (ou *timetable*) com todos os jogos do campeonato, respeitando as restrições e à qual está associada a distância total viajada pelos times, que pode ser calculada pela Função (3.1):

$$Z_{mTTP} = \sum_{k=0}^{m} \sum_{i=0}^{n} \sum_{j=0}^{n} x_{ijk} d_{ij} \tag{3.1}$$

É possível definir a função objetivo do mTTP conforme mostrado na Equação (3.2), que visa a minimização da distância total viajada pelos times.

$$\min Z_{mTTP} \tag{3.2}$$

3.2 Algoritmo Genético de Chaves Aleatórias Viciadas

Atualmente, existem diversos problemas práticos que estão relacionados à situações comuns do dia a dia, como por exemplo, problemas de geração de rotas mais curtas, controle de estoque, planejamento de produção e o problema de escalonamento que é tratado neste trabalho. Estes problemas estão incluídos em uma classe conhecida como Problemas de Otimização Combinatória, em que muitos deles são considerados difíceis por não possuírem uma solução em tempo aceitável.

Um dos motivos que dificultam a resolução de problemas de otimização, é a existência de um ótimo local, isto é, quando a solução chega em um ponto em que qualquer alteração leva a um resultado pior. Por outro lado, existe também o ótimo global, que corresponde ao melhor valor possível entre todos os ótimos locais dentro de um espaço de busca. Neste contexto, existem os métodos que utilizam estratégias de alto nível para ir além da otimalidade local

e aumentar as chances de chegar a um *ótimo global*, ou o mais próximo dele possível. Estes métodos são classificados como *Metaheurísticas*.

Algumas metaheurísticas se baseiam na teoria da evolução de *Darwin*, e são conhecidos como *Algoritmos Evolutivos*. A ideia por trás destas metaheurísticas é, em uma analogia com o conceito biológico, criar uma população de soluções que evoluirá ao longo de gerações (i.e., iterações do algoritmo), de maneira a otimizar o valor das soluções. Estes algoritmos utilizam o conceito de indivíduos que fazem parte de uma população, em que os melhores e mais adaptados são selecionados para participar de uma nova geração desta população.

Um indivíduo representa uma possível solução para o problema de otimização, enquanto a *função objetivo* estabelece a qualidade de cada indivíduo determinando se ele irá participar ou não da próxima geração e, dessa forma, melhorar o valor da solução final. Ainda, o conceito de *espaço de busca* é aplicado aos AGs, que é o conjunto de todas as soluções factíveis para o problema. Operadores também são utilizados como *movimentos* dentro do espaço de soluções, utilizados para transformar uma solução em outra, considerada *vizinha* desta. Diferentes movimentos definem diferentes *vizinhanças* dentro do espaço de busca, ou seja, delimitam uma determinada região do espaço de buscas em que soluções semelhantes podem ser encontradas.

Dentro da classe dos Algoritmos Evolutivos estão os *Algoritmos Genéticos* (AG), que têm como diferencial a utilização de *genes* e *cromossomos* para representar cada solução de uma população. A solução inicial consiste em um grupo de cromossomos (indivíduos) que são compostos por uma cadeia de valores, em que cada valor é representado por um gene. A geração de novas populações é feita através de operadores de recombinação (*crossover*) dos melhores cromossomos das gerações anteriores, o que intensifica a busca em uma determinada região de soluções. Este processo é realizado em pares de indivíduos denominados "pais" e que gera novos indivíduos "descendentes". O descendente receberá os genes dos dois cromossomos pais envolvidos, no intuito de que combine os melhores genes dos cromossomos pais.

Outro operador existente nos AGs é a *mutação*, em que um cromossomo pode ter seus genes alterados de maneira aleatória para gerar uma diversidade maior na nova população. O equilíbrio com que as populações são geradas, é estabelecido através dos parâmetros que definem as taxas de mutação e de *crossover*. A Figura 3.1 ilustra um esquema do funcionamento de um AG.

Dentre as variações dos AGs está o método Algoritmo Genético de Chaves Aleatórias (*Random-Key Genetic Algorithm*, RKGA), proposto por Bean (1994). Este método representa uma solução como um vetor de chaves aleatórias, em que cada uma dessas chaves possui um valor real dentro do intervalo [0,1) gerado aleatoriamente. É utilizado também um decodificador que mapeia a solução para o espaço de busca específico do problema tratado e calcula o valor desta solução. Além disso, o RKGA utiliza o conceito de elitismo, em que os melhores indivíduos de cada geração do AG vão pertencer a um conjunto chamado elite, que será inteiramente copiado para a próxima geração.

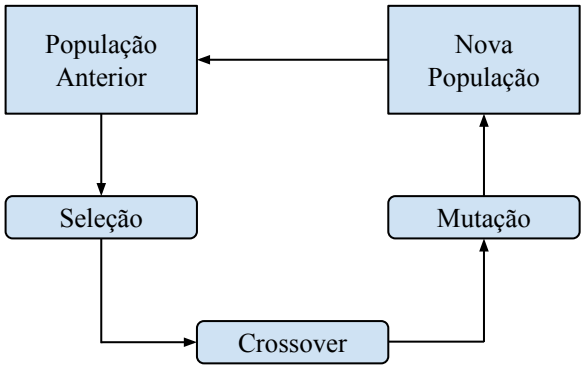

Figura 3.1: Esquema de um AG.

O restante da nova população é gerado através do *crossover* e da mutação. Neste método, o *crossover* não faz distinção entre elite e não elite, isto é, a recombinação pode ser feita a partir de qualquer par de indivíduos da população. Ainda, a mutação consiste em criar os indivíduos novamente de forma aleatória, diferente dos algoritmos genéticos clássicos que apenas alteram um gene do indivíduo. Os parâmetros que representam o tamanho do conjunto elite e as taxas de *crossover* e mutação afetam diretamente no valor da solução e, dependendo do problema, podem assumir diferentes valores.

Outra variação de AGs é o método Algoritmo Genético de Chaves Aleatórias Viciadas (*Biased Random-Key Genetic Algorithm*, BRKGA), proposto por Gonçalves e Resende (2011). Esta metaheurística é inspirada no RKGA, no qual ambos os métodos possuem as mesmas operações. A diferença entre eles está na forma com que os cromossomos pais são selecionados para o cruzamento e como este cruzamento é realizado. No BRKGA, os pais também são escolhidos de forma aleatória, porém, um deles deve obrigatoriamente pertencer ao conjunto de elite e o outro não. Ainda na fase de *crossover*, os genes do cromossomo pai vindo da elite devem ter maior probabilidade de serem escolhidos sobre os genes do pai não-elite, de modo que este influenciará mais a composição do cromossomo descendente. As etapas do BRKGA são descritas e exemplificadas nas seções a seguir.

3.2.1 Codificação

O indivíduo ou cromossomo é formado por um vetor de n chaves aleatórias, em que cada posição do vetor representa um gene, com uma chave de valor real aleatória entre 0 e 1. A utilização da aleatoriedade nas chaves dos cromossomos ajuda a manter a diversidade das soluções. O processo de codificação é importante para que não haja uma dependência entre a metaheurística e o problema tratado, fazendo com que o usuário do BRGKA se concentre apenas na etapa de decodificação, além de permitir o reuso do software em outros problemas. Um exemplo de um cromossomo com 4 chaves é apresentado na Figura 3.2.

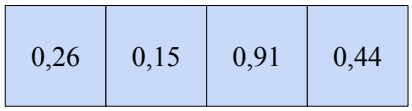

Figura 3.2: Exemplo de um cromossomo com 4 chaves.

3.2.2 Decodificação

Esta etapa é a que possui mais dependência com o problema tratado. O decodificador tem o papel de mapear o cromossomo a partir do espaço de chaves aleatórias para o espaço de solução do problema. Depois disso, os valores da função objetivo de cada solução (indivíduo) são calculados. Estes valores são avaliados por meio do conceito de *fitness*, que se refere ao nível de qualidade dos resultados de cada cromossomo. A Figura 3.3 ilustra um decodificador mapeando uma solução para o espaço de soluções específico do problema.

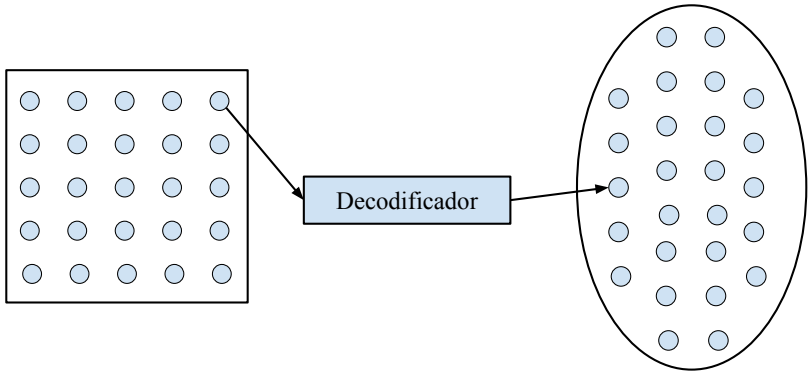

Figura 3.3: Decodificador mapeando uma solução para o espaço de soluções do problema. Adaptado de Gonçalves e Resende (2011).

3.2.3 População Inicial

A solução inicial é composta por m vetores (cromossomos) que foram gerados aleatoriamente na fase de codificação. As soluções são ordenadas de acordo com o *fitness* (ou valor da função objetivo) a partir desta população inicial, de modo que os melhores indivíduos são selecionados como elite, separando-os dos demais cromossomos que passarão pelo processo de recombinação (*crossover*) para que a nova população seja gerada.

3.2.4 Elitismo

Os melhores indivíduos de cada geração que foram selecionados como elite, serão copiados sem nenhuma alteração para a próxima população. A estratégia de elitismo serve para manter o controle de boas soluções que são encontradas durante as iterações do BRKGA. De acordo com Gonçalves e Resende (2011), para se obter bons resultados o tamanho do conjunto de

elite deve estar entre 10% e 25% da população total. A Figura 3.6 ilustra como os indivíduos da elite são alocados na nova população.

3.2.5 Mutação

A nova população possui uma pequena parte que é composta por indivíduos classificados como mutantes. Estes são novos indivíduos gerados de maneira aleatória para gerar diversidade na nova população e evitar que ela convirja para um ótimo local prematuramente. Além disso, gerar um mutante aleatório faz com que as soluções de piores resultados sejam totalmente substituídas, eventualmente por indivíduos melhores.

Diferente dos AGs tradicionais, que utilizam valores baixos para taxas de mutação, um tamanho da população mutante considerado bom no BRKGA está entre 10% e 30% da população total, de acordo com Gonçalves e Resende (2011). Estes valores interferem na velocidade com que a solução ótima pode ser encontrada, em que os valores fora deste intervalo apresentaram resultados ruins nos testes realizados pelos autores. Além disso, como a probabilidade de um indivíduo da elite ser escolhido para a próxima população é alta, a tendência é que as novas gerações convirjam sempre para a elite. Por este motivo, é importante que a taxa de mutação seja mais alta para que haja um equilíbrio maior nas próximas gerações. A Figura 3.6 mostra como os indivíduos mutantes estão dispostos na nova população.

3.2.6 *Crossover*

O restante da nova população é gerado a partir da recombinação de pares de indivíduos, no qual um deles pertence à elite e o outro não. O indivíduo descendente é composto pelos genes dos dois cromossomos envolvidos, em que, para cada gene, é sorteado um valor entre 0 e 100 para saber de qual pai o descendente irá herdar. Como a chance de escolher o indivíduo da elite é maior (o que caracteriza as chaves como viciadas), por exemplo 75%, se o número sorteado for menor que 75 o indivíduo irá herdar o gene do cromossomo da elite, caso contrário, o gene é herdado do outro cromossomo. Gonçalves e Resende (2011) recomendam que a probabilidade de um gene ser herdado do indivíduo da elite seja entre 50% e 80%. A Figura 3.4 representa como o *crossover* é realizado.

No processo de *crossover* ilustrado na Figura 3.4, o cromossomo 1 pertence à elite e cromossomo 2 não pertence. Como apresentado na figura, a probabilidade p de escolher o gene do indivíduo da elite é de 70% ($p = 70$). O primeiro número aleatório gerado é 58 que é menor que p, então o primeiro gene do descendente é herdado do cromossomo 1. O segundo número aleatório é 89, e como este número é maior que p, o segundo gene do descendente tem origem do cromossomo 2. Os dois últimos números gerados, 68 e 25, são menores que p, por isso os respectivos genes do descendentes vêm do cromossomo 1.

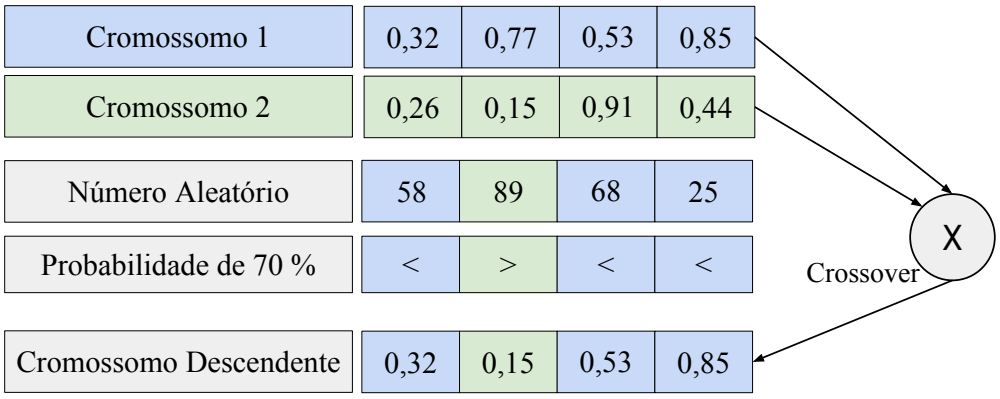

Figura 3.4: Exemplo de *crossover*. Adaptado de Gonçalves e Resende (2011).

3.2.7 Visão Geral

Como citado anteriormente, O BKRGA possui algumas características específicas que o difere dos outros AGs, como por exemplo a representação de um indivíduo como um vetor de chaves aleatórias e a maior probabilidade de um descendente herdar os genes de um indivíduo elite. Estes e outros conceitos são importantes nas etapas que compõem o BRKGA e fazem com que ele supere alguns outros métodos em relação ao desempenho. O esquema apresentado na Figura 3.5 ilustra o fluxo de execução do BRKGA com todas as etapas detalhadas anteriormente.

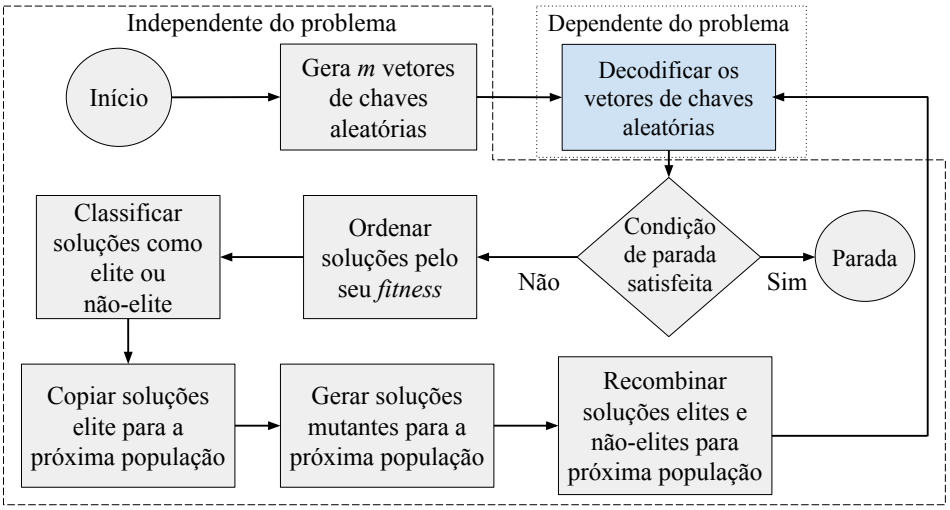

Figura 3.5: Fluxo de execução do BRKGA. Adaptado de Gonçalves e Resende (2011).

A disposição dos indivíduos a cada geração é apresentada na Figura 3.6, que mostra como o conjunto elite é copiado para a próxima geração e como os indivíduos mutantes estão dispostos na nova população. Além disso, a figura ilustra a seleção dos pais que irão participar do *crossover*, em que um deles pertence obrigatoriamente à elite enquanto o outro pertence aos

não-elite.

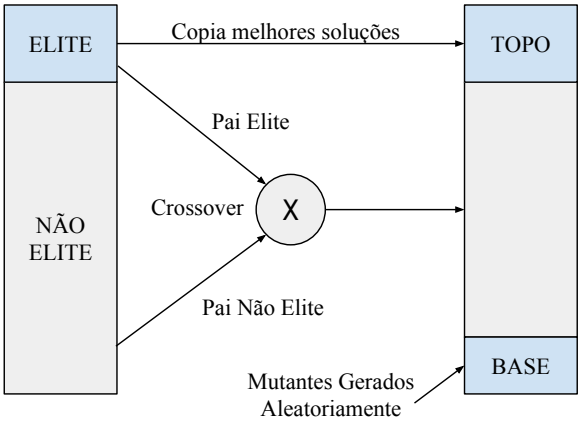

Figura 3.6: Esquema da geração da nova população. Adaptado de Gonçalves e Resende (2011).

Capítulo 4

Algoritmo Genético de Chaves Aleatórias Viciadas aplicado do Problema de Escalonamento de Competições Esportivas

Neste capítulo são apresentadas em detalhes as adaptações realizadas no BRKGA para a aplicação ao mTTP, tais como a representação dos cromossomos, a decodificação, o *crossover* e a verificação da viabilidade de uma solução. São apresentados também exemplos para facilitar o entendimento do que foi alterado nos processos do BRKGA.

Foi utilizada a biblioteca (ou *Application Programming Interface*, API) do BRKGA, proposta por Toso e Resende (2015), para facilitar a aplicação deste método ao problema tratado. As adaptações foram realizadas na implementação desta API.

4.1 Representação Computacional

Como mencionado no Capítulo 3, as informações de entrada para o mTTP são dadas em forma de uma matriz, em que cada elemento desta é um número inteiro que representa a distância entre os estádios (ginásios) de cada time.

Inicialmente, as equipes não são representadas pelos seus nomes reais, mas por números inteiros no intervalo $[0,\ n)$, em que n é o número de times do torneio. Os confrontos entre as equipes são representados de maneira semelhante, porém, o intervalo é $[1,\ m]$, sendo m o número de partidas possíveis, em que $m = n \times (n-1)/2$. Para 4 times (0, 1, 2 e 3), por exemplo, temos os 6 jogos possíveis representados da seguinte maneira:

- Jogo 1: 0×1;

- Jogo 2: 0×2;
- Jogo 3: 0×3;
- Jogo 4: 1×2;
- Jogo 5: 1×3;
- Jogo 6: 2×3.

Através desta padronização dos jogos, é construído um grafo de conflitos que define quais jogos podem ou não acontecer em uma mesma rodada. Este conflito se refere à situação em que o mesmo time joga mais de uma vez na mesma rodada. O grafo é representado por uma matriz binária C, em que $C_{ij} = 1$ se existe conflito entre os jogos i e j em questão (linha i e coluna j) e $C_{ij} = 0$ caso contrário. No exemplo citado, o Jogo 1 (0×1) não pode acontecer na mesma rodada que o Jogo 3 (0×3), pois assim, o time 0 estaria jogando duas vezes na mesma rodada. A matriz de conflitos para 4 equipes é representada na Tabela 4.1 e o grafo de conflitos é apresentado na Figura 4.1, em que os vértices representam os jogos e as arestas indicam o conflito entre os jogos, ou seja, eles não podem acontecer na mesma rodada.

	1	2	3	4	5	6
1	1	1	1	1	1	0
2	1	1	1	1	0	1
3	1	1	1	0	1	1
4	1	1	0	1	1	1
5	1	0	1	1	1	1
6	0	1	1	1	1	1

Tabela 4.1: Exemplo da matriz de conflitos dos jogos para 4 times.

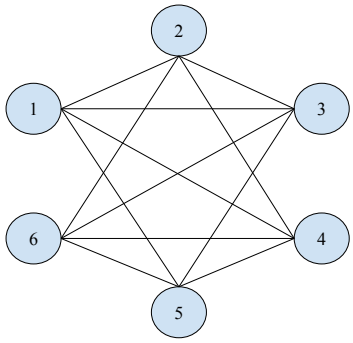

Figura 4.1: Grafo de conflito dos jogos para 4 times.

A representação do cromossomo utilizada neste trabalho sofre uma pequena alteração em relação ao cromossomo padrão do BRKGA. Por este motivo, foi necessário alterar a forma com que os cromossomos são gerados na API. Ao invés de chaves aleatórias entre 0 e 1,

como descrito no Capítulo 3, cada gene do cromossomo é representado por um número inteiro seguido de uma parte decimal, conforme apresentado na Figura 4.2.

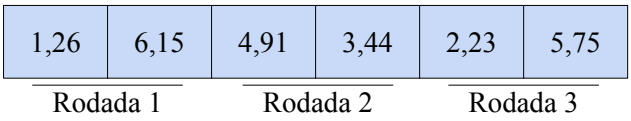

Figura 4.2: Representação do cromossomo neste trabalho.

Foi utilizado um vetor auxiliar com números inteiros no intervalo $[1, m]$ em ordem aleatória, que representam os índices dos jogos. Os elementos deste vetor são copiados para o novo indivíduo, considerando que este obedece uma segmentação lógica em rodadas cujo tamanho é definido de acordo com o número de times, como apresentado na Figura 4.2. A cada elemento do vetor auxiliar a ser copiado, é verificado se existe conflito entre ele e os que já foram adicionados dentro de uma mesma rodada. Caso exista conflito, tenta-se copiar o próximo elemento do vetor auxiliar. No início de cada rodada do novo indivíduo não é necessário verificar se há conflito, uma vez que nenhum jogo foi adicionado a esta rodada. O processo apresentado é repetido até que o cromossomo seja totalmente preenchido e, ao final, são adicionadas as partes decimais aleatórias a cada elemento copiado, gerando assim um novo cromossomo.

No caso de ocorrerem conflitos que impeçam que um cromossomo seja completamente preenchido, é utilizada a conhecida técnica de *backtracking* que, neste contexto, consiste em remover a última rodada adicionada ao cromossomo e retornar os índices destes jogos ao vetor auxiliar. Depois disso, o processo citado anteriormente é repetido envolvendo os próximos elementos do vetor auxiliar, gerando o cromossomo de maneira diferente da que resultou em conflitos.

4.2 Decodificação

Para decodificar os cromossomos, considera-se que a parte inteira da chave aleatória representa o índice do jogo e a parte decimal define qual time joga em casa e fora. Por exemplo, no jogo 1, que é o confronto entre os times 0 e 1, se a parte decimal for maior ou igual a 0,5 o time 0 joga em casa e o time 1 fora. Caso contrário, o time 1 joga em casa e o time 0 fora. No exemplo da Figura 4.2, a primeira rodada é composta pelos jogos 1 e 6, a segunda rodada pelos jogos 4 e 3 e a terceira pelos jogos 2 e 5. De acordo com os jogos e com os valores das partes decimais, tem-se o primeiro turno definido da seguinte maneira, considerando que o primeiro time relacionado em cada jogo, disputa a partida em casa:

- Rodada 1: 1×0 e 3×2;

- Rodada 2: 1×2 e 3×0;
- Rodada 3: 2×0 e 1×3.

O Algoritmo 1 apresenta o processo que transforma os dados contidos em um cromossomo em configurações de jogos na tabela que representa a solução. Para cada gene do cromossomo (laço das linhas 1 a 11), isola-se a parte decimal da chave através da função *parteDecimal* (linha 2). Depois disso, é verificado se este valor é maior ou igual a 0,5 (linha 3) para definir qual time joga em casa e qual joga fora (linhas 4-5 e 8-9). As variáveis *time1* e *time2* se referem às equipes que se enfrentam em um determinado jogo.

Algoritmo 1: Decodificador

```
   Entrada: cromossomo[m]
 1 para i ← 0 até m − 1 faça
 2 |   decimal ← parteDecimal(cromossomo[i]);
 3 |   se decimal >= 0,5 então
 4 |   |   timeCasa ← time1;
 5 |   |   timeFora ← time2;
 6 |   fim
 7 |   senão
 8 |   |   timeCasa ← time2;
 9 |   |   timeFora ← time1;
10 |   fim
11 fim
```

Uma vez que o primeiro turno é decodificado, repetem-se as mesmas operações no segundo turno, tendo assim a decodificação de uma solução completa. Para isto, é utilizado o mesmo procedimento descrito no Algoritmo 1, porém, invertendo o sinal da comparação da linha 3 para obter o mando de campo invertido. Ainda no exemplo da Figura 4.2, tem-se o segundo turno definido da seguinte maneira:

- Rodada 4: 0×1 e 2×3;
- Rodada 5: 2×1 e 0×3;
- Rodada 6: 0×2 e 3×1.

4.3 Função de Avaliação

Para avaliar um cromossomo, é considerado que os times começam a competição em suas sedes e, posteriormente, realizam as viagens para os estádios (ginásios) dos adversários. Ademais, outros casos especiais devem ser considerados, como a passagem do primeiro turno para o segundo e a última rodada do torneio, em que os times devem retornar para suas sedes. É

utilizado um vetor que armazena a localização atual de cada time a cada jogo para manter o controle das viagens realizadas pelas equipes ao calcular a distância percorrida.

Além destas considerações especiais, existe uma restrição nos torneios esportivos considerados que não permite que as equipes realizem mais do que três partidas consecutivas em casa ou fora. Para tratar esta restrição, foi adicionado um contador à cada time para controlar o número de partidas realizadas, o qual é incrementado a cada jogo consecutivo em casa ou fora. Se este contador atingir o valor 4, significa que esta restrição está sendo violada. O Algoritmo 2 mostra como o processo de viabilização de uma solução é realizado. As variáveis *contadorTime1* e *contadorTime2* se referem aos contadores de jogos consecutivos em casa ou fora para cada um dos times analisados.

Algoritmo 2: Viabilização de uma solução

```
   Entrada: cromossomo[m]
 1 para i ← 0 até (2 × m) − 1 faça
 2 |   contaPartidas(time1, time2);
 3 |   se contadorTime1 > 3 ou contadorTime2 > 3 então
 4 |   |   trocaMandoDeCampo();
 5 |   |   contaPartidas(time1, time2);
 6 |   |   se contadorTime1 > 3 ou contadorTime2 > 3 então
 7 |   |   |   penalizaSolucao();
 8 |   |   fim
 9 |   fim
10 fim
```

Para cada um dos jogos da tabela do campeonato, incluindo os dois turnos (laço das linhas 1-10), a função *contaPartidas* (linhas 2 e 5) realiza a contagem dos jogos consecutivos em casa ou fora dos dois times da seguinte maneira: se um time joga uma partida em casa, por exemplo, seu contador recebe o valor 1 e, se o próximo jogo também for em casa, este contador é incrementado para o valor 2. Este valor é incrementado enquanto o time estiver em uma sequência de jogos em casa. Caso o próximo jogo seja fora de casa, o contador é reiniciado e assume o valor 1 novamente. A contagem dos jogos consecutivos fora de casa é realizada da mesma maneira.

Se o contador de um dos times assumir um valor maior que 3 (linha 3), a função *trocaMandoDeCampo* é executada (linha 4). Esta função consiste em inverter o mando de campo da partida alterando a parte decimal da chave aleatória em questão, ou seja, o gene do cromossomo referente à partida é alterado. Se este valor for maior que 0,5 basta subtrair dele 0,5. Caso contrário, adiciona-se 0,5 ao valor. Depois disso, o processo descrito no Algoritmo 1 é repetido para poder inverter o mando de campo do jogo.

Os contadores são atualizados através da função *contaPartidas* (linha 5) e então verificados novamente (linha 6), pois, ao corrigir a sequência de jogos de um time, é possível que o seu ad-

versário passe a violar a restrição, sendo então impraticável realizar a troca novamente porque voltaríamos para a mesma situação com o primeiro time. Neste caso, em que um dos times sempre irá jogar mais do que 3 partidas consecutivas em casa ou fora, a função *penalizaSolucao* (linha 7) é executada para aplicar uma penalização ao valor da solução, adicionando-se um valor equivalente à soma da matriz de distâncias, com o intuito de isolar as solução inviáveis para que elas não sejam selecionadas para as próximas gerações. A Figura 4.3 exemplifica uma situação em que ocorre este tipo de conflito.

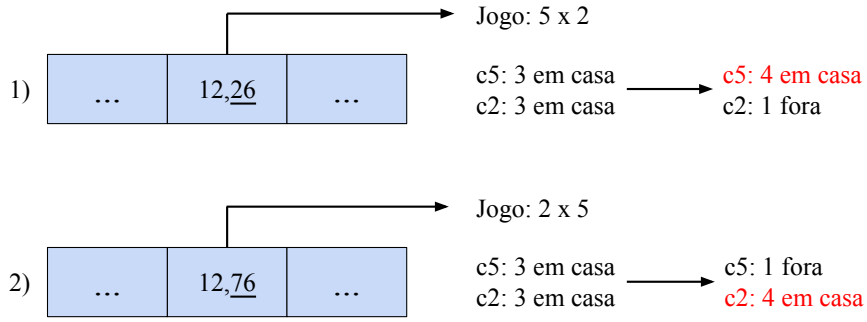

Figura 4.3: Exemplo de um conflito ao avaliar uma solução.

Neste exemplo, em um determinado momento da avaliação, tem-se uma chave cujo valor representa o jogo 12 (confronto entre os times 5 e 2), e considera-se que os contadores desses times (*c5* e *c2*) já estão com o valor de 3 jogos consecutivos em casa, decorrente de genes anteriores. No momento 1, ao computar mais um jogo em casa para o time 5, este totalizaria 4 jogos consecutivos em casa, o que caracteriza a inviabilidade da solução. Porém, no momento 2, ao corrigir este gene adicionando 0,5 à sua chave e invertendo o mando de campo, o contador do time 2 atinge o valor 4. Desta forma, tem-se uma situação em que um dos dois times sempre irá tornar a solução inviável, em que é necessário aplicar a penalização.

4.4 *Crossover*

Como a forma de representar o cromossomo no BRKGA foi alterada, o *crossover* também precisou passar por algumas modificações. Ao escolher os genes dos pais a serem copiados para o filho, é necessário verificar se o jogo em questão já foi adicionado anteriormente, para evitar que o mesmo jogo não ocorra mais de uma vez no mesmo turno. Para isto, foi utilizado um vetor auxiliar de inteiros que armazena quais jogos já foram adicionados no cromossomo filho e, a cada jogo selecionado, é verificado se ele já existe no vetor. Além disso, para evitar que haja conflitos entre os jogos dentro de uma mesma rodada, o *crossover* proposto seleciona uma rodada completa por vez, isto é, um bloco de genes a cada iteração. Este processo é realizado de maneira diferente do *crossover* do BRKGA padrão, que seleciona um gene por vez de cada pai para gerar o filho.

A seleção das rodadas é feita através de índices nos cromossomos pais e no filho, que controlam qual rodada irá ser copiada e em qual posição ela irá entrar no cromossomo filho em cada iteração. Inicialmente, os índices apontam para a primeira rodada dos pais e do filho e avançam de acordo com a seleção dos pais no cruzamento. Quando um pai é selecionado, a rodada indicada pelo seu índice é copiada para o filho e os índices do filho e do pai em questão avançam para a próxima rodada, enquanto o índice do outro pai permanece onde está. A Figura 4.4 mostra como o cromossomo filho é gerado no processo de *crossover* proposto.

Figura 4.4: Operação de *crossover* neste trabalho.

No exemplo mostrado na Figura 4.4, os primeiros genes a serem escolhidos são os da primeira rodada (R1) dos pais. O pai da elite é selecionado, então, a R1 deste (as chaves 1,59 e 6,31) é copiada para o filho. Com isso, os índices do filho e do pai elite vão para R2. Neste momento, a escolha é entre R2 do pai elite e R1 do pai não elite. O pai não elite é escolhido desta vez, então, a R1 deste pai é copiada para a R2 do filho e o índice deste mesmo pai avança para R2. Da mesma forma, a escolha passa a ser entre R2 dos dois pais, sendo que o da elite é selecionado novamente, copiando sua rodada R2 para a R3 do filho, preenchendo por completo este cromossomo, finalizando então o processo de *crossover*.

O conflito nesta etapa ocorre quando a rodada escolhida para ser copiada possui um ou mais jogos que já foram adicionados ao filho anteriormente. Neste caso, o índice do pai que contém esta rodada que gera o conflito é avançado para a próxima rodada e o sorteio do pai é feito novamente. Este processo se repete até que o cromossomo filho seja completamente preenchido. Se este conflito ocorrer de forma que não seja possível completar o filho, o processo de *crossover* é reiniciado alterando o cromossomo pai que foi escolhido inicialmente. Para isto, na primeira iteração do *crossover*, o pai selecionado (elite ou não elite) é armazenado para que, no reinício, ele não seja selecionado de novo na primeira iteração, a fim de iniciar o processo de maneira diferente da que resultou em conflito. Se o processo de reinício se repetir um determinado número de vezes sem sucesso, o *crossover* é interrompido para evitar um possível *loop* infinito, e o cromossomo filho recebe uma penalização equivalente à soma total da matriz de distâncias. O Algoritmo 3 apresenta o processo de execução do *crossover* proposto.

Algoritmo 3: Crossover

```
   Entrada: paiElite[m], paiNaoElite[m]
 1 para indiceFilho ← 0 até m-1 faça
 2 |   paiOrigem ← selecionaPai(paiElite, paiNaoElite);
 3 |   se indiceFilho = 0 então
 4 |   |   primeiroPai ← paiOrigem;
 5 |   fim
 6 |   se reinicio = verdadeiro então
 7 |   |   zeraIndices(indiceElite, indiceNaoElite, indiceFilho);
 8 |   |   se primeiroPai = paiElite então
 9 |   |   |   paiOrigem ← paiNaoElite;
10 |   |   fim
11 |   |   senão
12 |   |   |   paiOrigem ← paiElite;
13 |   |   fim
14 |   |   reinicio ← falso;
15 |   |   contReinicio ← 0;
16 |   fim
17 |   se conflito(paiOrigem, filho) = falso então
18 |   |   para j ← 0 até tamanhoRodada faça
19 |   |   |   filho[indiceFilho] ← paiOrigem[indicePaiOrigem]
20 |   |   |   indicePaiOrigem ← indicePaiOrigem + 1;
21 |   |   |   indiceFilho ← indiceFilho + 1;
22 |   |   fim
23 |   fim
24 |   senão
25 |   |   indicePaiOrigem ← indicePaiOrigem + tamanhoRodada;
26 |   fim
27 |   se indiceElite = m e indiceNaoElite = m e indiceFilho < m então
28 |   |   reinicio ← verdadeiro;
29 |   |   contReinicio ← contReinicio + 1;
30 |   fim
31 |   se contReinicio = MaxIteracao então
32 |   |   penalizaSolucao(filho);
33 |   fim
34 fim
```

Para cada gene do cromossomo (laço das linhas 1-34), é feita a escolha do pai de origem através da função *selecionaPai* (linha 2), o qual é armazenado na primeira iteração (linha 4) para ser utilizado posteriormente caso aconteça o reinício. Através da função *conflito* (linha 17), é verificado se existe conflito entre os jogos da atual rodada do pai de origem selecionado e os jogos que já foram adicionados ao filho. Caso não haja conflito, a rodada indicada pelo índice do pai de origem é copiada para a rodada atual do filho de gene a gene (linha 19) e, da mesma maneira, os índices do pai de origem e do filho são avançados para indicar a próxima

rodada (linhas 20-21). Caso haja conflito, é adicionado um valor igual ao tamanho da rodada para que o índice do pai selecionado avance para a rodada seguinte (linha 25).

Se os dois pais forem percorridos até o final e o filho não for completamente preenchido (linha 27), a variável *reinicio* recebe o valor verdadeiro (linha 28), indicando que o processo de *crossover* será reiniciado na próxima iteração. Além disso, o contador *contReinicio* é incrementado cada vez que o reinício acontece (linha 29) e, quando este contador atinge um valor máximo pré-definido (representado pela constante *MaxInteracao*, linha 31), o cromossomo filho é penalizado atravéz da função *penalizaSolucao*, que adiciona a soma total da matriz de distâncias ao seu valor de solução.

A cada iteração, é verificado se ocorre o reinício (linha 6), que consiste em zerar os índices dos pais e do filho através da função *zeraIndices* (linha 7), para que eles voltem a indicar a primeira rodada dos indivíduos envolvidos no cruzamento. Depois disso, é verificado qual dos pais foi selecionado na primeira iteração (linha 8). Caso este tenha sido o pai da elite, o *crossover* será reiniciado tendo o pai não elite como o pai de origem (linha 9). Caso contrário, o pai elite será selecionado para reiniciar o processo (linha 12). Por fim, o indicador *reinicio* é atualizado com o valor falso e o contador *contReinicio* recebe o valor zero (linhas 14-15), reiniciando então o processo de *crossover*.

4.5 Mutação

O processo original de mutação do BRKGA consiste basicamente em gerar novos indivíduos totalmente aleatórios, como explicado na Seção 3. Neste trabalho, a mutação ocorre de maneira diferente, em que os cromossomos mutantes são gerados a partir de um movimento denominado *Sequence Round Swap* (SRS) aplicado a indivíduos já existentes na população. Este método se mostrou melhor em relação ao tempo de execução e aos resultados obtidos, quando comparado à mutação original do BRKGA. O SRS é exemplificado na Figura 4.5.

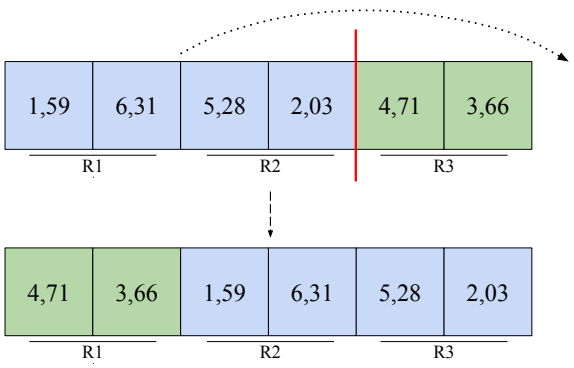

Figura 4.5: Movimento de *Sequence Round Swap*.

Primeiramente, seleciona-se um indivíduo aleatório da população, pertencente ao conjunto

elite ou não. Então, uma rodada é selecionada, também de forma aleatória (rodada 2 no exemplo da Figura 4.5) para ser o ponto de corte do SRS. Isto significa que, todas as rodadas que antecedem a rodada do ponto de corte, incluindo ela, serão realocadas para o final do cromossomo. A última rodada nunca é selecionada como ponto de corte, pois não haveria alteração alguma no indivíduo.

4.6 Buscas Locais Aplicadas ao mTTP

Na tentativa de melhorar os indivíduos gerados em uma nova população, foram utilizados diferentes métodos de busca local, que consistem em alterar partes de um indivíduo de forma aleatória e então recalcular o seu valor de solução (*fitness*). Estes métodos foram aplicados ao final de cada geração de uma nova população, sendo que apenas os cromossomos do conjunto elite foram submetidos à aplicação destes métodos. Por isso, foram consideradas apenas as alterações que melhoraram o *fitness* dos indivíduos, para evitar que a população convirja para um resultado ruim. Os métodos utilizados são *Home-Away Swap* (HAS), *Round Swap* (RS) e *Team Swap* (TS), os quais são descritos e exemplificados nas seções a seguir.

4.6.1 *Home-Away Swap* (HAS)

Este método consiste na simples troca do direito de jogar em casa em uma partida aleatória da competição. A Figura 4.6 apresenta um exemplo do HAS, cujo jogo selecionado é o de índice 2, entre os times 1 e 3 com o time 1 jogando em casa (1×3).

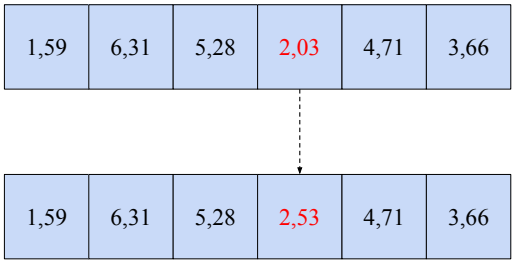

Figura 4.6: Movimento de *Home-away Swap*.

O método HAS inverte o mando de campo das equipes, fazendo com que o time 3 passe a jogar em casa e o time 1 fora (3×1). O Algoritmo 4 apresenta o funcionamento do HAS.

Inicialmente, é gerado um número aleatório que representa o índice do jogo a ser selecionado, através da função *numeroAleatorio* (linha 1). Com isso, retira-se a parte decimal do gene indicado por *indiceJogo* para se obter o mando de campo da partida em questão (linha 2). Verifica-se então se a parte decimal é maior ou igual a 0,5 (linha 3) e, se for, basta subtrair 0,5 para alterar o mando de campo (linha 4), caso contrário, adiciona-se 0,5 (linha 7).

Algoritmo 4: Home-away Swap

Entrada: *cromossomo*[*m*]

```
indiceJogo ← numeroAleatorio();
decimal ← parteDecimal(cromossomo[indiceJogo]);
se decimal >= 0,5 então
    cromossomo[indiceJogo] ← cromossomo[indiceJogo] - 0,5;
fim
senão
    cromossomo[indiceJogo] ← cromossomo[indiceJogo] + 0,5;
fim
```

4.6.2 *Round Swap* (RS)

O método RS consiste basicamente em selecionar duas rodadas aleatórias e trocá-las de posição no cromossomo, no intuito de alterar a ordem em que as partidas são realizadas, o que possibilita a redução da distância viajada pelas equipes. A Figura 4.7 ilustra como essa troca de rodadas é realizada.

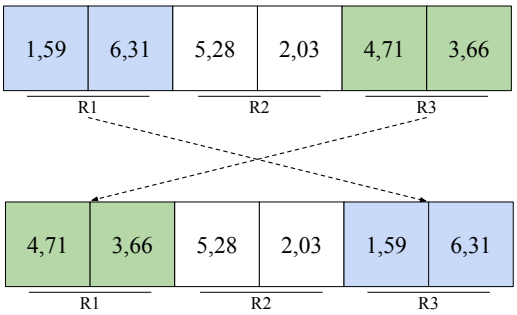

Figura 4.7: Movimento de *Round Swap*.

No exemplo da Figura 4.7, as rodadas escolhidas foram R1 e R3. Depois disso, o bloco de genes que representa a R3 passa a representar a R1, fazendo com que os jogos que seriam realizados na terceira rodada sejam realizados na primeira. Da mesma forma, os jogos da primeira rodada (R1) agora são realizados na terceira (R3). O Algoritmo 5 apresenta como o RS é realizado.

Algoritmo 5: Round Swap

Entrada: *cromossomo*[*m*], k

```
rodadaX ← rodadaAleatoria(cromossomo);
rodadaY ← rodadaAleatoria(cromossomo);
rodadaAuxiliar[k];
rodadaAuxiliar ← rodadaX;
rodadaX ← rodadaY;
rodadaY ← rodadaAuxiliar;
```

Primeiramente, selecionam-se as duas rodadas a serem trocadas através da função *rodadaAleatoria* (linhas 1 e 2) que gera um índice aleatoriamente para indicar uma rodada. Depois, é criado um vetor *rodadaAuxiliar* (linha 3) de tamanho k, que indica o tamanho de uma rodada, para ser utilizado na troca. Então, armazena-se a *rodadaX* no vetor auxiliar (linha 4), troca-se a *rodadaX* pela *rodadaY* (linha 5) e, por fim, a rodada auxiliar que contém os jogos da *rodadaX* é atribuída à *rodadaY* (linha 6).

4.6.3 *Team Swap* (TS)

O método TS consiste em trocar a tabela de jogos de dois times aleatórios no campeonato, isto é, trocar os adversários destes times em todas as rodadas, exceto naquelas em que os dois times se enfrentam, o que não surtiria efeito algum. A Figura 4.8 exemplifica a troca entre as tabelas dos times 2 e 4, de forma que, nas partidas em que o time 2 joga, jogará o time 4 com o mesmo adversário. Da mesma maneira, o time 2 é alocado para as partidas em que o time 4 jogaria. Com isso, a ordem dos jogos no cromossomo é alterada, com o intuito de reduzir a distância viajada.

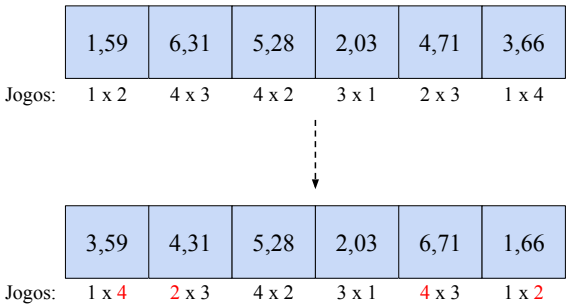

Figura 4.8: Movimento de *Team Swap*.

Pode ser observado na Figura 4.8, que todos o jogos listados no cromossomo inicial que possuem os times 2 ou 4, são trocados. Por exemplo, o primeiro jogo do cromossomo é entre os times 1 e 2. Ao realizar o TS, o primeiro jogo do cromossomo agora será entre os times 1 e 4. Já no terceiro jogo, entre os times 2 e 4, não é realizada a troca, pois se refere à partida em que 2 e 4 se enfrentam. O Algoritmo 6 apresenta o movimento TS.

Primeiramente, são selecionados de forma aleatória os dois times a serem trocados (linhas 1 e 2). Então, para cada posição do cromossomo (laço das linhas 3–21), retira-se a parte inteira do gene para detectar o jogo, armazenando este valor na variável *indiceJogo* (linha 4). O mesmo é feito com a parte decimal (linha 5), que será utilizada posteriormente. Através do índice do jogo obtém-se os índices das equipes que se enfrentam nesta partida, atribuindo-os às variáveis *t1* e *t2* (linhas 6 e 7). Tendo isto, é verificado se uma das equipes que se enfrentam no jogo em questão é uma das sorteadas para troca e, ao mesmo tempo, se o seu adversário não foi sorteado (linhas 8, 11, 14 e 17). Caso esta condição seja verdadeira, os índices de um

Algoritmo 6: Team Swap

```
   Entrada: cromossomo[m], jogos[m]
 1 timeX ← timeAleatorio(cromossomo);
 2 timeY ← timeAleatorio(cromossomo);
 3 para i ← 0 até m-1 faça
 4 |   indiceJogo ← parteInteira(cromossomo[i]);
 5 |   decimal ← parteDecimal(cromossomo[i]);
 6 |   t1 ← jogos[indiceJogo].time1;
 7 |   t2 ← jogos[indiceJogo].time2;
 8 |   se t1 = timeX e t2 != timeY então
 9 |   |   t1 ← timeY;
10 |   fim
11 |   se t1 = timeY e t2 != timeX então
12 |   |   t1 ← timeX;
13 |   fim
14 |   se t2 = timeX e t1 != timeY então
15 |   |   t2 ← timeY;
16 |   fim
17 |   se t2 = timeY e t1 != timeX então
18 |   |   t2 ← timeX;
19 |   fim
20 |   cromossomo[i] = novoIndiceJogo(t1, t2) + decimal;
21 fim
```

dos times do jogo atual é trocado pelo jogo selecionado (linhas 9, 12, 15 ou 18). Ao final, o valor do gene é atualizado (linha 20) através da função *novoIndiceJogo*, que retorna o índice de um jogo disputado por duas determinadas equipes, o qual é somado à parte decimal que foi armazenada anteriormente.

4.7 Diversificação da População Elite

Para evitar que uma população convirja rapidamente para um ótimo local, foi utilizada uma estratégia de diversificação da população elite. Este processo consiste em verificar se há indivíduos com *fitness* iguais ou com até 5% de diferença e, então, substitui-los por indivíduos mutantes. A população é ordenada novamente para garantir que as melhores soluções permaneçam na elite. A diferença entre os *fitness* é calculada a partir da Equação 4.1.

$$dif = \frac{fintessMaior}{fitnesMenor} \times 100 - 100 \tag{4.1}$$

A comparação dos *fitness* é sempre realizada entre dois indivíduos do conjunto elite e, caso

necessário, o que possui o maior *fitness* entre eles será substituído. Inicialmente, o primeiro e o segundo são comparados, depois o segundo com o terceiro, o terceiro com o quarto, e assim sucessivamente até que o penúltimo indivíduo da elite seja comparado com o último. O Algoritmo 7 apresenta o funcionamento da diversificação da população elite.

Algoritmo 7: Diversificação do conjunto elite

```
   Entrada: cromossomo[m]
 1 para i ← 1 até tamanhoElite faça
 2 |   fitness1 ← cromossomo[i − 1].getFitness();
 3 |   fitness2 ← cromossomo[i].getFitness();
 4 |   diferenca ← (fitness2/fitness1)*100 - 100;
 5 |   se diferenca <= 5 então
 6 |   |   cromossomo[i] ← gerarMutante();
 7 |   |   decodifica(cromossomo);
 8 |   fim
 9 |   populacao.ordena();
10 fim
```

Para cada indivíduo da população elite (laço das linhas 1-10), armazenam-se os *fitness* dos dois que serão comparados (linhas 2 e 3). É calculada a diferença entre os *fitness* (linha 4) de acordo com a Equação 4.1. Então, verifica-se se esta diferença é menor ou igual a 5% (linha 5) e, caso esta condição seja verdadeira, o cromossomo de maior *fitness* é substituído por um indivíduo mutante através da função *gerarMutante* (linha 6). Com isso, o novo indivíduo é decodificado (linha 7) para se obter o seu valor de solução. Ao final, a população atual é ordenada para manter as melhores soluções no conjunto elite (linha 9).

Capítulo 5

Experimentos

Os experimentos computacionais foram realizados em um computador *Intel Core i5 Quad Core* de 3.2 GHz com 16 GB de RAM sob o sistema operacional Ubuntu 12.4.1. O código foi implementado em C++, compilado com g++ 4.4.1 e opção de otimização -O3.

Foi utilizada a *Application Programming Interface* (API) para a linguagem C++, desenvolvida por Toso e Resende (2015) para facilitar o uso do BRKGA, contendo as implementações de todas as fases desta metaheurística, definidas no Capítulo 3. A utilização desta API facilita o uso do BRKGA porque passa a ser necessário implementar apenas a parte específica do problema tratado, alterando os parâmetros que definem o elitismo, *crossover* e mutação, por exemplo.

5.1 Experimentos Preliminares

Nesta seção são apresentados os experimentos que foram realizados antes da execução dos experimentos de comparação de resultados, o que inclui o ajuste de alguns parâmetros do BRKGA, além de um cálculo realizado para medir a eficiência do *crossover* proposto neste trabalho, o qual foi descrito no Capítulo 4.

5.1.1 Ajuste de Parâmetros

Para a execução básica do BRKGA, são considerados cinco parâmetros que precisam ser ajustados a fim de se encontrar a melhor configuração: o índice que determina o tamanho da população (p), a porcentagem da população elite (pe), a porcentagem de mutação (pm), a probabilidade da escolha do pai elite no cruzamento ($rhoe$) e o número máximo de iterações (MAX_GENS).

A incorporação dos métodos de busca local acrescenta mais três parâmetros a serem ajustados, sendo eles o número máximo de execução de cada um dos três métodos utilizados, *Home-away Swap* (MAX_HAS), *Team Swap* (MAX_TS) e *Round Swap* (MAX_RS).

Com a diversificação da população elite, outro parâmetro deve ser levado em consideração, que é a porcentagem mínima de diferença que os indivíduos da elite devem possuir entre si (DIF).

Para a escolha dos melhores valores, foi utilizado o *irace* (López-Ibáñez et al., 2016), uma ferramenta que implementa procedimentos de configuração automática, em particular oferece o procedimento de corrida iterada, que vem sendo usado com sucesso para configurar automaticamente vários algoritmos do estado da arte.

Para a calibragem dos parâmetros listados, foram escolhidas as instâncias que viriam a ser utilizadas nos experimentos, as quais são descritas na próxima seção. Alguns valores foram pré-selecionados conforme apresentados na Tabela 5.1, em que a coluna *Parâmetros* indica o parâmetro analisado e a coluna *Valores* apresenta os diferentes valores possíveis para cada parâmetro.

Tabela 5.1: Ajuste de parâmetros usando *irace*.

Parâmetros	*Valores*
p	{2, 3, 5, 10}
pe	{0,10, 0,15, 0,20, 0,25, 0,30}
pm	{0,05, 0,10, 0,15, 0,20, 0,25, 0,30}
$rhoe$	{0,55, 0,60, 0,65, 0,70, 0,75, 0,80, 0,85, 0,90, 0,95}
MAX_GENS	{300, 400, 500, 600, 700, 800, 900, 1000}
MAX_HAS	{0,30, 0,40, 0,50, 0,60, 0,70}
MAX_TS	{0,30, 0,40, 0,50, 0,60, 0,70}
MAX_RS	{0,30, 0,40, 0,50, 0,60, 0,70}
DIF (%)	{3, 5, 15, 30, 50}

Como resultado dos quatro primeiros parâmetros, foram recomendadas duas configurações bem semelhantes, que se diferem apenas na porcentagem que define o tamanho da população elite (pe). De acordo com o *irace*, elas seriam as melhores configurações para a execução dos experimentos, sendo elas:

- $p = 10$; $pe = 0{,}20$; $pm = 0{,}30$; $rhoe = 0{,}60$; $MAX\_GENS = 900$;
- $p = 10$; $pe = 0{,}30$; $pm = 0{,}30$; $rhoe = 0{,}60$; $MAX\_GENS = 900$.

Entre as configurações recomendadas, foi escolhida a segunda para ser utilizada nos experimentos. Apesar de não apresentarem muita diferença, uma população maior de elite pode proporcionar um tempo de execução menor, pois seriam realizadas menos operações de *crossover*, e talvez até obter melhores resultados, pelo fato de que a população elite armazena os melhores cromossomos. Para os parâmetros que definem o número máximo de vezes que cada busca local é aplicada, foram recomendadas as seguintes configurações:

- $MAX\_HAS = 0{,}40$; $MAX\_TS = 0{,}70$; $MAX\_RS = 0{,}70$;

- $MAX\_HAS = 0{,}40$; $MAX\_TS = 0{,}50$; $MAX\_RS = 0{,}70$;

Entre estas duas configurações, foi escolhida a segunda. Isso porque ela possui um número menor de iterações para a busca local TS, o que resultaria num tempo de execução menor. Os valores destes parâmetros são multiplicados pelo tamanho do cromossomo, indicando uma porcentagem deste número. Por exemplo, se o valor escolhido for 0,50, o número máximo de iterações será $n \times 0,50$, o que indica 50% do tamanho do cromossomo.

O parâmetro que define a diferença mínima que os indivíduos da elite devem possuir entre si (DIF), obteve duas possíveis configurações:

- $DIF = 5$;
- $DIF = 50$.

Entre os valores recomendados para este parâmetro, foi escolhido o primeiro, pois, quanto menor o valor de comparação, maior é a diversidade implantada no conjunto elite. Isto acontece porque uma quantidade maior de cromossomos serão substituídos por indivíduos mutantes, o que diminui a chance de uma rápida convergência para um ótimo local.

5.1.2 Eficiência do *Crossover*

Como descrito no Capítulo 4, o processo de *crossover* foi alterado para se adaptar à nova representação do cromossomo. Considerando, que a chance de escolher o pai da elite é maior do que a de escolher o pai não elite, pode-se observar que, teoricamente, esta maneira de realizar o *crossover* aumenta a probabilidade de os novos filhos tenderem a herdar dos pais elite em excesso, causando a convergência prematura do algoritmo.

Pensando nisto, foi realizado um experimento para analisar quantos filhos recebem genes do pai não elite a cada geração e calcular a média destes valores. O resultado obtido aponta que aproximadamente 69% dos filhos são gerados a partir dos dois pais, o que é um resultado satisfatório, porque garante que haja diversidade nas próximas gerações.

5.2 Comparação dos Resultados

Os valores de referência a serem comparados e as instâncias utilizadas nos experimentos são referentes ao site oficial de Michael Trick [1], sendo considerados neste trabalho apenas os resultados que se referem a campeonatos DRRT espelhados (mTTP). As instâncias possuem tamanhos variados e são separadas em 3 conjuntos, os quais são descritos nas próximas seções. As distâncias contidas nas instâncias e os resultados analisados são medidos em quilômetros (km).

[1] http://mat.gsia.cmu.edu/TOURN/

Dada a utilização de componentes de aleatoriedade pelo método proposto, foram realizadas 10 execuções independentes para cada uma das instâncias. Nas tabelas seguintes, a coluna *Instância* se refere ao nome das instâncias, B^* representa o valor da melhor solução conhecida até o momento na literatura, S^* indica o melhor valor de solução obtido pelo método proposto, S representa a solução média, σ indica o desvio padrão em relação à solução média, T representa o tempo de execução médio em segundos e *gap* apresenta a distância percentual entre os resultados obtidos e as melhores soluções para cada instância, calculado de acordo com a Equação (5.1).

$$gap = 100 \times \frac{S^* - B^*}{B^*} \tag{5.1}$$

5.2.1 Instâncias Circulares

Este conjunto contém nove instâncias circulares, contendo de quatro a vinte times. Estas instâncias não consistem em dados reais de distância entre as cidades dos times, mas se refere à equipes genéricas cuja distância entre os times i e j, com $i > j$, é o mínimo entre $i - j$ e $j - i + n$, sendo n o número de times. Todas as instâncias disponibilizadas neste conjunto foram consideradas nos experimentos realizados e os resultados obtidos são apresentados na Tabela 5.2. Os dados em negrito indicam que o resultado obtido é igual ao resultado de referência.

Tabela 5.2: Comparação dos resultados para as instâncias circulares.

Instância	B^*	S^*	S	T	*gap*	σ
circ4	20	**20**	20,00	0,17	0,00%	0,00
circ6	72	**72**	72,00	1,10	0,00%	0,00
circ8	140	148	154,40	4,16	5,71%	3,63
circ10	272	310	317,00	12,50	13,97%	4,92
circ12	432	532	548,80	30,85	23,15%	9,05
circ14	672	856	871,40	67,74	27,38%	9,80
circ16	968	1.306	1.319,00	134,34	34,92%	7,56
circ18	1.306	1.844	1874,80	257,04	41,19%	13,83
circ20	1.852	2.544	2587,40	452,95	37,37%	20,89

Os resultados ótimos foram encontrados para as duas menores instâncias. Porém, à medida que o tamanho das instâncias fica maior, a qualidade das soluções se deteriora. O *gap* médio obtido é de 20,41%, um valor mediano, em consequência da baixa qualidade das soluções obtidas para as instâncias maiores. O tempo médio obtido chega a 106,76 segundos mesmo se mantendo baixo para as instâncias pequenas e médias. Isso acontece porque este tempo é fortemente influenciado pelas instâncias maiores, que mesmo assim apresentam tempo razoável para o problema tratado. Em relação ao desvio padrão, o método apresenta uma variação baixa

no valor da solução, em que o desvio padrão médio obtido para este conjunto de instâncias é de 7,74.

5.2.2 Instâncias da *National League* (NL)

Este conjunto contém sete instâncias que consistem em dados reais da liga de Beisebol americana (*National League*, NL). O número de times destas instâncias varia entre quatro e dezesseis. A Tabela 5.3 apresenta os resultados obtidos, os quais são comparadas com os melhores conhecidos até o momento para este conjunto. Os dados em negrito indicam que o resultado obtido é igual ao resultado de referência.

Tabela 5.3: Comparação dos resultados para as instâncias NL.

Instância	B^*	S^*	S	T	*gap*	σ
nl4	8.276	**8.276**	8.276,00	0,17	0,00%	0,00
nl6	26.588	**26.588**	26.658,20	1,11	0,00%	62,31
nl8	41.928	43.732	44.470,20	4,15	4,30%	356,90
nl10	63.832	71.488	73.193,20	12,55	11,99%	1.019,33
nl12	119.608	138.966	140.214,40	30,95	16,18%	981,84
nl14	199.363	256.551	260.991,70	67,68	28,69%	2.366,47
nl16	278.305	355.954	367.911,60	134,37	27,90%	5.437,43

Neste conjunto, os resultados apresentaram características semelhantes a do conjunto anterior. Para as duas menores instâncias, o resultado obtido foi equivalente ao valor de referência, enquanto para o restante das instâncias, os resultados perdem qualidade a medida que o número de times fica maior. O *gap* médio obtido neste conjunto é de 12,72%. Pode-se observar que o tempo de execução foi baixo para todas as instâncias, com média de 35,85 segundos, uma vez que este conjunto possui apenas instâncias pequenas e médias. O desvio padrão médio obtido neste conjunto é de 1.460,61, fortemente influenciado pelo tamanho da maior instância, a qual apresentou o maior desvio padrão deste conjunto, 5.437.43. Entretanto, este valor é considerado baixo, sendo ele menor do que 1% da distância total obtida para esta instância.

5.2.3 Instância do Campeonato Brasileiro de 2003

Este conjunto contém apenas uma instância, que consiste em dados reais do campeonato brasileiro de futebol do ano 2003, em que vinte e quatro times participaram. A utilização desta instância é interessante para os testes porque ela possui uma variação muito grande nas distâncias entre os times participantes. Existem estádios de diferentes equipes cuja distância entre eles é de 0 km, como o caso dos times Atlético Mineiro e Cruzeiro que se encontram na mesma cidade. Por outro lado, no caso dos times Grêmio e Fortaleza, a distância é de 3.212 km. A Tabela 5.4 apresenta o resultado obtido para este conjunto.

Tabela 5.4: Comparação dos resultados para a instância do Campeonato Brasileiro de 2003.

Instância	B^*	S^*	S	T	gap	σ
cb2003_24	500.756	731.629	738.638,70	1.243,24	46,10%	4.970,11

Esta é a maior instância executada nos experimentos e, por isso, pode-se observar que este conjunto apresenta o maior tempo de execução, ainda que este tempo não seja considerado impraticável para o problema tratado. Além disso, o *gap* calculado também é o maior encontrado nos testes realizados. Porém, ao realizar a comparação com a solução real do calendário oficial do campeonato no ano de 2003, o método proposto obteve um resultado 30% melhor. Naquela edição do campeonato, a distância total viajada pelas equipes foi de 1.048.134,00 km, enquanto na solução obtida pela implementação do BRKGA proposta, a distância total é 731.629,00 km. Novamente, o desvio padrão obtido (4.970,11) é um valor menor do que 1% da distância total encontrada, apresentando uma pequena variação entre as soluções obtidas para este conjunto.

Capítulo 6

Conclusões

O Problema de Escalonamento de Competições Esportivas (ou mTTP) é um problema NP-Difícil que visa otimizar a tabela de jogos de torneios esportivos, fazendo com que os times viajem o mínimo possível, reduzindo assim os gastos das equipes e o cansaço dos atletas. É possível concluir que o mTTP é um problema importante no contexto esportivo, por se tratar de renomadas competições com grandes recompensas, além de contar com um forte envolvimento de dinheiro em atletas, *marketing* e em equipes em geral, o que transforma os esportes em grandes formas de negócios.

Foram introduzidas neste trabalho a definição formal, uma revisão detalhada da literatura e uma aplicação do BRKGA ao mTTP. Apesar do fácil entendimento desta metaheurística, algumas de suas fases sofreram alterações significativas, aumentando a dificuldade na implementação e afetando um pouco na eficiência do método.

Neste trabalho também foram apresentados três métodos de busca local aplicados ao mTTP, sendo eles o *Home-away Swap*, *Team Swap* e o *Round Swap*. O intuito da utilização destes métodos é melhorar os resultados obtidos inicialmente, objetivo o qual foi atingido apesar destas melhorias não levarem as soluções a valores muito próximos ou iguais aos melhores conhecidos.

Os experimentos computacionais envolveram um total de 17 instâncias divididas em três conjuntos, sendo eles o de instâncias circulares, instâncias reais da liga de Beisebol americana (NL) e uma instância real do campeonato brasileiro de de futebol referente à temporada 2003. Os resultados gerados foram comparados com os melhores conhecidos na literatura. Os valores obtidos nos experimentos se mostraram bons para as instâncias pequenas, os quais se igualaram aos valores de referência. Entretanto, a qualidade das soluções se degradou à medida em que o tamanho das instâncias aumentou, característica que se repetiu em todos os conjuntos. O tempo de execução se manteve baixo para os conjuntos de instâncias circulares e NL, se elevando um pouco para a instância do campeonato brasileiro. A utilização dos métodos de busca local melhorou os resultados obtidos inicialmente. Por outro lado, houve um aumento do tempo de execução do método proposto, embora estes valores ainda sejam

considerados praticáveis no contexto do problema tratado.

Os trabalhos futuros incluem o aprimoramento do método proposto através da incorporação de diferentes métodos de busca local, além dos que foram utilizados neste trabalho. Serão realizados novos experimentos computacionais e análises adicionais à nova versão do método.

Referências Bibliográficas

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160.

Biajoli, F. L. e Lorena, L. A. N. (2006). Mirrored traveling tournament problem: an evolutionary approach. In *Advances in Artificial Intelligence-IBERAMIA-SBIA 2006*, pp. 208–217. Springer.

Briskorn, D. (2008). *Sports leagues scheduling: models, combinatorial properties, and optimization algorithms*, volume 603. Springer Science & Business Media.

Briskorn, D. e Drexl, A. (2009). A branching scheme for finding cost-minimal round robin tournaments. *European Journal of Operational Research*, 197(1):68–76.

Carvalho, M. A. M. e Lorena, L. A. N. (2012). New models for the mirrored traveling tournament problem. *Computers & Industrial Engineering*, 63(4):1089–1095.

Cheung, K. K. (2008). Solving mirrored traveling tournament problem benchmark instances with eight teams. *Discrete Optimization*, 5(1):138–143.

Cheung, K. K. (2009). A benders approach for computing lower bounds for the mirrored traveling tournament problem. *Discrete Optimization*, 6(2):189–196.

Della Croce, F. e Oliveri, D. (2006). Scheduling the italian football league: An ilp-based approach. *Computers & Operations Research*, 33(7):1963–1974.

Gonçalves, J. F. e Resende, M. G. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525.

Goossens, D. e Spieksma, F. (2009). Scheduling the belgian soccer league. *Interfaces*, 39(2):109–118.

Goossens, D. R. e Spieksma, F. C. (2012). Soccer schedules in europe: an overview. *Journal of scheduling*, 15(5):641–651.

Gupta, D.; Goel, L. e Aggarwal, V. (2013). A hybrid biogeography based heuristic for the mirrored traveling tournament problem. In *Contemporary Computing (IC3), 2013 Sixth International Conference on*, pp. 325–330. IEEE.

Kendall, G.; Knust, S.; Ribeiro, C. C. e Urrutia, S. (2010). Scheduling in sports: An annotated bibliography. *Computers & Operations Research*, 37(1):1–19.

Knust, S. e Lücking, D. (2009). Minimizing costs in round robin tournaments with place constraints. *Computers & Operations Research*, 36(11):2937–2943.

López-Ibáñez, M.; Dubois-Lacoste, J.; Cáceres, L. P.; Birattari, M. e Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Nemhauser, G. L. e Trick, M. A. (1998). Scheduling a major college basketball conference. *Operations research*, 46(1):1–8.

Prasetyo, H.; Fauza, G.; Amer, Y. e Lee, S. (2015). Survey on applications of biased-random key genetic algorithms for solving optimization problems. In *Industrial Engineering and Engineering Management (IEEM), 2015 IEEE International Conference on*, pp. 863–870. IEEE.

Rasmussen, R. V. e Trick, M. A. (2007). A benders approach for the constrained minimum break problem. *European Journal of Operational Research*, 177(1):198–213.

Rasmussen, R. V. e Trick, M. A. (2008). Round robin scheduling–a survey. *European Journal of Operational Research*, 188(3):617–636.

Ribeiro, C. C. (2012). Sports scheduling: Problems and applications. *International Transactions in Operational Research*, 19(1-2):201–226.

Ribeiro, C. C. e Urrutia, S. (2006). Scheduling the brazilian soccer tournament with fairness and broadcast objectives. In *International Conference on the Practice and Theory of Automated Timetabling*, pp. 147–157. Springer.

Ribeiro, C. C. e Urrutia, S. (2007). Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, 179(3):775–787.

Ribeiro, C. C. e Urrutia, S. (2012). Scheduling the brazilian soccer tournament: Solution approach and practice. *Interfaces*, 42(3):260–272.

Toso, R. F. e Resende, M. G. (2015). A c++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93.

Urrutia, S. e Ribeiro, C. C. (2004). Minimizing travels by maximizing breaks in round robin tournament schedules. *Electronic Notes in Discrete Mathematics*, 18:227–233.

Van Hentenryck, P. e Vergados, Y. (2006). Traveling tournament scheduling: a systematic evaluation of simulated annealling. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pp. 228–243. Springer.