Universidade Federal Fluminense

MARCUS VINÍCIUS DE SOUZA PAVAN

Heurística para Programação de Tabelas de Competições Esportivas e sua Aplicação ao Campeonato Nacional de Basquete

NITERÓI
2006

MARCUS VINÍCIUS DE SOUZA PAVAN

Heurística para Programação de Tabelas de Competições Esportivas e sua Aplicação ao Campeonato Nacional de Basquete

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de mestre. Área de concentração: Otimização Combinatória.

Orientador:
Celso da Cruz Carneiro Ribeiro

Co-orientador:
Sebastián Alberto Urrutia

UNIVERSIDADE FEDERAL FLUMINENSE

NITERÓI
2006

Heurística para Programação de Tabelas de Competições Esportivas e sua Aplicação ao Campeonato Nacional de Basquete

Marcus Vinícius de Souza Pavan

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de mestre.

Aprovada por:

_____
Prof. Celso da Cruz Carneiro Ribeiro / IC–UFF (Presidente)

_____
Prof. Sebastián Alberto Urrutia / DCC–UFMG

_____
Prof. Luiz Satoru Occhi / IC–UFF

_____
Prof. Alexandre Plastino de Carvalho / IC–UFF

_____
Profa. Simone de Lima Martins / IC–UFF

Niterói, 27 de Outubro de 2006.

Aos meus pais, com todo amor e carinho.

Agradecimentos

Em primeiro lugar agradeço a DEUS, por me dar força e sabedoria durante o desenvolvimento deste trabalho e tranqüilidade nos momentos mais difíceis.

Agradeço a toda minha família, pelas palavras de conforto e por sempre torcer e acreditar em mim. Em especial a minha mãe Maria de Souza Pavan, pelas orações e apoio incondicional e ao meu pai Antônio Fioravante Pavan, pelo esforço e exemplo de vida.

A mulher mais linda que já conheci, Karime Mohamad Sakhr, pelo apoio, paciência, amor e carinho nos momentos mais difíceis. Muito obrigado minha linda, amo você.

Ao orientador e amigo Sebastián Alberto Urrutia, pelas orientações excepcionais no desenvolvimento desse trabalho, pela idealização do mesmo, pela confiança, pelo companheirismo e por toda dedicação e cuidado que teve ao longo deste trabalho.

Ao meu co-orientador Celso da Cruz Carneiro Ribeiro, o muitíssimo obrigado pelas ótimas idéias, pela orientação e atenção dedicadas a este trabalho, pelas revisões de texto, paciência e confiança.

Ao professor Luiz Satoru Ochi, pelos ensinamentos e apoio durante todo curso de mestrado, pelo suporte a esse trabalho e por acreditar em mim.

A todos os demais professores, colegas e funcionários que contribuíram de alguma maneira para a realização deste trabalho.

Resumo

A programação de tabelas de competições esportivas é a principal área de aplicação dos métodos de Pesquisa Operacional em esportes. Muitas ligas e federações de diferentes esportes lidam com esse problema diversas vezes, ao criarem as tabelas das competições que organizam. Essas tabelas precisam satisfazer diferentes tipos de restrições e, muitas vezes, otimizar objetivos tais como a distância percorrida pelas equipes durante a competição. A programação de tabelas para competições esportivas é uma tarefa difícil, na qual diversas técnicas de otimização combinatória têm sido aplicadas.

Esta dissertação tem como objetivo desenvolver uma estratégia heurística para a programação de tabelas de competições esportivas aplicada ao Campeonato Nacional Masculino de Basquete de 2004. Experimentos computacionais foram realizados a fim de comparar os resultados obtidos pelo algoritmo proposto com os resultados existentes para o Campeonato Nacional Masculino de Basquete de 2004, permitindo verificar a eficácia das estratégias propostas.

Abstract

The scheduling of sport leagues has become an important class of combinatorial optimization problems in recent years. Numerous professional sport leagues deal with this problem in every season. There are important issues that have to be considered in the scheduling, besides traditional requirements like the distance that teams need to travel. Different techniques have been used to solve this kind of problem.

In this work, we propose a heuristic for sports scheduling problems and apply it to the 2004 edition of the men's Brazilian Basketball Tournament. Computational experiments are performed and the results obtained by the heuristic are compared with those published by *Confederação Brasileira de Basquete* to assess the effectiveness of the proposed solution strategy.

Palavras-chave

1. Programação de tabelas esportivas
2. Minimização de distâncias
3. Otimização combinatória
4. Metaheurísticas

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O problema de programação de tabelas de competições esportivas pode ser definido como determinar a seqüência de jogos de cada time dentro da competição envolvida. Este tipo de problema vem despertando o interesse de muitos pesquisadores (ver por exemplo, [1-4, 6-8, 10, 15-17, 20-26]) devido à sua elevada complexidade computacional, sua formulação simples e sua aplicabilidade em situações reais. Muitas ligas e federações de diferentes esportes lidam com esse problema diversas vezes por ano, ao montarem as tabelas dos jogos dos times associados em cada campeonato.

Diversas questões devem ser consideradas na programação dessas tabelas, tais como: minimização de quebras (número de vezes em que um time tem jogos consecutivos dentro ou fora de casa), indisponibilidade de estádios (datas em que determinados estádios não estão disponíveis para realização de jogos) e minimização de distâncias (soma das distâncias que os times percorrem durante o campeonato), entre outras.

A dificuldade dos problemas de programação de tabelas cresce quando as sedes das diferentes equipes estão geograficamente distribuídas em uma grande região. Em muitos casos, a distância percorrida pelas equipes é uma importante variável a ser minimizada a fim de diminuir as despesas com viagens e dar mais tempo de descanso aos jogadores.

Várias abordagens [7] têm sido propostas por diferentes autores para resolver problemas de programação de tabelas de competições esportivas, usando diferentes técnicas como programação inteira, algoritmos de grafos, programação por restrições e metaheurísticas. Estas últimas têm sido aplicadas para resolver problemas complexos que envolvem muitos times, muitas restrições e muitos jogos. Entre elas, destacam-se busca tabu [9], algoritmos genéticos [18], *simulated annealing* [1] e GRASP [19], existindo ainda algoritmos híbridos que combinam características de duas ou mais metaheurísticas [4]. Tais problemas abrangem os mais variados esportes, como basquete, futebol, hóquei, beisebol e futebol americano [2-4, 12, 20, 21, 25], entre outros.

No Brasil, o basquete, assim como a maioria dos esportes olímpicos, não conta com grandes investimentos financeiros, mas é um dos esportes mais praticados nas escolas do país, assim como o atletismo, futsal, handebol e voleibol.

A Confederação Brasileira de Basquete (CBB) foi fundada na cidade do Rio de Janeiro, em 25 de dezembro de 1933. Ela é a única entidade nacional filiada à Federação Internacional de Basketball (FIBA) e ao Comitê Olímpico Brasileiro (COB), sendo responsável por promover, organizar e garantir o funcionamento do Campeonato Nacional de Basquete, incluindo a elaboração da tabela.

No ano de 2004, o Campeonato Nacional Masculino de Basquete contou com a participação de 16 times na sua fase de classificação, que foi disputada em dois turnos (turno e returno) de 25 janeiro de 2004 a 5 de maio de 2004 (totalizando 240 jogos), tendo sua tabela sido elaborada pelos organizadores da Confederação Brasileira de Basquete. Cada jogo foi disputado na sede de uma das equipes envolvidas no confronto. O time que pertence à sede joga em casa e é chamado de mandante, enquanto que o outro time joga fora de casa e é chamado de visitante. Durante a fase de classificação, cada time jogou contra todos os demais duas vezes, sendo uma como mandante e outra como visitante.

Os custos de viagens são pagos pela CBB, enquanto que as despesas de hospedagem das delegações ficam a cargo dos clubes. Devido a fatores econômicos dos clubes, da CBB, dos patrocinadores e das emissoras de televisão, diversas restrições devem ser consideradas na elaboração da tabela da fase de classificação do campeonato, tais como garantir um número mínimo e máximo de jogos por semana para cada time, garantir um número mínimo de jogos toda sexta-feira e todo domingo (restrição imposta pelas emissoras de televisão) e garantir um número máximo de jogos consecutivos dentro e fora de casa para cada time, entre outras. Além disso, a tabela deve minimizar as distâncias percorridas pelos clubes nas viagens necessárias para a realização dos jogos, a fim de permitir maior tempo de descanso aos jogadores e reduzir os custos com viagens pagas pela CBB.

Em 2005, ocorreu uma crise do esporte no País, em que parte dos clubes filiados à CBB, juntamente com alguns dos maiores ex-atletas do esporte no País, como Oscar Schmidt, Paula Gonçalves e Hortência Marcari se uniram para formar a Nossa Liga de Basquete (NLB) devido à insatisfação com certos aspectos da administração e organização do Campeonato Nacional de Basquete de 2004, tais como o repasse de verbas de patrocinadores e o corte da ajuda de custo aos clubes, como hospedagens e transportes para viagens. A CBB, presidida por Gerasime Nicolas Bozikis, não concordou com a formação da Liga, proibindo as federações e os clubes filiados de participarem da NLB.

O presente trabalho tem como objetivo apresentar um algoritmo para solucionar de forma eficiente o problema da programação da tabela da fase classificatória do Campeonato Nacional de Basquete, gerando soluções que atendam aos requisitos definidos pela Confederação Brasileira de Basquete e minimizem as distâncias percorridas pelos clubes na tabela. O algoritmo é baseado em heurísticas a fim de se obter soluções de boa qualidade em um tempo computacional satisfatório. Os resultados obtidos serão comparados com a tabela utilizada pela CBB no Campeonato Nacional Masculino de 2004.

A presente dissertação está organizada da seguinte forma. No próximo capítulo, apresenta-se uma revisão da literatura sobre problemas de programação de tabelas de competições esportivas e uma descrição detalhada do problema do Campeonato Nacional Masculino de Basquete de 2004. No Capítulo 3, apresenta-se a heurística empregada na solução do problema do Campeonato Nacional Masculino de Basquete de 2004, a heurística de construção, as buscas locais, procedimentos de perturbação e outras características do algoritmo proposto para solucionar o problema. Os resultados computacionais são apresentados no Capítulo 4. Finalmente, no Capítulo 5 as considerações finais sobre o trabalho desenvolvido são apresentadas.

Capítulo 2

O problema de programação de tabelas de competições esportivas

2.1 Introdução

O problema de programação de tabelas de competições esportivas consiste em determinar quando e onde as partidas de um campeonato devem ser disputadas, de forma que um conjunto de restrições seja satisfeito e uma determinada função objetivo seja otimizada.

Em uma competição composta por n (número inteiro positivo par) times e m jogos, os jogos são distribuídos em blocos de tempo. Tais blocos podem representar um dia, um intervalo de horários num mesmo dia, uma semana, ou outro intervalo de tempo, dependendo da particularidade do problema considerado. Em geral, os blocos representam um dia, que por sua vez, estão associados às r rodadas em que os jogos da competição são divididos.

Um problema é considerado compacto quando possui o número mínimo de blocos para que os jogos sejam distribuídos. Caso haja mais blocos do que o mínimo necessário para a distribuição dos jogos, o problema é considerado não compacto. Em geral, tem-se uma restrição que proíbe que cada time jogue mais de uma vez por rodada. Sendo assim, cada rodada tem no máximo $n/2$ jogos.

Em competições *inter-sedes*, cada equipe participante possui uma sede onde são realizados os seus jogos em casa. Logo, a escolha do local de cada jogo se limita a duas possibilidades, ou seja, na sede de uma das equipes que se enfrentam.

Em um torneio *round robin* simples (SRR, do inglês *single round robin*), todas as equipes se enfrentam uma única vez ao longo da competição. Os jogos são realizados na sede de uma das equipes envolvidas no confronto. Já em torneios *round robin* duplo (DRR, do inglês *double round robin*) todas as equipes se enfrentam duas vezes, sendo geralmente uma vez em cada sede. Tais torneios são chamados de espelhados quando os jogos contidos na

primeira metade da tabela são repetidos na mesma ordem na segunda metade da tabela, invertendo-se apenas as equipes mandante e visitante.

Em um torneio SRR compacto, os $n(n\text{-}1)/2$ jogos do torneio são divididos em n-1 rodadas com $n/2$ jogos cada uma. Já em um torneio DRR compacto, os $n(n\text{-}1)$ jogos são divididos em $2(n\text{-}1)$ rodadas com $n/2$ jogos cada uma.

Diversas competições de vários esportes diferentes como futebol, basquete, beisebol e hóquei são organizados como torneios SRR ou DRR, tendo sido aplicadas diversas técnicas para a programação de tabelas dessas competições. Costa [4] considerou o problema de programação dos jogos da National Hockey League (NHL) dos Estados Unidos, cujo objetivo é minimizar a distância percorrida pelas equipes durante o torneio. Para tanto, foi utilizada uma abordagem que combina busca tabu com algoritmos genéticos, que Costa chamou de busca tabu evolucionária. Nemhauser e Trick [17] trataram do problema da alocação de jogos de um torneio de basquete envolvendo nove times de diferentes localidades dos Estados Unidos utilizando uma combinação de programação inteira e uma técnica enumerativa. O procedimento proposto contém três fases. A primeira fase consiste em gerar um conjunto de padrões de cardinalidade igual ao número n de times. Cada padrão é uma seqüência de n-1 caracteres, sendo que cada caractere pode assumir um dos valores: casa (C) ou fora (F). Na segunda fase, os jogos são designados aos padrões, sem se estabelecer quais são os times envolvidos em cada jogo. Finalmente, na última fase, os times são designados aos padrões, produzindo uma tabela. Yang e Huang [25] trataram do problema da criação da tabela da *Major League Baseball* (MLB) dos Estados Unidos de 1998, que possui 30 times, tendo como objetivo minimizar o custo das viagens dos times e atender a um conjunto de restrições. Tal problema é considerado bastante complexo devido ao número de times envolvidos, ao número de restrições e ao número de jogos de cada time (162). Eles comparam os resultados obtidos pela sua abordagem evolucionária, que combina algoritmos genéticos com busca local usando *simulated annealing* e uma vizinhança 2-optimal, com os resultados alcançados pelos organizadores da MLB que criaram a tabela de forma manual.

2.2 Tipos de problema

Problemas envolvendo torneios SRR e DRR são freqüentemente encontrados na literatura de programação de tabelas esportivas. Entretanto, existem variações desses problemas, dentre os quais se destacam o problema do torneio com viagens [6], o problema de

minimização de quebras [16] e o problema de modelos de torneios balanceados [7, 10], entre outros.

2.2.1 O problema do torneio com viagens

O problema do torneio com viagens, formulado por Easton et al. [6], é um problema de programação de tabelas esportivas no qual as viagens que os times devem fazer para realizar seus jogos é uma questão importante.

Ele caracteriza-se por ser um problema referente a um torneio DRR *inter-sedes* compacto. Quando um time tem que realizar um jogo fora de casa, assume-se que há uma viagem a ser feita até a cidade sede de seu oponente. Quando existem jogos fora de casa consecutivos, os times viajam da cidade de um oponente até a cidade do próximo adversário diretamente. Assume-se que cada time inicia o torneio na sua sede e deve retornar a ela no final do torneio.

O objetivo desse problema consiste em encontrar a melhor seqüência de jogos a fim de minimizar a distância total percorrida pelos times sujeito a que nenhum time jogue mais de três jogos consecutivos em casa, nem jogue mais de três jogos consecutivos fora de casa.

Além disso, ainda pode haver mais restrições envolvidas, tais como o espelhamento (jogos contidos na primeira metade da tabela são repetidos na mesma ordem na segunda metade da tabela, invertendo-se apenas os times mandante e visitante) ou inexistência de repetições (o mesmo jogo entre os times i e j não pode ocorrer em duas rodadas consecutivas). Nota-se que em torneios espelhados não há repetições.

O problema do torneio com viagens envolve questões de viabilidade (nenhum time pode jogar mais de três jogos consecutivos em casa nem mais de três jogos consecutivos fora de casa) e questões de otimização (minimização da soma das distâncias que os times devem percorrer). A combinação desses aspectos vem despertando o interesse de muitos pesquisadores, que, por sua vez, têm utilizado diferentes técnicas como programação inteira, programação por restrições, algoritmos genéticos, busca tabu, *simulated annealing*, GRASP e ILS (ver, por exemplo, [1, 23]).

Easton et al. [6] aplicaram diversas técnicas para resolver o problema utilizando diversas instâncias, com diferentes números de times. Para instâncias pequenas (com quatro times) é possível solucionar o problema em poucos segundos usando técnicas exatas, como programação inteira e programação por restrições. Para instâncias com seis times, as soluções

ótimas são encontradas em torno de quatro horas de processamento. As instâncias com oito ou mais times não foram resolvidas até hoje, exceto através de processamento paralelo em casos especiais.

2.2.2 O problema de minimização de quebras

Em programação de tabelas esportivas, considera-se indesejável que um time tenha que realizar dois jogos consecutivos dentro ou fora de casa [16]. Se em uma determinada tabela, um time tem que realizar dois jogos consecutivos dentro de casa nas rodadas *d*-1 e *d*, diz-se que há uma quebra em casa na rodada *d*. Caso um time tenha dois jogos consecutivos fora de casa nas rodadas *d*-1 e *d*, diz-se que há uma quebra fora de casa na rodada *d*.

Dada uma tabela de um torneio SRR sem as atribuições de mando de campo, o problema de minimização de quebras consiste em determinar a equipe mandante dos jogos de maneira que as quebras sejam minimizadas. Para um torneio com uma quantidade par de times, essa escolha do time mandante de cada jogo deve ser tal que o número de jogos em casa para cada time seja igual ao número de jogos fora de casa menos um ou igual ao número de jogos fora mais um. Sabe-se que, para qualquer torneio SRR, o número de quebras é maior ou igual a n - 2, onde n é o número de times. Sabe-se também, que para qualquer n inteiro positivo, existe pelo menos um par de tabelas do torneio e uma escolha de mandantes no qual o número de quebras é igual a n - 2 [5]. A Tabela 1 mostra um torneio hipotético com oito times. A Tabela 2 mostra uma escolha de mandantes dos jogos do torneio representado pela Tabela 1 com o mínimo de quebras possível. A letra “C” representa que o jogo é em casa, enquanto que e letra “F” representa que o jogo é fora de casa.

	Rodadas						
Times	1	2	3	4	5	6	7
1	8	5	2	3	4	6	7
2	6	7	1	8	3	5	4
3	7	6	8	1	2	4	5
4	5	8	7	6	1	3	2
5	4	1	6	7	8	2	3
6	2	3	5	4	7	1	8
7	3	2	4	5	6	8	1
8	1	4	3	2	5	7	6

Tabela 1: Tabela de um torneio hipotético com oito times.

	Rodadas						
Times	1	2	3	4	5	6	7
1	F	C	F	C	F	C	F
2	C	F	C	F	F	C	F
3	C	F	C	F	C	C	F
4	F	F	C	F	C	F	C
5	C	F	C	F	C	F	C
6	F	C	F	C	C	F	C
7	F	C	F	C	F	F	C
8	C	C	F	C	F	C	F

Tabela 2: Escolha de mandantes dos jogos do torneio representado pela Tabela 1.

Para qualquer atribuição de estádios, nota-se que, em qualquer rodada, metade dos times joga em casa e a outra metade joga fora de casa. Logo, o número de quebras em casa em uma rodada *d* é igual ao número de quebras fora de casa na mesma rodada. Isto implica que o problema de minimização de quebras equivale à minimização de quebras dentro de casa (ou, de modo equivalente, minimizar as quebras fora de casa).

2.2.3 O problema de modelos de torneios balanceados

Embora os torneios *inter-sedes* sejam os mais tratados na literatura [7], existem outros torneios que têm recebido atenção. Conforme dito anteriormente, em torneios *inter-sedes* os jogos são disputados na sede de um dos times envolvidos no confronto. Entretanto, no problema de modelos de torneios balanceados existem instalações comuns (ou apenas uma) onde todos os jogos são realizados. Esse problema consiste na programação de tabelas de torneios *round robin* (simples ou duplos), mas requer que os jogos de um determinado time sejam distribuídos eqüitativamente nos recursos disponíveis (instalações ou intervalos de tempo). Um torneio balanceado pode ser visto como um torneio onde os jogos são disputados em um único estádio em diferentes intervalos de tempo. Assim, os jogos de cada time devem ser distribuídos de maneira equalitária (ou o mais próximo possível) sobre os intervalos disponíveis. Um exemplo desse tipo de problema pode ser visto em [10].

2.3 Formalização do problema de programação de tabelas de competições esportivas

Considera-se uma competição esportiva da qual participam n times. Esses times jogam um determinado número m de jogos. Dado um conjunto de times $T = \{1,2,...,n\}$, denota-se um jogo entre o time $i \in T$ e o time $j \in T$ pelo par (i, j), com $i \neq j$, sendo o primeiro o time mandante e o segundo, o time visitante. Na sua versão mais simples, o problema de programação de tabelas consiste em dividir os m jogos em r rodadas, sujeito a que nenhuma equipe jogue mais de uma vez em cada rodada. Cada rodada tem no máximo $n/2$ jogos.

2.3.1 Fatoração de grafos

Um *grafo* $G = (V,E)$ é um objeto matemático composto por um conjunto de *nós* V e um conjunto de *arestas* E. Cada aresta conecta um par de nós. Dois nós unidos por uma aresta são adjacentes e são incidentes à aresta que os une. Um subgrafo de G é um grafo $G' = (V',E')$ tal que $V' \subseteq V$ e $E' \subseteq E$. O grau $d_G(v)$ de um nó $v \in V$ é igual à quantidade de arestas de E que tem v como extremidade. Se as arestas do grafo têm uma orientação (chama-se de arcos às arestas orientadas), o grafo é orientado. Neste caso, o par (u, v) representa o arco que vai do nó u ao nó v. Denota-se por $\vec{G} = (V,A)$, um grafo orientado. Um grafo não orientado é completo se entre todo par de nós existe uma aresta. Um grafo orientado é completo se entre todo par de nós existem dois arcos, um do primeiro para o segundo e outro do segundo para o primeiro. A Figura 1 mostra um exemplo de um grafo não orientado completo com seis nós e a Figura 2, um grafo orientado completo também com seis nós.

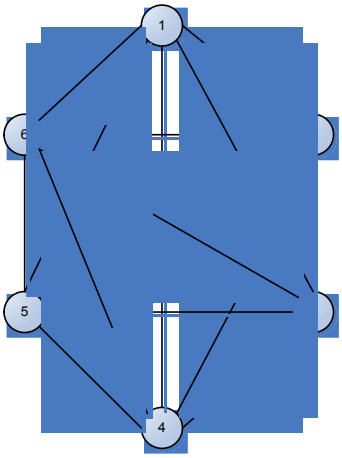

Figura 1: Grafo não orientado completo com seis nós.

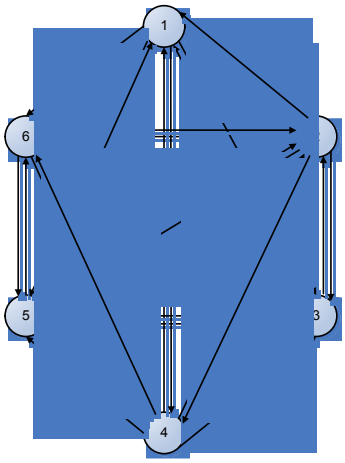

Figura 2: Grafo orientado completo com seis nós.

Dado um grafo $G = (V,E)$, um *fator* de G é um subgrafo $G' = (V,E')$ com $E' \subseteq E$. Uma *fatoração* de G é um conjunto de fatores $F = \{f_1, \ldots, f_p\}$, onde $f_i = (V, E_i)$ para $i = 1 \ldots p$, $E_i \cap E_j = \emptyset$, $1 \leq i \leq j \leq p$ e $\bigcup_{i=1}^{p} E_i = E$. Um *fator* f é um $[a,b]$-*fator* se $a \leq d_f(v) \leq b$, $\forall\, v \in V$. Em uma $[a,b]$-*fatoração* de G, todos os fatores são $[a,b]$-fatores. Uma $[a,a]$-fatoração é também chamada de uma *a-fatoração*. Quando uma certa ordem é atribuída aos fatores de uma fatoração, ela é chamada de ordenada. Em fatorações de grafos não orientados, pode-se atribuir uma orientação a cada aresta de cada fator. Neste caso, diz-se que a fatoração é *orientada*. Não deve-se confundir fatorações orientadas de grafos não orientados com fatorações de grafos orientados. Neste último caso, os arcos que são distribuídos em fatores já tem uma orientação única definida.

Um torneio pode ser associado a um grafo, onde cada nó do grafo representa um time e cada aresta representa um jogo a ser disputado entre os times representados pelos nós que lhe são incidentes. Em um grafo orientado, os arcos indicam também a sede na qual o jogo será realizado, onde o nó de origem representa o time visitante e o nó destino representa o time mandante. Uma tabela de um torneio pode ser vista como uma [0,1]-fatoração deste grafo com r [0,1]-fatores, sendo r o número de rodadas do torneio e cada [0,1]-fator uma possível rodada do torneio. Nesse caso, como o grau de cada nó em um [0,1]-fator é no máximo um, não é possível que um time jogue mais de um jogo por rodada. A Tabela 3 representa uma possível tabela para os jogos de um campeonato hipotético com cinco rodadas composto por seis times representados por números de 1 a 6. A Figura 3 representa a 1-fatoração do grafo correspondente a essa tabela com cinco 1-fatores.

Rodada 1	Rodada 2	Rodada 3	Rodada 4	Rodada 5
3 vs 5	4 vs 1	5 vs 2	2 vs 6	3 vs 6
4 vs 6	2 vs 3	1 vs 6	4 vs 5	1 vs 5
1 vs 2	5 vs 6	3 vs 4	1 vs 3	2 vs 4

Tabela 3: Tabela hipotética com seis times e cinco rodadas.

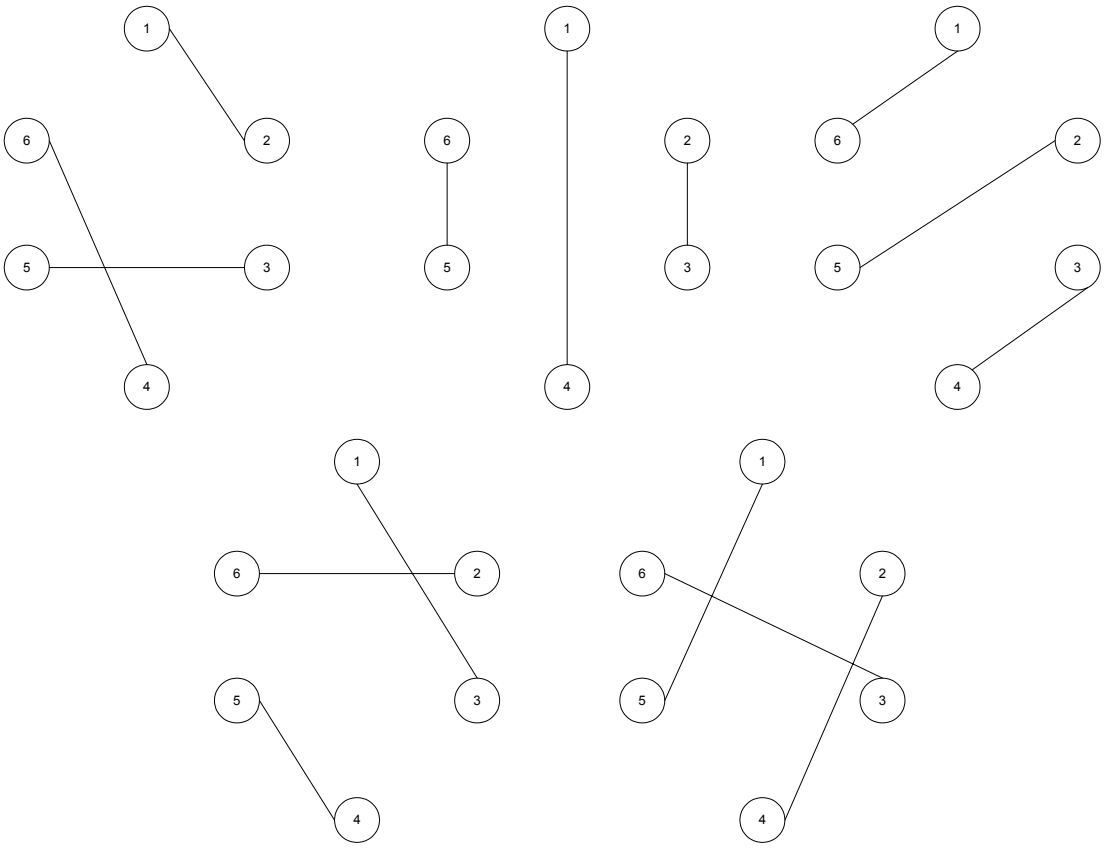

Figura 3: 1-fatoração do grafo correspondente à Tabela 3.

Diversas vantagens podem ser observadas na utilização da fatoração de grafos em problemas de programação de tabelas de competições esportivas, tais como a determinação das propriedades do problema, formulação simples, visualização das restrições envolvidas e definição das estruturas de vizinhança, entre outras.

Como em um torneio SRR cada time joga uma vez contra todos os outros, o grafo associado a esse torneio é completo. Se o torneio SRR é compacto e n é par, então há $r = n$-1 rodadas, todas as equipes jogam em todas as rodadas e toda tabela corresponde a uma 1-fatoração ordenada do grafo completo. Se o torneio é *inter-sedes*, a tabela pode ser associada a uma 1-fatoração ordenada e orientada, de tal forma que o nó que é apontado por um arco represente a equipe que joga em casa no jogo correspondente.

O grafo associado a um torneio DRR *inter-sedes* é orientado e completo. Se o torneio DRR é compacto e n é par, cada 1-fatoração ordenada deste grafo representa uma possível tabela do torneio. Caso o torneio DRR não seja compacto e contenha r rodadas, cada possível tabela desse torneio é representada por uma [0,1]-fatoração com r [0,1]-fatores. A Tabela 4 representa um torneio DRR inter-sedes com n = 4 times e r = 8 rodadas, e a Figura 4 a [0,1]-fatoração do grafo correspondente à Tabela 4.

Rodada 1	Rodada 2	Rodada 3	Rodada 4	Rodada 5	Rodada 6	Rodada 7	Rodada 8
1 vs 3	2 vs 3	4 vs 1	3 vs 4	1 vs 4	4 vs 2	2 vs 1	4 vs 3
2 vs 4			1 vs 2	3 vs 2	3 vs 1		

Tabela 4: Torneio DRR com quatro times e oito rodadas.

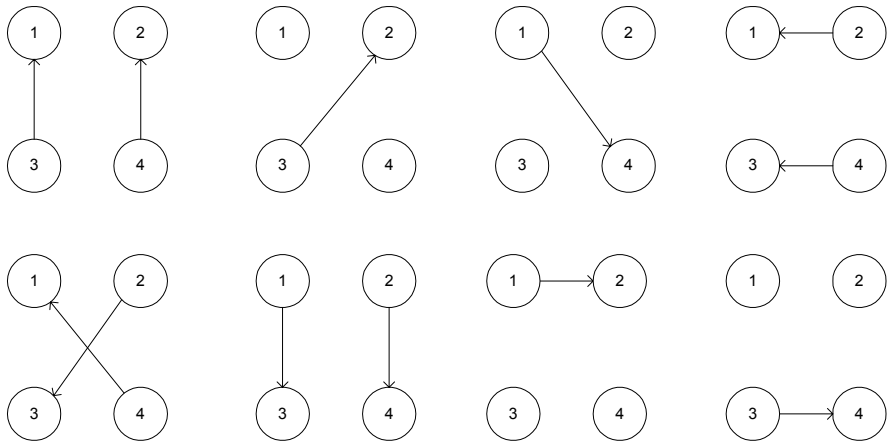

Figura 4: [0,1]-fatoração do gráfico correspondente à Tabela 4.

Nota-se a diferença entre [0,1]-fatorações ordenadas e orientadas de grafos completos não orientados (que representam torneios SRR) e [0,1]-fatorações ordenadas de grafos completos orientados (que representam torneios DRR). No primeiro caso, trata-se de [0,1]-fatorações com r [0,1]-fatores, onde cada um desses [0,1]-fatores contém de zero a n/2 arcos, cada um deles associados a uma das n.(n-1)/2 arestas do grafo completo à qual é atribuída uma orientação. No segundo caso, trata-se de [0,1]-fatorações com r [0,1]-fatores, onde cada um desses [0,1]-fatores contém de zero a n/2 arcos, correspondendo cada um deles a um dos n.(n-1) arcos do grafo orientado completo.

2.4 O problema do Campeonato Nacional Masculino de Basquete de 2004

Em 2004, o Campeonato Nacional Masculino de Basquete contou, em sua fase de classificação, com a participação de 16 clubes filiados à CBB. A tabela dos jogos do campeonato foi criada pela CBB.

A fase de classificação teve seu início em 25 de janeiro e terminou em 2 de maio. Foi interrompida durante o Carnaval, de 17 a 27 de fevereiro, totalizando 90 dias disponíveis para jogos. Durante esta fase, todos os times jogaram entre si duas vezes em turno e returno, totalizando 240 jogos. A relação entre o número de jogos e os dias disponíveis indica que o campeonato é um DRR não compacto, com uma média de 2,6 jogos por dia. Os times que participaram do campeonato são de oito unidades da federação e de 15 cidades diferentes. A Tabela 5 relaciona cada time à sua unidade federativa e à sua cidade de origem. Em cada uma dessas cidades há um estádio onde o time da cidade sedia seus jogos em casa.

Unidade Federativa	Cidade	Time
Distrito Federal	Brasília	Universo/BRB/DF
Goiás	Goiânia	Universo/Ájax
Minas Gerais	Belo Horizonte	Universo/BH/Minas
Minas Gerais	Uberlândia	Unit/Uberlândia
Paraná	Cornélio Procópio	Londrina/TIM
Rio de Janeiro	Campos	ACF Campos/Universo
Rio de Janeiro	Rio de Janeiro	Flamengo/Petrobras
Rio de Janeiro	Rio de Janeiro	Tijuca Tênis Clube
Rio Grande do Sul	Torres	Sport Club Ulbra
Santa Catarina	Blumenau	URB/Adeblu
São Paulo	Araraquara	Uniara/Araraquara
São Paulo	Casa Branca	Liberty Seguros/Casa Branca
São Paulo	Franca	Franca Basquete
São Paulo	Mogi das Cruzes	Corinthians/UMC
São Paulo	Ribeirão Preto	COC/Ribeirão Preto
São Paulo	São Paulo	Paulistano/UNIFMU

Tabela 5: Times participantes e suas cidades e unidades federativas.

A distância média entre cada par de cidades é de 753 km. A maior distância é de 2027 km entre Torres e Brasília, enquanto que a menor é de 46 km entre São Paulo e Mogi das Cruzes.

Conforme indicação dos organizadores da CBB, neste trabalho assume-se que quando os times jogam dois jogos consecutivos fora de casa em um intervalo de no máximo quatro

dias (por exemplo, quinta e domingo, sexta e segunda, ou segunda e quinta), eles devem viajar diretamente da cidade de um oponente à cidade do outro. Caso contrário, o time visitante retorna da cidade do primeiro oponente para sua cidade de origem, e depois viaja de sua sede até a cidade do segundo adversário.

Na programação da tabela do Campeonato Nacional Masculino de Basquete há um número muito grande de restrições, pois trata-se de um problema real. Tais restrições envolvem questões de interesse dos clubes, da Confederação, das emissoras de televisão e dos patrocinadores. Além disso, o problema envolve ainda a minimização da distância total percorrida pelas equipes na tabela. Conforme notícia divulgada no dia 14 de abril de 2004 no sítio do jornal Folha de São Paulo (Folha Online) [13], pode-se comprovar que essa questão é de grande importância para os clubes e para os atletas, pois, conforme noticiado, a CBB cortou a concessão de passagens aéreas aos clubes, obrigando as equipes a viajarem de ônibus para jogar fora de casa. Ainda segundo a notícia, além do desgaste físico provocado pela duração e distância do percurso terrestre, os jogadores reclamam de falta de conforto nos ônibus, já que os atletas são em sua maioria de grande estatura e os assentos não oferecem boas condições para o descanso, o que tem gerado constantes reclamações e seguidas derrotas nas excursões dos times.

As restrições visam estabelecer um critério justo de distribuição dos jogos ao longo da temporada para todos os times participantes e proporcionar condições para atrair patrocinadores e emissoras de televisão. Algumas dessas restrições foram obtidas através de entrevistas junto à CBB e outras através da análise da tabela da fase de classificação de 2004 divulgada no sítio da CBB [27]:

1. Todo par de times tem que se enfrentar duas vezes, uma vez no estádio do primeiro e a outra no estádio do segundo.
2. Na primeira metade do campeonato (turno), cada time enfrenta todos os demais adversários uma única vez. Na segunda metade (returno), cada time volta a jogar contra todos os demais, mas com a inversão da sede de cada jogo em relação à primeira metade.
3. Cada time joga no máximo uma vez por dia.
4. Deve haver pelo menos um jogo toda sexta-feira e dois jogos todo domingo (restrição imposta pelas emissoras de televisão).
5. Deve haver pelo menos um jogo no Rio de Janeiro (Flamengo ou Tijuca jogando em casa) em todo final de semana (sexta, sábado ou domingo).

6. Cada time joga no máximo nove jogos consecutivos em casa.
7. Cada time joga no máximo sete jogos consecutivos fora de casa.
8. Todos os times têm que jogar no primeiro dia do campeonato.
9. Todos os times jogam seu último jogo no último dia da fase de classificação.
10. Cada time joga no máximo quatro vezes por semana.
11. Se um time tiver que viajar para realizar algum jogo em dias consecutivos (dois jogos fora de casa ou um jogo em casa e outro fora), a distância entre as cidades de origem e destino não pode ultrapassar a distância entre Brasília e Goiânia (209 km). Além disso, não pode haver uma outra viajem no dia que sucede esta viagem.
12. Cada time joga no máximo três jogos em casa em dias consecutivos.
13. Cada time deve jogar pelo menos seis jogos em casa nos finais de semana (sexta, sábado ou domingo) e no máximo 13.
14. Cada time tem que jogar pelo menos 15 jogos em finais de semana (sexta, sábado ou domingo).

Mesmo tendo sua tabela feita de forma manual e violando a restrição nove (por deixar um jogo de fora no último dia), o campeonato de 2004, segundo a CBB [27], teve 26% dos jogos transmitidos ao vivo pelos canais SPORTV e REDE TV. Nas emissoras de televisão, aberta e por assinatura, foram 373 horas, 55 minutos e 43 segundos entre reportagens e jogos transmitidos ao vivo e retransmissões. Ao todo, os jornais publicaram 157.034,5 cm, o que equivale a 491,5 páginas de um jornal no formato padrão. Esses números representam um retorno de mídia espontâneo de R$ 236.557.459,00, equivalente a US$ 80,187,126.

CAPÍTULO 3

Heurística proposta

3.1 Estratégia algorítmica

Para resolver o problema do Campeonato Nacional Masculino de Basquete de 2004 de maneira eficiente, deve-se produzir uma tabela que satisfaça todas as restrições definidas na Seção 2.4, minimizando a distância total percorrida pelos times na tabela.

Foram utilizadas técnicas heurísticas a fim de compor um algoritmo para solucionar o problema em questão. Heurísticas são algoritmos que visam encontrar boas soluções em tempo reduzido, sem, no entanto, garantir a otimalidade dessas soluções, nem tampouco prever quão próxima elas estão da solução ótima.

Metaheurísticas são procedimentos compostos por uma ou mais heurísticas destinados à encontrar uma boa solução, eventualmente ótima. Tais heurísticas devem ser adaptadas para cada problema específico. Contrariamente às heurísticas convencionais, as metaheurísticas são de caráter geral e têm condições de escapar de ótimos locais. Algoritmos genéticos [18], busca tabu [9], GRASP [19] e ILS [14] são exemplos de metaheurísticas amplamente empregadas na literatura.

A estratégica algorítmica empregada para resolver o problema do Campeonato Nacional Masculino de Basquete de 2004 baseia-se na metaheurística *Iterated Local Search* (ILS) (ver, por exemplo, Lourenço et al. [14]). Ela consiste basicamente em construir uma seqüência de soluções através da aplicação consecutiva de uma modificação ou de uma perturbação na solução atual, seguida pela aplicação de um procedimento de busca local.

A Figura 5 apresenta o pseudo-código da metaheurística ILS. De forma mais detalhada, a ILS inicia o processo de busca a partir de uma solução gerada por uma heurística construtiva e, em seguida, aprimorada por um método de busca local (linhas 1 e 2). Dada a solução atual, aplica-se uma perturbação que leva a uma solução intermediária, também chamada solução

perturbada (linha 4). Uma busca local é aplicada a esta solução, alcançando-se, assim, uma nova solução (linha 5). Se esta solução passar por um determinado critério de aceitação, ela torna-se a nova solução atual no processo de busca; caso contrário, retorna-se para aquela à qual foi aplicada a perturbação (linha 6). Estes passos são realizados iterativamente, até que a condição de parada seja verificada (linha 3).

```
procedimento ILS();
   1. s0 ← GeraSolucaoInicial()
   2. s* ← BuscaLocal (s0)
   3. enquanto condição de parada não encontrada faça
   4.    s' ← Perturbação (s*)
   5.    s' ← BuscaLocal (s')
   6.    s* ← Critério de aceitação (s*, s')
   7. fim-enquanto
   8. retorne s*
fim-ILS
```

Figura 5: Pseudo-código da metaheurística ILS.

Nas seções seguintes são descritos os principais componentes da estratégia algorítmica utilizada no problema da programação da tabela do Campeonato Nacional Masculino de Basquete de 2004.

3.2 Heurística construtiva

A solução inicial (passo 1 da Figura 5) é construída por uma heurística que visa atribuir um dia para cada um dos 240 jogos, de maneira que nenhum time tenha mais do que um jogo por dia, evitando assim que haja mais do que oito jogos em cada um dos 90 dias. Além disso, são satisfeitas as restrições 2, 8, 9 e 11, não permitindo que par algum de times se enfrente duas vezes no turno ou no returno, que time algum não jogue no primeiro e no último dia da fase de classificação do campeonato e que time algum tenha que realizar uma viagem em dois dias consecutivos cuja distância entre as cidades ultrapasse a distância entre Goiânia e Brasília (209 km).

3.2.1 Formalização

O Campeonato Nacional Masculino de Basquete de 2004 pode ser visto como um grafo $\vec{K}_n = (T, A)$ orientado completo, composto por um conjunto de nós $T = \{1, 2, ..., n\}$, onde $n = 16$, e um conjunto de arcos $A = \{1, 2, ..., m\}$, onde $m = 240$. Cada nó $i \in T$ representa um time e cada arco $a \in A$ representa um jogo entre os times associados aos nós adjacentes. O nó de origem de a representa o time visitante e o nó destino representa o time mandante. Nesse campeonato considera-se também o conjunto de dias $D = \{1, 2, ..., r\}$, onde $r = 90$, que representa os dias disponíveis para alocação dos jogos.

Como o Campeonato Nacional Masculino de Basquete de 2004 foi não compacto, uma tabela desse campeonato pode ser vista como uma [0,1]-fatoração ordenada $F = \{f_1, ..., f_r\}$ de K composta por r [0,1]-fatores, onde r representa o número de rodadas. Uma rodada d é uma porção de tempo em que um time joga no máximo uma vez. Já que, nesse campeonato, cada time pode jogar no máximo um jogo por dia, cada dia pode ser visto como uma rodada. Assim, cada [0,1]-fator corresponde a um dia (ou rodada) $d \in D$.

A solução inicial consiste em uma [0,1]-fatoração ordenada de K com exatamente r [0,1]-fatores, fazendo com que haja um [0,1]-fator para cada dia (ou rodada).

Já que todos os fatores de F são [0,1]-fatores, nenhum time terá mais de um jogo por dia, o que satisfaz a restrição 3. A restrição 1 também é satisfeita, pois a fatoração F é uma [0,1]-fatoração do grafo orientado completo K.

Como o problema possui muitas restrições e o algoritmo é sensível à solução inicial gerada, faz-se necessário criar soluções que atendam ao maior número possível de restrições. Para que isso seja possível, a [0,1]-fatoração é gerada pela heurística construtiva de forma que:

- O fator f_1 é obrigatoriamente um 1-fator, satisfazendo a restrição 8.
- O fator f_{90} é obrigatoriamente um 1-fator, satisfazendo a restrição 9.

3.2.2 Estruturas de dados e outras notações

Uma solução (ou tabela) para o Campeonato Nacional Masculino de Basquete de 2004 também pode ser representada por uma matriz quadrada Z de ordem n. Cada célula $[i][j]$ dessa matriz representa um jogo a ser realizado em um determinado dia $d \in D$ entre o time mandante $i \in T$ e o time visitante $j \in T$, com $i \neq j$.

A construção da solução inicial consiste então em atribuir um dia $d \in D$ aleatoriamente a cada célula de Z de tal forma que as restrições 1, 3, 8 e 9 sejam satisfeitas, conforme definido na Seção 3.2.1.

Para representar uma solução e otimizar o cálculo do número de violações das restrições, as seguintes estruturas de dados foram utilizadas:

1. Estrutura principal:

   É representada por uma matriz quadrada Z de ordem n, onde cada célula [i][j] representa o jogo do time i contra o time j. O valor dessa célula representa o dia d em que esses times jogam no estádio do time i, sendo assim um número entre 1 e 90. Os valores da diagonal principal não têm significado.

2. Estrutura auxiliar:

   É representada por uma matriz $Zaux$ n x r, onde cada célula [i][d] representa o adversário j do time i no dia d. Para o time mandante i, o valor da célula [i][d], será igual a j. Já para o time visitante j, o valor da célula [j][d] será o simétrico de i (isto é, -i).

A Figura 6 mostra um exemplo da estrutura principal que representa a solução associada à Tabela 4 da Seção 2.3.1 e a Figura 7 apresenta um exemplo da estrutura auxiliar associada à mesma tabela.

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

Figura 6: Exemplo da estrutura principal.

Time\Dia	1	2	3	4	5	6	7	8
1	3		-4	2	4	-3		-2
2	4	3		-1		-4	-3	1
3	-1	-2		4		1	2	-4
4	-2		1	-3	-1	2		3

Figura 7: Exemplo da estrutura auxiliar.

Além das estruturas que representam a solução, também são utilizadas outras estruturas, tais como:

- Um vetor que armazena o dia da semana (segunda-feira, terça-feira, etc.) de cada dia.
- Um vetor que indica a semana de cada dia.
- Uma matriz quadrada *H* de ordem *n* que armazena as distâncias entre as cidades sede dos times. A diagonal principal dessa matriz é preenchida com o valor zero, já que a distância entre uma cidade e ela mesma é zero.
- Um vetor que armazena o dia inicial de cada semana.

A restrição de jogos em turno e returno (restrição 2) também é considerada na construção da solução inicial, de forma que o jogo (*i*, *j*) é programado para o turno, caso o jogo (*j*, *i*) esteja no returno ou ainda não esteja programado. Caso contrário (jogo (*j*, *i*) programado para o turno), o jogo (*i*, *j*) será programado para um dia no returno. Para isso, os 90 dias são divididos em duas partes iguais, sendo os primeiros 45 dias considerados como turno e os 45 dias restantes considerados como returno. Isso torna o resultado do algoritmo mais restrito do que a tabela manual feita pela CBB, já que na tabela da CBB não há essa divisão exata dos 90 dias.

A restrição 11 também não é violada na solução inicial, pois tanto para o time *i* quanto para o time *j*, após a escolha de um dia *d* para o jogo (*i*, *j*) é verificado se as distâncias entre as cidades sede dos jogos anterior e posterior ao dia *d* são maiores do que a distância entre Goiânia e Brasília (209 km). Caso verdade, o dia *d* é rejeitado, sendo aceito caso contrário.

A Figura 8-a apresenta, de forma mais detalhada, o procedimento de criação da solução inicial representada pelas variáveis *Z* e *Zaux*. Ele inicia-se estabelecendo os jogos do primeiro e último dia, o que é feito pelos procedimentos `CriaPrimeiroDia` e `CriaUltimoDia` nas linha 1 e 2, respectivamente. Uma vez estabelecidos os jogos desses dias, o procedimento percorre a estrutura principal para atribuir um dia a cada jogo (células da estrutura principal). Esse percurso é feito a partir de dois laços que têm como índices as variáveis *i* e *j*, respectivamente (linhas 3 e 4). Para cada jogo (*i*, *j*), realiza-se a escolha aleatória de um dia *d* $\in$ *D*, onde os times i e j não tenham jogo nesse dia (linhas 7, 8 e 9). Após isso, verifica-se se esse jogo pode ser programado para o dia *d* quanto à restrição de turno e returno (restrição 2). Caso o jogo (*j, i*) esteja programado para o turno e o dia *d* também estiver no turno, o dia *d* será deslocado para o returno somando-se 45 ao valor de *d* (linhas 11, 12 e 13). De maneira análoga, se o dia *d* e o jogo (*j, i*) estiverem no returno (valores maiores que 45), desloca-se o jogo para o returno subtraindo-se 45 do valor de *d* (linhas 14, 15 e 16). Além disso, verifica-se também se o dia escolhido não viola a restrição 11. Essa verificação é realizada pelo

procedimento TrataRestrOnze, na linha 18. Caso a restrição 11 seja violada, o laço que se inicia na linha 6 é repetido até que o dia escolhido não viole a restrição 11. Uma vez que o dia não viole as restrições 2 e 11, as estruturas de dados principal e auxiliar são atualizadas (linhas 19, 20 e 21).

```
procedimento CriaSolucaoInicial(Z, Zaux);
1.    CriaPrimeiroDia(Z, Zaux)
2.    CriaUltimoDia(Z, Zaux)
3.    para i ← 1 até n faça
4.       para j ← 1 até n faça
5.          se (i <> j)
6.             faça
7.                faça
8.                   Escolher aleatoriamente d ∈ D
9.                enquanto ((Zaux[i][d] <> NULO) ou (Zaux[j][d] <> NULO))
10.               se Z[j][i] <> NULO
11.                  se (d <= r/2) e (Z[j][i] <= r/2)
12.                     d ← d + (r/2)
13.                  fim-se
14.                  se (d > r/2) e (Z[j][i] > r/2)
15.                     d ← d - (r/2)
16.                  fim-se
17.               fim-se
18.            enquanto (TrataRestrOnze(i, j, d, Zaux) = 1)
19.            Z[i][j] ← d
20.            Zaux[i][d] ← j
21.            Zaux[j][d] ← -i
22.         fim-se
23.      fim-para
24.   fim-para
25.   fim - CriaSolucaoInicial
```

Figura 8-a: Procedimento de criação da solução inicial e validação da restrição 2.

Já o procedimento TrataRestrOnze, descrito nas Figura 8-b e 8-c, se inicia verificando se há algum jogo fora de casa programado para o time *j* (time visitante) no dia imediatamente anterior ao dia *d* (linha 1). Se o valor da célula (*j*, *d*-1) na matriz auxiliar *Zaux* for menor do que zero, significa que o time *j* tem um jogo fora de casa no dia *d*-1. Se o valor dessa mesma célula for maior do que zero, significa que o time *j* tem um jogo em casa nesse dia. Caso haja jogo fora de casa no dia *d*-1, armazena-se o adversário de *j* nesse dia na variável *t* (linha 2). Se a distância entre a cidade sede do time *t* e a cidade sede do time *i* (local onde será realizado

o jogo (*i, j*) em questão) for maior que a distância entre Brasília e Goiânia (209 km), o procedimento retorna o valor 1, o que representa que a restrição 11 está sendo violada (linhas 3, 4 e 5). Caso o time *j* tenha algum jogo no dia *d*-2, o procedimento retorna o valor 1, indicando que a restrição 11 está sendo violada (linhas 6, 7 e 8). Nos passos seguintes, verifica-se se o time *j* tem um jogo dentro de casa programado para o dia *d*-1 (linha 10). Caso haja, verifica-se se a distância entre as cidades sede dos times *i* e *j* é maior do que a distância entre Brasília e Goiânia (linhas 11, 12 e 13), assim como, se há algum jogo fora de casa no dia *d*-2, retornando 1 nesses casos (linhas 14, 15 e 16). Para o time mandante *i*, verifica-se se há jogo fora de casa no dia *d*-1 (linha 18). Caso haja, armazena-se na variável *t* o time adversário de *i* nesse dia (linha 19). Em seguida, verifica-se se a distância entre as cidades sede dos times *t* e *i* é maior do que a distância entre Brasília e Goiânia (linha 20), e se o time *i* tem algum jogo no dia *d*-2 (linha 23), retornando o valor 1 nesses casos (linhas 21 e 24).

```
procedimento TrataRestrOnze (i, j, d, Zaux);
1.    se (Zaux[j][d-1] < 0)
2.       t ← -Zaux[j][d-1]
3.       se (H[t][i] > 209)
4.          retorne 1
5.       fim-se
6.       se (Zaux[j][d-2] <> NULO)
7.          retorne 1
8.       fim-se
9.    fim-se
10.   se (Zaux[j][d-1] > 0)
11.      se (H[j][i] > 209)
12.         retorne 1
13.      fim-se
14.      se (Zaux[j][d-2] < 0)
15.         retorne 1
16.      fim-se
17.   fim-se
18.   se (Zaux[i][d-1] < 0)
19.      t ← -Zaux[i][d-1]
20.      se (H[t][i] > 209)
21.         retorne 1
22.      fim-se
23.      se (Zaux[i][d-2] <> NULO)
24.         retorne 1
25.      fim-se
26.   fim-se
```

Figura 8-b: Início do procedimento que valida a restrição 11.

Nos demais passos, apresentados pela Figura 8-c (da linha 27 até a linha 52), os mesmos testes são realizados, só que ao invés de se considerar o dia imediatamente anterior a *d*, testa-se o dia imediatamente posterior a *d*. Por fim, caso nenhuma das condições acima seja verdadeira, o dia *d* é aceito, retornando-se o valor 0.

```
27.   se (Zaux[j][d+1] < 0)
28.      t ← -Zaux[j][d+1]
29.      se (H[t][i] > 209)
30.         retorne 1
31.      fim-se
32.      se (Zaux[j][d+2] <> NULO)
33.         retorne 1
34.      fim-se
35.   fim-se
36.   se (Zaux[j][d+1] > 0)
37.      se (H[j][i] > 209)
38.         retorne 1
39.      fim-se
40.      se (Zaux[j][d+2] < 0)
41.         retorne 1
42.      fim-se
43.   fim-se
44.   se (Zaux[i][d+1] < 0)
45.      t ← -Zaux[i][d+1]
46.      se (H[t][i] > 209)
47.         retorne 1
48.      fim-se
49.      se (Zaux[i][d+2] <> NULO)
50.         retorne 1
51.      fim-se
52.   fim-se
53.   retorne 0
fim - TrataRestrOnze
```

Figura 8-c: Continuação do procedimento que valida a restrição 11.

3.3 Busca local inicial

Uma busca local é um procedimento que visa encontrar uma solução melhor do que a solução corrente mediante a exploração de suas soluções vizinhas. Uma solução é dita vizinha se for possível produzi-la fazendo-se um determinado tipo de modificação na solução corrente. Tal modificação é conhecida como um movimento. A solução s^* obtida por uma

busca local é um ótimo local, isto é, uma solução para a qual não existe outra melhor do que ela na sua vizinhança.

Para o problema do Campeonato Nacional Masculino de Basquete de 2004 foram criadas duas buscas locais. Chama-se de busca local inicial àquela que é realizada imediatamente após a heurística construtiva (passo 2 da Figura 5), sendo, portanto, acionada uma única vez a cada execução do algoritmo. A solução corrente muda toda vez que é verificado que um movimento gera melhoria na solução corrente. Caso um movimento não gere melhoria na solução corrente, ela permanece inalterada até que se alcance um número máximo de movimentos sem melhorar a solução corrente.

Para verificar se um movimento gera melhoria na solução corrente, as restrições foram classificadas em fortes e fracas. As restrições 2, 8 e 9 foram classificadas como fracas, mantendo-se as demais restrições com fortes. Assim sendo, as restrições fortes predominam sobre as restrições fracas. Uma solução é considerada viável caso não viole nenhuma restrição forte e nenhuma restrição fraca.

A verificação de melhoria na solução corrente se dá através da avaliação lexicográfica do número de violações das restrições fortes, do número de violações das restrições fracas e do valor da função objetivo. O valor da função objetivo determina o custo de uma solução, que, por sua vez, é igual à soma das distâncias percorridas pelos times. Cada uma dessas três partes possui níveis de prioridades diferentes, conforme descrito na próxima seção.

Na busca local inicial, a estrutura de vizinhança utilizada é obtida alterando-se o dia de apenas um jogo escolhido de forma aleatória. Tal movimento é denominado *deslocamento*. Nesse movimento, um jogo (i, j) programado para um dia d passa a ser realizado no dia *dnovo* escolhido de forma aleatória, desde que os times i e j não tenham jogos programados para o dia *dnovo*. Essa estrutura de vizinhança corresponde à retirada de um arco do [0,1]-fator f_{d}, seguida da inserção desse mesmo arco no [0,1]-fator f_{dnovo}. Nesse caso, os nós i e j têm que ter grau nulo no [0,1]-fator f_{dnovo} (i e j não podem ter outro jogo no dia *dnovo*). A Figura 9 representa o movimento *deslocamento* nessa estrutura de vizinhança para $n = 4$ times, onde o jogo (3,4) foi movimentado do dia 4 para o dia 7.

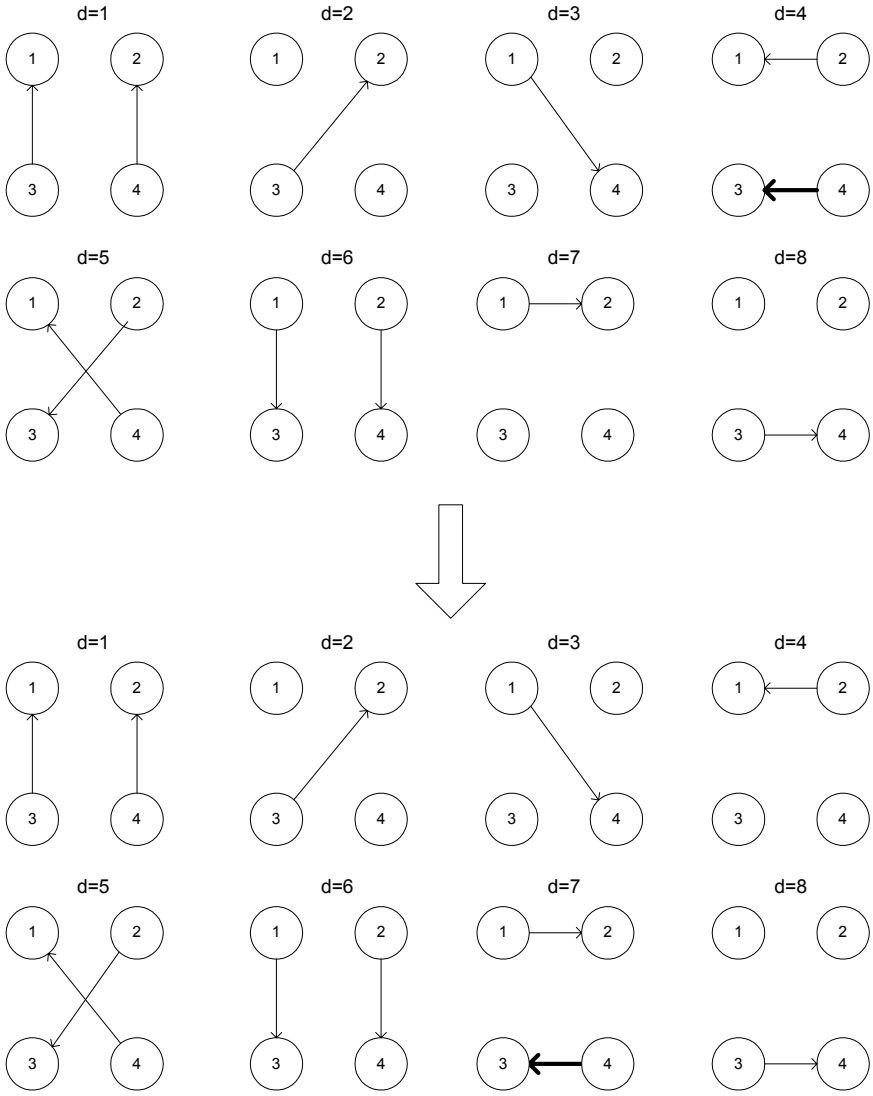

Figura 9: Movimento *deslocamento*, onde o jogo (3,4) é retirado do dia 4 e inserido no dia 7.

O pseudo-código do procedimento da busca local inicial é apresentado na Figura 10. Ele recebe como parâmetros as estruturas *Z* e *Zaux* que representam a solução corrente, a variável *x* que armazena o número de violações das restrições fortes da solução corrente, a variável *y* que armazena o número de violações das restrições fracas da solução corrente, a variável *w* que guarda a distância total da solução corrente e *maxit* que representa o número máximo de iterações consecutivas sem melhoria. A busca começa atribuindo o valor zero para a variável *iteracoes_sem_melhoria* para controlar a iteração da busca local (linha 1), que é encerrada quando *iteracoes_sem_melhoria* é igual a *maxit*. O primeiro passo de cada iteração é a escolha aleatória de um jogo entre um time mandante *i* (linha 3) e um time visitante *j* (linha 4). Após a escolha de um jogo (*i, j*), determina-se aleatoriamente um novo dia para esse jogo. A escolha desse novo dia *dnovo* é feita com um laço que é executado até que os times *i* e *j* não tenham jogo programado para o dia *dnovo* (linhas 5, 6 e 7). Após a escolha do novo dia

dnovo para o jogo (*i, j*), guarda-se o dia em que esse jogo estava programado *d* (linha 8). O procedimento `Houvemelhoria` é acionado em seguida para testar se o movimento de *deslocamento* do jogo (*i, j*) do dia *d* para o dia *dnovo* gera alguma melhoria na solução. O procedimento `Houvemelhoria` é descrito com detalhes na próxima seção. Em caso de melhoria, é atribuído o valor zero para a variável *iteracoes_sem_melhoria* (linha 10) e as estruturas de dados principal e auxiliar são atualizadas (linhas 11 a 15). Caso contrário, o valor da variável *iteracoes_sem_melhoria* é incrementado (linha 17) para que o número de iterações do procedimento da busca local inicial possa ser novamente testado, até que se alcance o número máximo de iterações permitido sem melhorias (linha 2).

```
procedimento BuscaLocalInicial(Z, Zaux, x, y, w, maxit);
1.    iteracoes_sem_melhoria ← 0
2.    enquanto iteracoes_sem_melhoria < maxit
3.        Escolher aleatoriamente i ∈ T
4.        Escolher aleatoriamente j ∈ T, sendo j ≠ i
5.        faça
6.            Escolher aleatoriamente dnovo ∈ D
7.        enquanto ((Zaux[i][dnovo] <> NULO) ou (Zaux[j][dnovo] <> NULO))
8.        d ← Z[i][j]
9.        se Houvemelhoria(Z, Zaux, x, y, w, i, j, d, dnovo)
10.           iteracoes_sem_melhoria ← 0
11.           Z[i][j] ← dnovo
12.           Zaux[i][d] ← NULO
13.           Zaux[j][d] ← NULO
14.           Zaux[i][dnovo] ← j
15.           Zaux[j][dnovo] ← -i
16.       senão
17.           iteracoes_sem_melhoria ← iteracoes_sem_melhoria + 1
18.       fim-se
19.   fim-enquanto
fim - BuscaLocalInicial
```

Figura 10: Pseudo-código da busca local inicial.

Ao final da busca local inicial é obtida uma solução melhor ou equivalente à solução inicial em termos de avaliação lexicográfica. Isso é garantido pelo procedimento `Houvemelhoria`, que tem como finalidade verificar se o movimento de deslocamento produz melhoria na solução considerada, sendo aceito somente caso verdade. Experimentos

computacionais mostraram que, dependendo do valor de *maxit*, ao final da busca local inicial é possível obter uma solução viável, ou seja, sem violar restrição alguma, assim como reduzir significativamente a distância total da solução.

A seguir são descritos os principais componentes da busca local.

3.3.1 Comparação de soluções

A comparação de soluções vizinhas é feita lexicograficamente pelo número de violações das restrições fortes, pelo número de violações das restrições fracas e pela soma das distâncias percorridas pelos times na solução. Uma redução no número de violações das restrições fracas e na soma das distâncias, por maior que seja, não compensa um aumento no número de violações das restrições fortes. Assim como, uma redução na soma das distâncias, por maior que seja, não compensa um aumento no número de violações das restrições fracas. Portanto, as restrições fortes predominam sobre as fracas, assim como as restrições fracas predominam sobre a soma das distâncias percorridas pelos times. Uma solução é considerada viável caso o número de restrições violadas seja igual a zero.

Observa-se que uma vez que uma solução viável (sem violar restrição alguma) é encontrada, torna-se necessário minimizar a soma das distâncias percorridas pelos times (função objetivo).

Um movimento realizado em uma solução viável sempre produzirá uma solução inviável se:

- O movimento envolver algum jogo programado para o primeiro dia (restrição 8).
- O movimento envolver algum jogo programado para o último dia (restrição 9).
- O movimento modificar algum jogo do turno para o returno ou vice-versa (restrição 2).

Para evitar que tal situação ocorra e permitir que o algoritmo percorra de maneira completa o conjunto de soluções viáveis (espaço de busca), as restrições 2, 8 e 9 foram relaxadas, isto é, foram classificadas como fracas, mantendo-se as demais restrições consideradas como fortes. Assim sendo, as restrições fortes predominam sobre as fracas. A classificação das restrições está resumida na Tabela 6.

Restrição	Tipo
1	Forte
2	Fraca
3	Forte
4	Forte
5	Forte
6	Forte
7	Forte
8	Fraca
9	Fraca
10	Forte
11	Forte
12	Forte
13	Forte
14	Forte

Tabela 6: Classificação das restrições em fortes e fracas.

A variação no custo de uma solução viável decorrente de um movimento é dada pela soma das distâncias que cada time percorre na solução.

O procedimento `Houvemelhoria`, acionado no passo 9 do procedimento da busca local inicial descrito na seção anterior, determina se o movimento de deslocamento reduz o número de violações das restrições fortes, das restrições fracas ou da distância total da solução. Para isso, ele recebe como parâmetros as estruturas *Z* e *Zaux* que representam a solução corrente, a variável *x* que armazena o número de violações das restrições fortes da solução corrente, a variável *y* que armazena o número de violações das restrições fracas da solução corrente, a variável *w* que guarda a distância total da solução corrente, o jogo *(i, j)* que será verificado, o dia *d* em que o jogo (*i, j*) está programado e o dia *dnovo* que será testado como alternativa para esse jogo.

O teste consiste em recalcular o número de violações das restrições fortes *x'*, o número de violações das restrições fracas *y'* e a distância total da solução *w'* decorrente da retirada do jogo (*i, j*) do dia *d* seguida da sua inserção no dia *dnovo*. Considera-se que um movimento de

deslocamento gera melhoria em uma solução quando uma das seguintes condições for verdadeira:

1. *x'* for menor do que *x*.
2. *x'* for igual a *x* e *y'* for menor do que *y*.
3. *x'* for igual a *x*, *y'* for igual a *y* e *w'* for menor do que *w*.

Caso alguma das condições acima seja verdadeira, o procedimento `houvemelhoria` retorna verdadeiro, indicando que houve melhoria; caso contrário retorna falso, indicando que não houve melhoria. É importante salientar que o recálculo de *y'* só é realizado se $x' \leq x$, assim como o recálculo de *w'* só é realizado se $x' \leq x$ e $y' \leq y$.

3.3.2 Reavaliação de soluções vizinhas

Durante a busca local, é necessário avaliar diversas soluções vizinhas, como pode ser observado na linha 9 do procedimento representado na Figura 10. Portanto, torna-se necessário que essa avaliação seja feita de maneira eficiente, o que se agrava pelo número de restrições envolvidas no problema do Campeonato Nacional Masculino de Basquete de 2004.

Quando, na busca local, um novo dia *dnovo* é escolhido para o jogo (i, j), as estruturas de dados não são alteradas. A reavaliação é feita verificando-se o efeito que a movimentação do jogo (i, j) do dia *d* para o dia *dnovo* gera na solução corrente. A reavaliação consiste em três passos: determinar o número *x'* de violações das restrições fortes da solução vizinha, determinar o número *y'* de violações das restrições fracas da solução vizinha e determinar a distância total *w'* percorrida na solução vizinha. A avaliação de melhoria na solução corrente é feita através da comparação lexicográfica de *x'*, *y'* e *w'* com *x*, *y*, e *w*. Somente se houver melhoria na solução corrente, as estruturas de dados são alteradas para refletir o movimento realizado.

A avaliação das violações da maioria das restrições fortes é feito em tempo $O(1)$. Isto é possível para as restrições 4, 5, 10, 11, 12, 13 e 14, pois, nesses casos, são armazenados na memória a quantidade de jogos em finais de semana para cada time, a quantidade de jogos em casa em finais de semana para cada time, a quantidade de jogos de cada time em cada semana e a quantidade de jogos por dia. Esses valores permitem comparações simples. Por exemplo, para saber se um movimento viola a restrição de no mínimo seis jogos em casa em finais de semana para cada time (restrição 13), verifica-se o dia da semana referente ao dia do qual o jogo está sendo retirado e o dia da semana do qual o jogo está sendo inserido.

Esse recálculo é feito através da comparação entre o dia *d* (dia de onde o jogo (*i, j*) está sendo retirado) com o dia *dnovo* (dia onde o jogo (*i, j*) está sendo inserido). Caso o dia *d* caia em um final de semana (sexta, sábado ou domingo) e o dia *dnovo* não caia em um final de semana (segunda a quinta), verifica-se se a quantidade de jogos em casa em finais de semana do time *i* (time mandante) é menor ou igual a seis. Caso verdade, incrementa-se o valor de *x'* em uma unidade, indicando que esse deslocamento provoca uma violação da restrição 13. Analogamente, para verificar se o movimento melhora a solução para a restrição 13, verifica-se se o dia *d* não cai em um final de semana, se o dia *dnovo* cai em um final de semana e se a quantidade de jogos em casa em finais de semana para o time *i*, é menor do que seis. Nesse caso, o valor de *x'* é decrementado de uma unidade, indicando que esse deslocamento provoca uma melhoria na solução quanto à restrição 13.

As violações das restrições 4, 5, 10, 11 e 14 são avaliadas de forma semelhante.

As restrições 1 e 3 não são recalculadas, já que cada solução corresponde a uma [0,1]-fatoração ordenada do grafo orientado completo. Além disso, as estruturas de dados utilizadas não permitem que essas restrições sejam violadas.

As demais restrições fortes (6 e 7) têm suas violações recalculadas em tempo $O(r)$ no pior caso, onde *r* corresponde ao número de dias. Entretanto, na maioria dos casos o recálculo se comporta como se fosse $O(1)$. Isto ocorre porque nessas restrições é necessário percorrer a estrutura auxiliar para verificar a ocorrência de jogos em casa ou fora para os times envolvidos no movimento e esse percurso é limitado a *r* no pior caso (*r* representa o número de dias). No entanto, mesmo para soluções com muitas restrições violadas, tal situação dificilmente ocorrerá, pois seria necessário que todos os jogos fossem seguidos em casa ou fora. Além disso, assim que uma solução se torna viável, o recálculo dessas restrições é executado em tempo constante, já que há um limite de nove jogos consecutivos em casa (restrição 6) e de sete jogos consecutivos fora de casa (restrição 7). Esse caso torna-se bastante comum durante a segunda busca local, que é executada freqüentemente pelo algoritmo.

Quanto às restrições fracas, todas as violações são avaliadas em tempo $O(1)$ sem necessidade de se armazenar qualquer valor adicional em memória, já que para avaliar se um determinado movimento viola, por exemplo, a restrição 8 (todos os times tem que jogar no primeiro dia), basta verificar se o dia do qual o jogo está saindo é o primeiro dia, uma vez que o dia no qual o jogo está sendo inserido é sempre diferente do dia anterior ao movimento. Nesse caso, o contador de violações de restrições fracas é incrementado.

Já o recálculo da distância se baseia em duas seqüências de dias:

- Seqüência 1: intervalo de dias entre o dia do jogo anterior ao dia em que o jogo (*i, j*) será removido (dia *d*) e o dia do jogo posterior ao dia em que o jogo (*i, j*) será removido.
- Seqüência 2: intervalo de dias entre o dia anterior ao dia em que o jogo (*i, j*) será inserido (dia *dnovo*) e o dia do jogo posterior ao dia em que o jogo (*i, j*) será inserido.

A Figura 11 apresenta o pseudo-código do procedimento `RecalculaDistancia` referente à seqüência 1.

O procedimento `RecalculaDistancia` começa atribuindo à variável *w'* a distância total percorrida na solução atual (armazenado em *w*) na linha 1 e à variável *passoucarnaval* o valor falso na linha 2. Em seguida inicia-se um laço para determinar o dia do jogo antecessor ao jogo (*i, j*) na linha 6. Durante esse laço, caso o Carnaval ocorra durante a seqüência 1, a variável *passoucarnaval* recebe o valor verdadeiro (linhas 7, 8 e 9). Ao final desse laço, a variável *dant* indica o dia do jogo antecessor ao jogo (*i, j*). Para o time mandante *i*, caso esse jogo seja fora de casa (linha 12), subtrai-se da distância atual *w'*, a distância entre a sede do adversário do time *i* no dia *dant* e a sede do time *i*, uma vez que, com o deslocamento do jogo (*i, j*) para o dia *dnovo*, essa viagem não será realizada (linha 14).

De maneira análoga, o mesmo procedimento é feito para a seqüência de dias entre o dia *d* e o dia do jogo posterior ao jogo *(i, j*), ou seja, determina-se o dia do jogo posterior ao jogo (*i, j*) a partir do laço na linha 17 verificando a ocorrência do Carnaval e armazenando esse dia na variável *dpos*. Caso esse jogo seja fora de casa, subtrai-se de *w'* a distância entre a sede do time *i* e a cidade sede do time adversário de *i* nesse dia (linhas 24 e 25), uma vez que essa viagem também não será realizada com o deslocamento.

Conforme indicação da CBB, quando os times jogam dois jogos consecutivos fora de casa em um intervalo maior ou igual a quatro dias, eles devem retornar da cidade do primeiro oponente para sua cidade de origem e depois viajar de sua sede até a cidade do segundo adversário. Assim, com o deslocamento, caso a diferença entre o dia do jogo anterior e o dia do jogo posterior ao jogo (*i, j*) seja maior que quatro (representado pela constante *NDIASVIAGEMDIRETO)* ou haja Carnaval nesse período (linha 27), o time *i* deverá viajar da cidade sede do adversário do jogo anterior ao jogo (*i, j*) para a sua cidade sede, para depois viajar da sua cidade sede para a cidade sede do adversário do jogo posterior ao jogo (*i, j*). Nesse caso, deve-se somar a *w'* as distâncias referentes a essas viagens (linhas 28 e 29). Entretanto, caso não haja Carnaval nesse período e a diferença entre o dia do jogo anterior e posterior ao jogo (*i, j*) seja menor do que quatro, o time *i* deve viajar da cidade sede do

primeiro adversário diretamente para a cidade sede do segundo adversário. Logo, a distância entre essas cidades deve ser somada à *w'* (linha 31). É importante ressaltar que, caso o jogo anterior ou posterior ao jogo (*i, j*) seja em casa, o valor das variáveis *adversárioant* e *adversáriopos* serão iguais a *i* (conforme atribuição efetuada nas linhas 3 e 4), o que fará com que as operações de soma e subtração de distâncias atribuídas a *w'* nas linhas 28 e 29 sejam feitas a partir do valor zero, já que a diagonal principal da matriz de distâncias *H* é preenchida com valor zero, conforme descrito na Seção 3.2.2.

```
procedimento RecalculaDistancia (i , j, Zaux, w, d, dnovo);
1.    w' ← w
2.    passoucarnaval ← falso
3.    adversarioant ← i
4.    adversariopos ← i
5.    dant ← d - 1
6.    Enquanto (Zaux[i][dant] = NULO)  e (dant <> 1)
7.       se (dant = DIAINICIOCARNAVAL) ou (dant = DIAFIMCARNAVAL)
8.              passoucarnaval ← verdadeiro
9.       fim-se
10.      dant ← dant - 1
11.   fim-enquanto
12.   se (Zaux[i][dant] < 0)
13.      adversarioant ← -Zaux[i][dant]
14.      w' ← w' - H[adversarioant][i]
15.   fim-se
16.   dpos ← d + 1
17.   Enquanto (Zaux[i][dpos] = NULO)  e (dpos <> r)
18.      se (dpos = DIAINICIOCARNAVAL) ou (dpos = DIAFIMCARNAVAL)
19.             passoucarnaval ← verdadeiro
20.      fim-se
21.      dpos ← dpos + 1
22.   fim-enquanto
23.   se (Zaux[i][dpos] < 0)
24.      adversariopos ← -Zaux[i][dpos]
25.      w' ← w' - H[adversariopos][i]
26.   fim-se
27.   se (dpos - dant >= NDIASVIAGEMDIRETO) ou passoucarnaval
28.      w' ← w' + H[adversarioant][i]
29.      w' ← w' + H[i][adversariopos]
30.   senão
31.      w' ← w' + H[adversarioant][adversariopos]
32.   fim-se
33.   retorne w'
fim - RecalculaDistancia
```

Figura 11: Procedimento de recálculo da distância.

Para a seqüência 2, o procedimento `RecalculaDistancia` efetua os mesmos passos, mas nesse caso, consideram-se as seqüências de dias dos jogos anterior e posterior ao dia *dnovo* ao invés do dia *d*. Além disso, esse mesmo procedimento também realiza os mesmos passos para o time visitante *j*.

A Figura 12 representa o mecanismo do recálculo da distância para um time hipotético *j*. Considera-se que em qualquer seqüência, a diferença de dias entre os jogos é sempre menor do que quatro (o que não obriga *j* a voltar a sua sede). Antes de realizar o movimento do jogo (*i, j*) da seqüência 1 para a seqüência 2, nota-se pela seqüência 1, que o time *j* deve viajar da sua cidade sede (cidade j) até a cidade *i,* e posteriormente, da cidade *i* até a cidade *l*. Na seqüência 2, nota-se que há uma viagem da cidade *g* até a cidade *z*. Depois do movimento, na seqüência 1 tem-se apenas o percurso da cidade *j* até a cidade *l*, enquanto que na seqüência 2 tem-se o percurso da cidade *g* para a cidade *i* e posteriormente da cidade *i* para a cidade *z*.

Assim sendo, a distância percorrida pelo time *j* seria recalculada a partir do valor da distância anterior ao movimento, subtraído das distâncias entre as cidades *j-i*, *i-l* e *g-z* e acrescido das distâncias entre as cidades *j-l*, *g-i* e *i-z*.

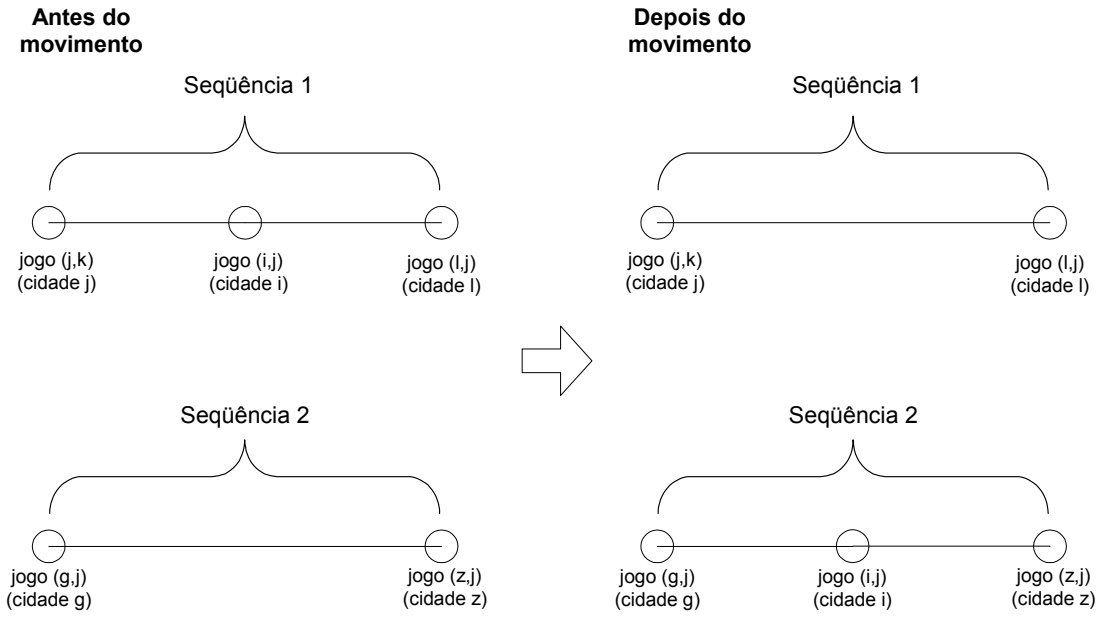

Figura 12: Ilustração do recálculo da distância.

3.4 Perturbação

A fim de explorar novas soluções dentro do espaço de busca, uma estratégia de diversificação é incorporada ao algoritmo. De acordo com Reeves [18], as estratégias de diversificação visam gerar soluções que tenham atributos significativamente diferentes daqueles encontrados nas melhores soluções obtidas.

Essa estratégia é realizada mediante um procedimento de perturbação na solução obtida pela busca local. O procedimento de perturbação visa produzir uma nova solução *s'* a partir de uma ou mais modificações na solução corrente *s** (passo 4 da Figura 5). Como a freqüência de execução desse procedimento é muito menor do que os recálculos efetuados na busca local, não é necessário que o recálculo do número de violações das restrições fortes, fracas e da distância sejam executados em tempo constante.

As estruturas de vizinhança utilizadas nas perturbações são baseadas nos seguintes movimentos:

1. *Deslocamento com troca de dias* – Nesta vizinhança, a modificação corresponde a uma certa quantidade de movimentos de *deslocamento* seguida de movimentos de *troca de dias*. O movimento de *troca de dias* consiste em inverter os dias de dois jogos aleatórios (i, j) e (p, q). Nesse movimento, é necessário que os times i, j, p e q sejam distintos, o que garante que a restrição 3 seja satisfeita.

Uma *troca de dias* pode ser vista como a aplicação de dois *deslocamentos*, onde o jogo (i, j) é deslocado para o dia do jogo (p, q) e vice-versa. Nesse caso, é necessário que os times i e j não tenham nenhum jogo programado para o dia do jogo (p, q) e que os times p e q não tenham nenhum jogo programado para o dia do jogo (i, j). Assim sendo, o recálculo do número de violações das restrições fortes, fracas e da distância total é feito em cada deslocamento, ou seja, primeiro recalcula-se o número de violações das restrições fortes, o número de violações das restrições fracas e a distância total. Em seguida desloca-se o jogo (i, j) para o dia do jogo (p, q). Posteriormente, recalcula-se o número de violações das restrições fortes, o número de violações das restrições fracas, e a distância total, deslocando-se em seguida, o jogo (p, q) para o dia do jogo (i, j).

A *troca de dias* corresponde à troca de dois arcos entre dois [0,1]-fatores diferentes, sendo os nós de cada um desses arcos distintos. A Figura 13 ilustra o movimento *troca de dias* em uma [0,1]-fatoração com seis [0,1]-fatores de seis nós.

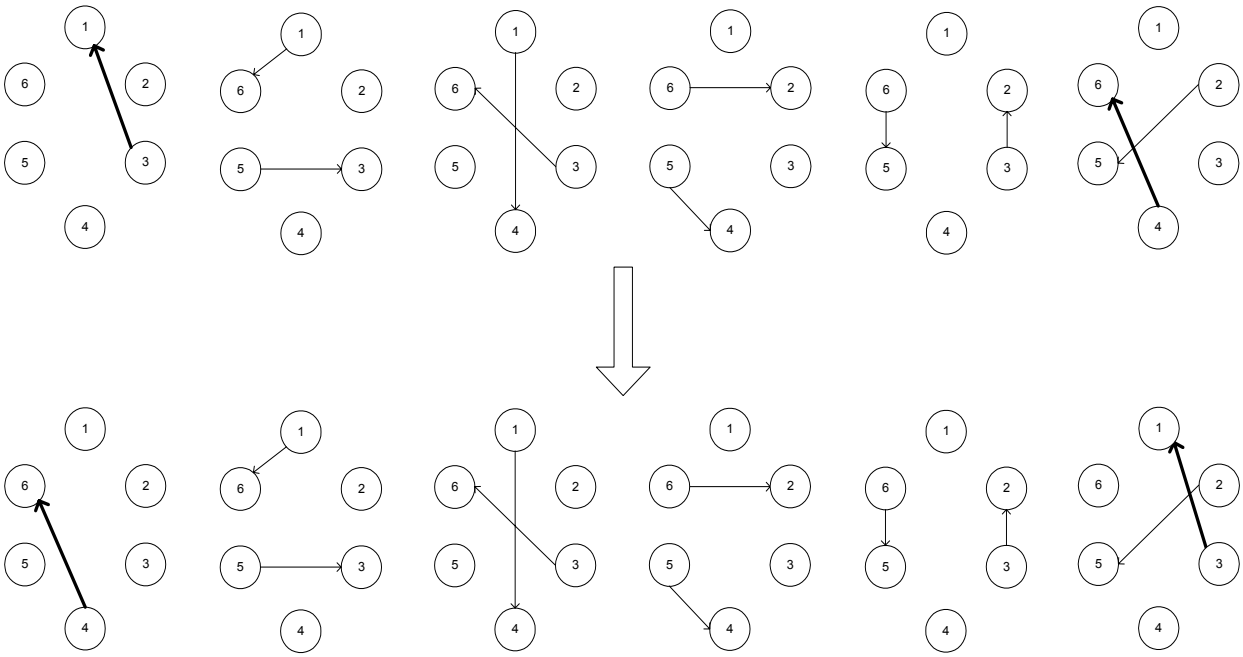

Figura 13: Troca de dias envolvendo os jogos (1,3) e (6,4).

2. *Troca de semanas* – Uma semana é composta por um conjunto de dias. Semanas também podem ser vistas como um conjunto de fatores ordenados. Na vizinhança *troca de semanas* cada solução é obtida com a troca de todos os jogos de duas semanas aleatórias s_1 e s_2. As semanas envolvidas na troca também devem ser distintas.

O movimento é realizado em tempo $O(n)$, pois para cada dia das semanas envolvidas na troca, todos os n times podem ter seus jogos trocados. Uma vez realizado o movimento, é necessário refazer o cálculo do número de violações das restrições fortes, do número de violações das restrições fracas e da distância percorrida pelos times na solução.

A *troca de semanas* corresponde à troca da ordem dos [0,1]-fatores que correspondem às semanas s_1 e s_2, mantendo a ordem desses [0,1]-fatores dentro de cada semana. A Figura 14 representa o movimento de *troca de semanas* para uma [0,1]-fatoração com três semanas, onde as semanas s_1 e s_3 são trocadas. É importante ressaltar que a ordem dos [0,1]-fatores dentro de cada semana é mantida.

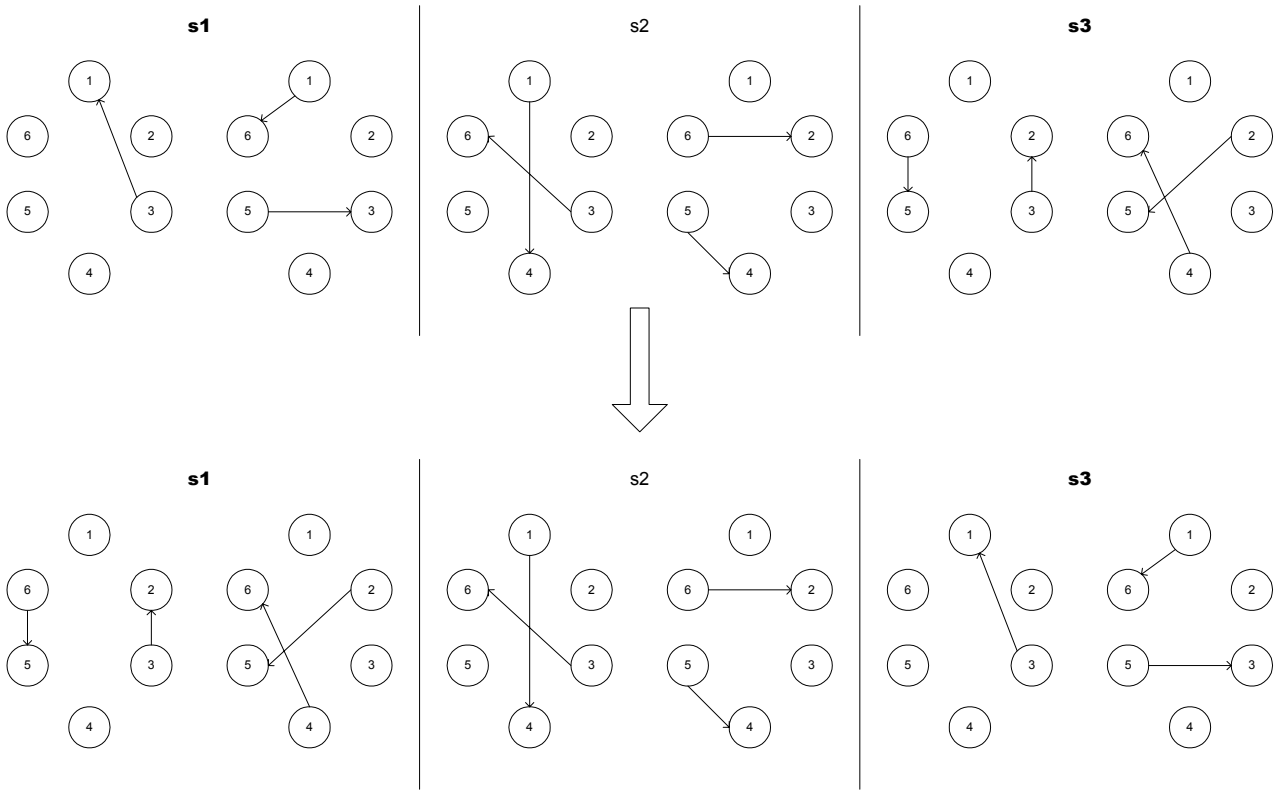

Figura 14: Troca de semanas envolvendo as semanas s_1 e s_3.

3. *Troca de times* – Nesta vizinhança, cada solução é obtida a partir da troca de todos os jogos de dois times distintos *i* e *j*. Se em um determinado dia o time *p* joga contra o time *i* em casa, após a troca o time *p* jogará em casa contra o time *j* nesse mesmo dia.

O tempo de execução do movimento é $O(r)$, pois todos os r [0,1]-fatores que representam a solução devem ser percorridos para a troca dos nós envolvidos. Depois do movimento é refeito o cálculo das distâncias percorridas por todos os times na solução, assim como o cálculo das violações das restrições fortes de toda a solução. O número de violações das restrições fracas não se altera com esse movimento, logo não é necessário realizar nenhum tipo de cálculo de violações dessas restrições, sendo mantido o número de violações da solução corrente.

A *troca de times* corresponde à troca dos nós que representam os times *i* e *j* em todos os [0,1]-fatores da [0,1]-fatoração. A Figura 15 representa o movimento de *troca de times* para uma [0,1]-fatoração com [0,1]-fatores de seis nós onde os times 1 e 4 têm seus jogos trocados.

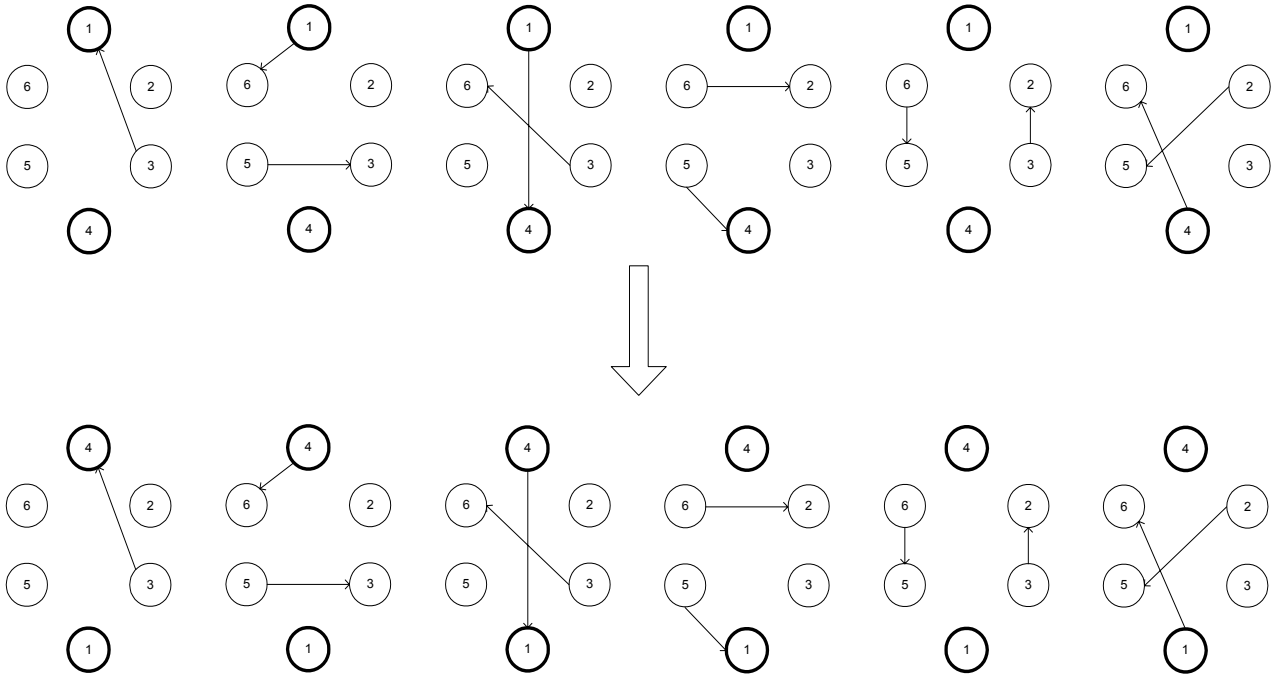

Figura 15: Troca de times envolvendo os times 1 e 4.

A cada execução do procedimento de perturbação, é utilizada apenas uma das estruturas de vizinhança citadas acima. O objetivo é trocar sistematicamente as vizinhanças utilizadas na busca, de forma que as vizinhanças cresçam na medida em que não se obtém melhoria na solução corrente. O princípio é iniciar a perturbação utilizando estruturas menores, com soluções mais próximas da solução corrente, e passar a explorar soluções mais distantes quando não se consegue melhoria na atual vizinhança da solução. Esse princípio é similar ao da metaheurística *Variable Neighborhood Search* (VNS) [11].

Para determinar a escolha da estrutura de vizinhança utiliza-se a variável *hist* e os parâmetros β, α e Δ, de tal forma que $\beta < \alpha < \Delta$. A variável *hist* representa o histórico das estruturas utilizadas e os valores β, α e Δ definem limites para a utilização de cada estrutura de vizinhança. A variável *hist* é inicializada com o valor zero no início do procedimento ILS (Figura 5 na Seção 3.1). A cada iteração do procedimento ILS sem melhoria na solução corrente, *hist* é incrementada em uma unidade até alcançar o valor do parâmetro Δ (quando é reinicializada). Quando há melhoria na solução corrente, *hist* é reinicializada com o valor zero. De uma forma geral, a escolha da estrutura de vizinhança se dá da seguinte forma.

- *Deslocamento com troca de dias*: quando $0 \leq hist < \beta$
- *Troca de times*: quando $\beta \leq hist < \alpha$
- *Troca de semanas*: quando $\alpha \leq hist < \Delta$.

Após aplicado o critério de aceitação, descrito na Seção 3.6 deste capítulo, verifica-se se *hist* deve ser incrementado (o que ocorre quando não há melhoria na solução corrente) ou reinicializado (o que ocorre quando há melhoria na solução corrente). Se a solução obtida após a busca local for aceita, passa-se a explorar a vizinhança de menor ordem dessa nova solução. Caso contrário, explora-se uma estrutura de vizinhança de mais alta ordem, com soluções mais distantes da solução atual.

Para controlar a amplitude da estrutura de vizinhança *deslocamento com troca de dias*, utiliza-se o parâmetro γ. O objetivo é efetuar poucos movimentos no início da perturbação e aumentar gradualmente o número de movimentos aplicados (mesmo princípio da metaheurística VNS [11]). Inicia-se a perturbação efetuando-se apenas um *deslocamento com troca de dias* em até γ iterações do procedimento `ILS` (Figura 5) enquanto não houver melhoria na solução corrente. Após essas iterações, caso ainda não haja melhoria na solução corrente, nas γ iterações seguintes do procedimento `ILS` serão efetuados dois *deslocamentos com troca de dias* enquanto não houver melhoria na solução corrente. O número de movimentos cresce até que sejam alcançadas β iterações do procedimento `ILS` sem gerar melhoria na solução corrente, o que faz com que seja utilizada a vizinhança *troca de times*. Se, por exemplo, γ = 3 e β = 9, quando o procedimento de perturbação for acionado pela primeira vez, será efetuado um *deslocamento com troca de dias.* Enquanto não houver melhoria na solução corrente, essa quantidade de movimentos será feita até três vezes (γ = 3). Se depois disso ainda não houver melhoria na solução corrente, serão efetuados dois *deslocamentos com troca de dias.* Esses dois *deslocamentos com troca de dias* serão efetuados até três vezes, enquanto não houver melhoria na solução corrente. Se depois disso ainda não houver melhoria na solução corrente, serão efetuados três *deslocamentos com troca de dias* que, por sua vez, continuarão sendo efetuados até três vezes, enquanto não houver melhoria na solução corrente. Depois disso, caso ainda não haja melhoria na solução corrente, será utilizada a estrutura de vizinhança *troca de times*, já que o número de iterações sem melhoria na solução corrente será igual a β. É importante salientar que quando há melhoria na solução corrente, na iteração seguinte será efetuado um *deslocamento com troca de dias*, já que a variável *hist* será reinicializada com o valor zero.

A Figura 16 ilustra o procedimento de perturbação de forma mais detalhada. Ele inicia verificando se o valor de *hist* é menor do que o parâmetro β (linha 1). Caso o valor de *hist* seja menor do que β, será utilizada a estrutura de vizinhança de *deslocamentos com troca de dias.* Antes de se realizar o movimento, verifica-se o número de movimentos que serão realizados. Isto é limitado pelo conteúdo da variável *vezes* que é inicializada dividindo-se o valor da

variável *hist* pelo parâmetro γ e somando-se um posteriormente (linha 2). Por exemplo, se *hist* = 12 e γ = 3, serão realizados cinco movimentos de *deslocamento com troca de dias*, pois (12 / 3) + 1 = 5. Cada movimento é realizado em uma iteração do laço que é executado enquanto o valor da variável *vezes* for maior do que zero (linhas 3, 4 e 5). Caso o valor de *hist* seja menor do que o parâmetro α (linha 8), onde β < α < Δ, será efetuado um movimento de troca de times (linha 9). Caso contrário, será efetuado um movimento de troca de semanas (linha 11).

```
procedimento PerturbaSolucao (Z, Zaux, hist);
1.    se (hist < β)
2.       vezes ← ⌊(hist/γ)⌋ + 1
3.       Enquanto (vezes > 0) faça
4.          Realiza DeslocamentoComTrocadeDias(Z, Zaux)
5.          vezes ← vezes - 1
6.       fim-enquanto
7.    senão
8.       se (hist < α)
9.          Realiza TrocadeTimes(Z, Zaux)
10.      senão
11.         Realiza TrocadeSemanas(Z, Zaux)
12.      fim-se
13.   fim-se
fim - PerturbaSolucao
```

Figura 16: Procedimento de perturbação.

Os valores dos parâmetros γ, α, β e Δ foram objetos de experimentos computacionais apresentados no Capítulo 4, assim como a ordem de precedência das estruturas de vizinhança.

3.5 Segunda busca local

Chama-se de segunda busca local àquela que é acionada após o procedimento de perturbação em cada iteração do procedimento `ILS` (passo 5 da Figura 5).

O segundo procedimento de busca local visa encontrar um ótimo local a partir da solução perturbada *s'* utilizando movimentos de deslocamento. A diferença em relação à busca local inicial está no fato de que quando um movimento é avaliado e verifica-se que ele

não melhora a solução corrente, o dia seguinte ao dia escolhido é testado. Todos os outros dias seguintes são avaliados até que todos tenham sido testados de forma cíclica (depois que o último dia é testado, verifica-se o primeiro dia), ou quando se verifica que o deslocamento gera melhoria na solução corrente (primeiro movimento aprimorante). Além disso, esse procedimento é executado para todos os jogos da solução corrente até que não haja melhoria com o deslocamento de nenhum jogo da solução.

A Figura 17 apresenta de forma mais detalhada o procedimento da segunda busca local. Conforme dito anteriormente, o objetivo desse procedimento é verificar se o deslocamento de cada jogo da solução corrente para outro dia gera melhoria na solução. O primeiro jogo (*i, j*) é escolhido aleatoriamente na linha 1, isto é, escolhe-se um time mandante *i* e um time visitante *j*. Após essa escolha, a variável *jogo_inicial* guarda esse jogo (linha 2). Inicia-se então o laço do procedimento de busca (linha 3) para verificar se o deslocamento de cada jogo para outro dia gera melhoria na solução corrente. Este teste é feito a partir da escolha de um dia *dnovo*, onde os times *i* e *j* não tenham jogo nesse dia (linhas 5, 6 e 7) e pela verificação de se o deslocamento do jogo (*i, j*) para o dia *dnovo* gera melhoria na solução corrente (procedimento `Houvemelhoria` na linha 11). Se o deslocamento do jogo (*i, j*) para *dnovo* não gerar melhoria na solução corrente, o teste continua para o próximo dia, até que todos os dias tenham sido testados sem gerar melhoria na solução, sendo interrompido casa haja melhoria. É importante ressaltar que o procedimento `proximodia` na linha 13 retorna o dia seguinte ao dia passado como parâmetro (*dnovo*), mas quando *dnovo* alcança o último dia, o primeiro dia é retornado. Caso haja melhoria com o deslocamento do jogo (*i, j*) para *dnovo*, as estruturas de dados principal e auxiliar são atualizadas (linhas 14 a 20).

O próximo jogo a ser testado é determinado pelo procedimento `proximojogo` (linha 22), cuja função é retornar o próximo jogo (variável *jogo*). Quando há melhoria no deslocamento de um jogo, a variável *jogo_inicial* guarda esse jogo (linhas 23, 24 e 25) para que o laço de busca só termine quando todos os jogos tiverem sido testados para todos os dias sem haver melhoria na solução corrente em nenhum desses deslocamentos (linha 26), o que garante que um ótimo local foi encontrado pelo procedimento `SegundaBuscaLocal`.

```
procedimento SegundaBuscaLocal(Z, Zaux, x, y, w);
1.    Escolher jogo(i, j) aleatoriamente
2.    jogo_inicial ← jogo
3.    faça
4.       aprimorou ← falso
5.       faça
6.          Escolher dnovo ∈ D aleatoriamente
7.       enquanto (Zaux[i][dnovo]<>NULO) ou (Zaux[j][dnovo]<>NULO)
8.       dia_inicial ← dnovo
9.       faça
10.         d ← Z[i][j]
11.         aprimorou ← Houvemelhoria(Z,Zaux,x,y,w,i,j,d,dnovo)
12.         se  (não aprimorou)
13.            dnovo ← proximodia(dnovo)
14.         senão
15.            Z[i][j] ← dnovo
16.            Zaux[i][d] ← NULO
17.            Zaux[j][d] ← NULO
18.            Zaux[i][dnovo] ← j
19.            Zaux[j][dnovo] ← -i
20.         fim-se
21.      enquanto (não aprimorou) e (dnovo<>dia_inicial)
22.      jogo ← proximojogo(jogo)
23.      se (aprimorou)
24.         jogo_inicial ← jogo
25.      fim-se
26.   enquanto (jogo <> jogo_inicial)
fim - SegundaBuscaLocal
```

Figura 17: Pseudo-código da segunda busca local.

3.6 Critério de aceitação

A solução *s'* obtida pela segunda busca local (passo 5 da Figura 5) é aceita ou não como nova solução corrente *s** dependendo do critério de aceitação utilizado pelo algoritmo, conforme mostra o passo 6 do procedimento ILS da Figura 5 (Seção 3.1). O critério de aceitação é um procedimento que visa comparar a solução obtida pela segunda busca local com a solução corrente do algoritmo, atualizando a solução corrente caso a primeira seja

melhor ou equivalente à segunda em termos de comparação lexicográfica. Com isso, na iteração seguinte, um novo procedimento de perturbação e busca local será executado sobre a nova solução corrente. Caso a solução obtida pela segunda busca local não seja melhor ou tão boa quanto a solução corrente, a solução corrente se mantém inalterada e na iteração seguinte uma nova perturbação e busca local serão executados sobre a mesma solução corrente.

O critério de aceitação se baseia no número de violações das restrições fortes, no número de violações das restrições fracas e na soma das distâncias percorridas pelos times na solução. Uma solução *s'* é aceita como nova solução corrente *s** se uma das condições abaixo for verdadeira:

1. O número de violações das restrições fortes de *s'* é menor que o número de violações das restrições fortes de *s**.
2. O número de violações das restrições fortes de *s'* é igual ao número de violações das restrições fortes de *s** e o número de violações das restrições fracas de *s'* é menor que o número de violações das restrições fracas de *s**.
3. O número de violações das restrições fortes de *s'* é igual ao número de violações das restrições fortes de *s**, o número de violações das restrições fracas de *s'* é igual ao número de violações das restrições fracas de *s** e a distância total de *s'* é menor ou igual à distância total de *s**.

Considera-se que, nesses casos, a solução encontrada pela segunda busca local é melhor, ou tão boa quanto a solução corrente, o que faz com que a solução corrente seja atualizada. Entretanto, na terceira condição acima, considera-se que houve melhoria somente quando a distância total de *s'* for menor do que a distância total de *s**.

Para controlar a estrutura de perturbação utilizada na iteração seguinte do procedimento `ILS`, em caso de melhoria, após o procedimento do critério de aceitação a variável *hist* é reinicializada com o valor zero (*hist* ← 0), fazendo com que a primeira estrutura de vizinhança seja utilizada, isto é, movimentos de deslocamento com troca de dias. Caso não haja melhoria (solução pior ou equivalente à solução corrente), o valor de *hist* é incrementado (*hist* ← *hist* + 1), fazendo com que, na iteração seguinte, o procedimento de perturbação verifique a estrutura de vizinhança a ser utilizada conforme os parâmetros β, α e Δ.

É importante salientar, conforme descrito na terceira condição acima, que soluções equivalentes, ou seja, com o mesmo número de violações de restrições fortes e fracas e com a mesma distância total são aceitas como nova solução corrente. Essa decisão permite que o algoritmo utilize soluções diferentes, percorrendo de forma mais completa o espaço de busca.

3.7 Oscilação estratégica

Outra estratégia adotada para explorar melhor o espaço de busca foi a inclusão de uma fase de oscilação estratégica [9]. De acordo com Anagnostopoulos et al. [1], a oscilação estratégica (também conhecida como relaxação adaptativa) é normalmente utilizada em algoritmos avançados de busca tabu a fim de explorar soluções viáveis e inviáveis. No problema do Campeonato Nacional Masculino de Basquete de 2004, quando uma solução viável é encontrada, os jogos do primeiro e do último dia não serão modificados, assim como os jogos do turno não passarão para o returno e vice-versa, uma vez que isso violaria as restrições 2, 8 ou 9, limitando, assim, a exploração do espaço de busca. Para contornar esse problema, é permitido que soluções inviáveis sejam exploradas, modificando, por exemplo, os jogos desses dias (primeiro e último).

A abordagem de oscilação estratégica foi empregada através da modificação na avaliação de soluções vizinhas e no critério de aceitação, onde uma penalidade de δ quilômetros é aplicada a uma solução para cada violação de uma restrição fraca. Com isso, a avaliação de uma solução vizinha é feita considerando-se primeiro o número de violações das restrições fortes. Caso o número de violações das restrições fortes da solução vizinha seja menor ou igual ao número de violações fortes da solução corrente, compara-se o valor *distância total* + *número de violações das restrições fracas* x δ x ω, para as duas soluções, onde δ representa o quantidade de quilômetros da penalidade e ω representa o peso da penalidade (os valores de δ e ω são descritos no Capítulo 4). Assim, uma solução vizinha é aceita se:

1. O número de violações das restrições fortes da solução vizinha for menor do que o número de violações das restrições fortes da solução corrente.
2. O número de violações das restrições fortes da solução vizinha for igual ao número de violações das restrições fortes da solução corrente e o valor dado pela fórmula para a solução vizinha for menor ou igual ao valor dado pela fórmula para a solução corrente.

Com a oscilação estratégica, o procedimento `Houveamelhoria` descrito na Seção 3.3.1, foi adaptado para considerar as condições acima. Sendo assim, logo após o recálculo do número de violações das restrições fracas da solução vizinha, é efetuado o recálculo da

distância total da solução vizinha. Para cada restrição fraca violada, acrescenta-se à distância total da solução δ quilômetros multiplicados por uma penalidade ω.

O peso dessa penalidade varia no decorrer dessa fase, começando com o valor um, passando a dois e, por fim, a quatro. O que determina a mudança no peso da penalidade é o número de iterações do procedimento `ILS` (ver Figura 5 da Seção 3.1) sem melhoria na solução corrente. Sendo assim, primeiro é utilizado o procedimento `Houveamelhoria` normal (descrito na Seção 3.3.1), uma vez que com essas penalidades, soluções inviáveis podem ser geradas. Depois que um certo número de iterações no procedimento `ILS` sem melhoria na solução corrente é alcançado, o algoritmo entra na fase de oscilação estratégica. Essa fase começa com a aplicação de peso unitário até que o número de iterações sem melhoria seja igual ao parâmetro θ, o que faz com que seja aplicado o valor dois que, por sua vez, permanece até que seja aplicado o valor quatro. Porém, depois que o valor quatro é aplicado, volta-se a utilizar o procedimento `Houveamelhoria` normal para que novas soluções viáveis sejam produzidas com a distância já minimizada. Durante a oscilação estratégica a tendência é que a distância seja minimizada em prejuízo de violações das restrições fracas.

Capítulo 4

Experimentos computacionais

O algoritmo proposto foi implementado na linguagem de programação C, utilizando a ferramenta de desenvolvimento Microsoft Visual C++ (aplicação console). Os experimentos foram executados em uma máquina Intel Pentium IV com relógio de 3.0 GHz e 1 Gbyte de memória RAM, sob o sistema operacional Windows XP. O algoritmo foi executado compartilhando-se o processamento da máquina com aplicações do sistema.

As distâncias entre as cidades da Seção 2.4 foram obtidas em [28] e referem-se, na sua maioria, a distâncias de centro a centro entre duas cidades pelos caminhos mais curtos, dando preferência às rodovias asfaltadas, sobretudo federais.

Cada execução pode produzir uma solução diferente (pois o gerador de números aleatórios utilizado usa sementes diferentes a cada execução).

Durante o desenvolvimento do algoritmo diversas estratégias e conjunto de parâmetros foram testados visando obter soluções de boa qualidade em relação à tabela utilizada pela CBB. Com intuito de analisar o desempenho de cada parte do algoritmo proposto, experimentos computacionais foram realizados e verificou-se o comportamento de cada parte do algoritmo descrito nas seções anteriores. As seções seguintes relatam algumas estratégias e ajustes de parâmetros realizados em cada parte do algoritmo.

4.1 Experimentos com a busca local inicial

Na busca local inicial tem-se como parâmetro *maxit*, que corresponde ao número máximo de iterações dessa busca sem gerar melhoria na solução corrente. Além disso, é necessário estabelecer o tempo máximo de execução do algoritmo proposto descrito anteriormente, que foi de 300 segundos.

No primeiro experimento, definiu-se o parâmetro *maxit* = 10.000 (10.000 iterações sem melhoria na solução corrente) e tempo máximo de execução de 300 segundos. A Tabela 7

apresenta os resultados obtidos pelo procedimento `BuscaLocalInicial` descrito na Seção 3.3 (Figura 10) em dez execuções do algoritmo proposto. A primeira coluna corresponde a cada execução do algoritmo. As três colunas seguintes relatam as características da solução inicial gerada pelo procedimento `CriaSolucaoInicial` descrito na Seção 3.2.2: a distância total da solução, o número de violações das restrições fortes e o número de violações das restrições fracas. A quinta, sexta e sétima coluna apresentam as características da solução obtida após a execução do procedimento `BuscaLocalInicial`: a distância total da solução, o número de violações das restrições fortes e o número de violações das restrições fracas. Na oitava coluna apresenta-se o tempo em segundos gasto pelo procedimento `BuscaLocalInicial`. Finalmente, a nona coluna mostra a redução percentual da distância total obtida com a execução da busca local inicial.

Conforme descrito nas Seções 3.2.1 e 3.2.2, observa-se que as soluções iniciais geradas não violam nenhuma restrição fraca, mas violam 39,8 restrições fortes na média. O valor médio da distância total dessas soluções é de 302.325,6 km. O tempo médio de execução da busca local inicial foi de 1,2 segundos, obtendo, na média, soluções com 0,5 violações das restrições fortes e 0,3 violações de restrições fracas. Obteve-se uma redução média de 34,43% da distância total da solução.

	Solução inicial			Solução após a busca local inicial				
Execução	Distância	Violações fortes	Violações fracas	Distância	Violações fortes	Violações fracas	Tempo (seg.)	Redução (%)
1	307.211,0	14	0	187.531,0	0	0	1,3	38,96
2	300.463,0	51	0	195.553,0	1	1	1,2	34,92
3	296.601,0	49	0	207.505,0	0	0	1,0	30,04
4	305.557,0	47	0	205.258,0	0	1	1,1	32,82
5	311.047,0	51	0	198.642,0	1	0	1,3	36,14
6	285.547,0	30	0	188.628,0	0	0	1,3	33,94
7	308.999,0	51	0	206.634,0	1	0	1,2	33,13
8	291.646,0	46	0	199.036,0	1	1	1,1	31,75
9	299.187,0	27	0	195.490,0	0	0	1,2	34,66
10	316.998,0	32	0	196.715,0	1	0	1,3	37,94
Média	302.325,6	39,8	0	198.099,2	0,5	0,3	1,2	34,43

Tabela 7: Resultados da busca local inicial com *maxit* = 10.000.

Ao se ajustar o parâmetro *maxit* para 50.000, obtêm-se os resultados apresentados na Tabela 8, cujas colunas são as mesmas da Tabela 7. As soluções iniciais utilizadas foram as mesmas do experimento anterior (com *maxit* = 10.000). Apenas 70% das soluções obtidas

pela busca local inicial são viáveis. Observa-se também que a redução média da distância total das soluções obtidas é de 36,37%.

	Solução inicial			Solução após a busca local inicial				
Execução	Distancia	Violações fortes	Violações fracas	Distância	Violações fortes	Violações fracas	Tempo (seg.)	Redução (%)
1	307.211,0	14	0	184.755,0	0	0	4,4	39,86
2	300.463,0	51	0	184.727,0	0	0	5,2	38,52
3	296.601,0	49	0	192.195,0	0	0	3,3	35,20
4	305.557,0	47	0	202.821,0	0	1	2,5	33,62
5	311.047,0	51	0	193.039,0	0	0	3,2	37,94
6	285.547,0	30	0	183.551,0	0	0	4,2	35,72
7	308.999,0	51	0	201.337,0	1	0	2,5	34,84
8	291.646,0	46	0	195.612,0	0	0	1,3	32,93
9	299.187,0	27	0	193.752,0	0	0	2,4	35,24
10	316.998,0	32	0	190.798,0	1	0	3,4	39,81
Média	302.325,6	39,8	0	192.258,7	0,2	0,1	3,2	36,37

Tabela 8: Resultados da busca local inicial com *maxit* = 50.000.

Mais uma vez, ao se ajustar o parâmetro *maxit* para 100.000, obtêm-se os resultados apresentados na Tabela 9, cujas colunas são as mesmas das Tabela 7 e 8. As soluções iniciais dos dois experimentos anteriores foram mantidas (com *maxit* = 10.000 e *maxit* = 50.000). Observa-se que, embora a redução média da distância não tenha sido tão significativa em relação ao experimento anterior (redução adicional de 0,92%), todas as soluções obtidas pela busca local inicial são viáveis. Isto ocorre porque, durante a busca local, movimentos que reduzem o número de violações das restrições fortes e fracas geram soluções melhores (em termos de comparação lexicográfica) do que as soluções geradas por movimentos que reduzem apenas a distância em detrimento das violações das restrições fortes e fracas.

	Solução inicial			Solução após a busca local inicial				
Execução	Distancia	Violações fortes	Violações fracas	Distância	Violações fortes	Violações fracas	Tempo (seg.)	Redução (%)
1	307.211,0	14	0	184.231,0	0,0	0	6,6	40,03
2	300.463,0	51	0	183.303,0	0,0	0	7,2	38,99
3	296.601,0	49	0	192.195,0	0,0	0	4,5	35,20
4	305.557,0	47	0	191.052,0	0,0	0	8,1	37,47
5	311.047,0	51	0	193.039,0	0,0	0	3,8	37,94
6	285.547,0	30	0	182.838,0	0,0	0	6,2	35,97
7	308.999,0	51	0	200.019,0	0,0	0	5,6	35,27
8	291.646,0	46	0	189.081,0	0,0	0	10,0	35,17
9	299.187,0	27	0	189.775,0	0,0	0	7,0	36,57
10	316.998,0	32	0	189.179,0	0,0	0	7,2	40,32
Média	302.325,6	39,8	0	189.471,2	0,0	0	6,6	37,29

Tabela 9: Resultados da busca local inicial com *maxit* = 100.000.

Um resumo dos resultados obtidos pela busca local inicial é apresentado nos gráficos das Figuras 18 e 19. Cada ponto no gráfico da Figura 18 representa a média do número de violações das restrições fortes e fracas em relação ao tempo médio de execução. Quanto maior o valor de *maxit*, mais iterações são realizadas pelo procedimento BuscaLocalInicial, o que aumenta o tempo de execução desse procedimento, mas como pode ser observado pelo gráfico da Figura 18, melhora-se a qualidade das soluções, tanto em relação às restrições fortes quanto em relação às restrições fracas. Pode-se observar que, com um tempo de execução médio de 6,6 segundos, a busca local inicial obteve soluções viáveis. O mesmo comportamento pode ser observado em relação à distância, como mostra a Figura 19, onde a distância total da solução mostra-se inversamente proporcional ao tempo de execução da busca local inicial, pois quanto maior o tempo de execução, menor a distância total.

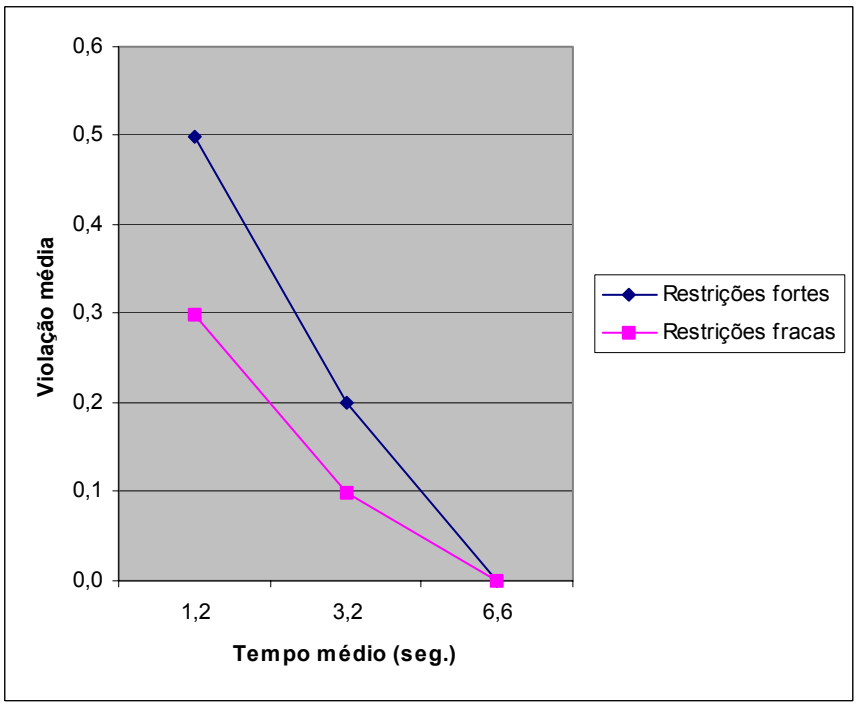

Figura 18: Tempo da busca local inicial e o número de restrições violadas.

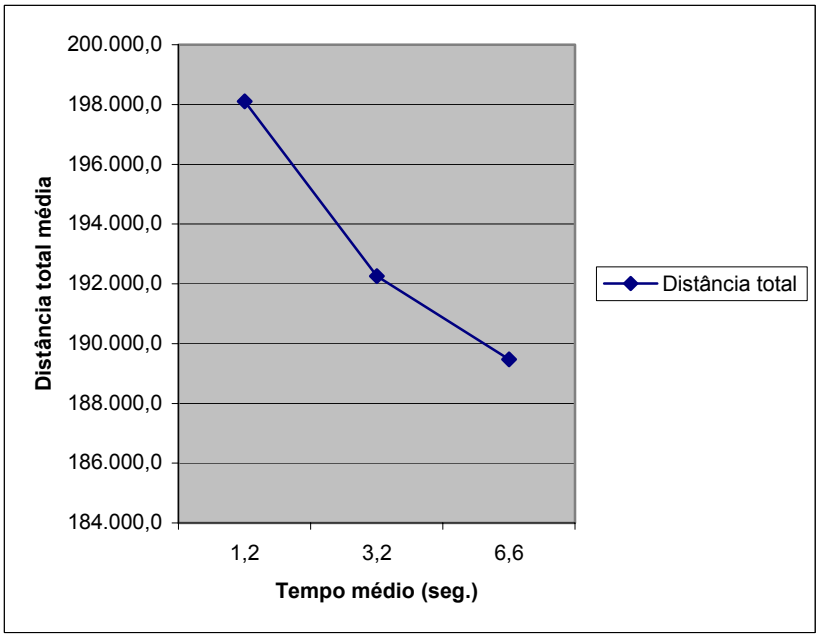

Figura 19: Tempo da busca local inicial e a distância total.

4.2 Experimentos com a perturbação

A fim de avaliar o comportamento de cada estrutura de vizinhança utilizada pelo procedimento de perturbação descrito na Seção 3.4 (`PerturbaSolucao`), foram realizados experimentos com cada estrutura de vizinhança isoladamente. Nesse experimento, o algoritmo proposto foi executado cinco vezes com o conjunto de parâmetros descrito a seguir. A condição de parada foi de 300 segundos. Na busca local inicial, o valor de *maxit* foi de 100.000. Não foram utilizados os parâmetros α, β e Δ, pois cada estrutura de vizinhança foi testada isoladamente. O parâmetro γ corresponde à quantidade de movimentos de deslocamento com troca de dias. Para a vizinhança deslocamento com troca de dias, utilizou-se $\gamma = 3$.

A Tabela 10 exibe os resultados para esse conjunto de parâmetros. Foram realizadas cinco execuções completas do algoritmo. Em cada execução foram testadas diferentes estruturas de vizinhança isoladamente. O parâmetro fornecido ao gerador de números aleatórios foi fixado a cada execução, o que fez com que fosse obtida a mesma solução inicial e a mesma solução após a busca local inicial. Nas cinco execuções, definiu-se *maxit* = 10.000 e a condição de parada foi o tempo total de 300 segundos. A busca local inicial obteve soluções viáveis. Por essa razão, omitiu-se o número de violações das restrições fortes e fracas das soluções nessa tabela, apresentando-se apenas a distância total em cada execução.

A primeira coluna mostra cada execução realizada. Para cada execução, a segunda coluna separa os valores da distância obtidos após a busca local inicial e ao término do algoritmo. Na terceira coluna apresenta-se, para cada execução, os valores da distância total quando a estrutura de vizinhança utilizada é deslocamento com troca de dias. A quarta coluna apresenta os mesmos valores que a terceira coluna, mas nesse caso a estrutura de vizinhança utilizada foi troca de times. A quinta coluna apresenta os mesmos valores que a terceira e quarta colunas, mas para a estrutura de vizinhança troca de semanas.

Observa-se que, na média, as melhores soluções (maiores reduções percentuais) foram obtidas quando a estrutura utilizada é deslocamento com troca de dias. Isto ocorre porque essa estrutura mantém o princípio de iniciar a perturbação produzindo soluções mais próximas da solução corrente, gerando soluções mais distantes da solução corrente somente em fases posteriores da perturbação, ou seja, na medida em que não se obtém melhoria na solução corrente.

Execução		Desloc.+ troca dias, Distância	Troca de times Distância	Troca semanas Distância
1	Após busca local inicial	188.302,0	188.302,0	188.302,0
	Solução final	174.293,0	172.547,0	175.670,0
	Redução %	7,44	8,37	6,71
2	Após busca local inicial	192.257,0	192.257,0	192.257,0
	Solução final	174.104,0	176891,0	176.267,0
	Redução %	9,44	7,99	8,32
3	Após busca local inicial	190.186,0	190.186,0	190.186,0
	Solução final	173.110,0	181.669,0	174.541,0
	Redução %	8,98	4,48	8,23
4	Após busca local inicial	180.807,0	180.807,0	180.807,0
	Solução final	169.841,0	175.244,0	172.856,0
	Redução %	6,07	3,08	4,40
5	Após busca local inicial	184.791,0	184.791,0	184.791,0
	Solução final	170.379,0	169.025,0	171.224,0
	Redução %	7,80	8,53	7,34
Média		7,94	6,49	7,00

Tabela 10: Diferentes estruturas de vizinhança na perturbação.

Para verificar a melhor ordem de utilização das estruturas de vizinhança, foram realizados experimentos com as três estruturas de vizinhanças aplicadas em ordens diferentes.

Nesses experimentos, o algoritmo proposto foi executado cinco vezes com o mesmo conjunto de parâmetros do experimento anterior e a mesma semente inicial fornecida ao gerador de números aleatórios, o que fez com que fosse obtida a mesma solução inicial e a mesma solução após a busca local inicial. No procedimento de perturbação, para definir os limites superiores para utilização de cada estrutura de vizinhança, utilizam-se os parâmetros α, β e Δ, de tal forma que $\beta < \alpha < \Delta$. Assim sendo, a primeira estrutura será utilizada quando $0 \leq hist < \beta$, a segunda quando $\beta \leq hist < \alpha$ e a terceira quando $\alpha \leq hist < \Delta$. Utilizou-se $\gamma = 3$, $\alpha = 15$, $\beta = 18$ e $\Delta = 21$.

A Tabela 11 exibe os resultados para esse conjunto de parâmetros. A primeira coluna mostra cada execução realizada. Para cada execução, a segunda coluna separa os valores da distância obtidos após a busca local inicial e ao término do algoritmo. Na terceira coluna apresenta-se, para cada execução, os valores da distância total quando a ordem de precedência das estruturas de vizinhança utilizadas é deslocamento com troca de dias, troca de times e troca de semanas. A quarta coluna apresenta os mesmos valores que a terceira coluna, mas nesse caso a ordem de precedência utilizada foi troca de semanas, deslocamento com troca de dias e troca de times. A quinta coluna apresenta os mesmos valores que a terceira e quarta colunas, mas para a ordem de precedência composta por troca de times, deslocamento com troca de dias e troca de semanas. A sexta coluna também apresenta a distância total das soluções em cada uma das cinco execuções do algoritmo, mas nesse caso a ordem de precedência das estruturas de vizinhança foi troca de times, troca de semanas e deslocamento com troca de dias. Já a sétima coluna apresenta os mesmos valores que as colunas anteriores, considerando a troca de semana, seguida da troca de times e, por último, o deslocamento com troca de dias. Finalmente, a oitava coluna exibe a distância total de cada execução para a ordem composta por deslocamento com troca de dias, troca de semanas e troca de times.

Observa-se que, na média, as melhores soluções (maiores reduções percentuais) foram obtidas com as seguintes ordens de precedência:

- Deslocamento com troca de dias, seguido de troca times e troca de semanas (redução média de 8,51%).
- Deslocamento com troca de dias, seguido de troca de semanas e troca times (redução média de 8,52%).
- Troca de times, seguido de troca de semanas e deslocamento com troca de dias (redução média de 8,46%).

Nesses casos, as reduções percentuais na distância total da solução foram muito próximas, com diferença de apenas 0,06 % entre a melhor e a pior das três.

Execução		Desloc.+ troca dias, troca times, troca semanas Distância	Troca semanas, desloc. + troca dias, troca times Distância	Troca times, desloc. + troca dias, troca semanas Distância	Troca times, troca semanas, desloc. + troca dias Distância	Troca semanas, troca times, desloc. + troca dias Distância	Desloc. + troca dias, troca semanas, troca times Distância
1	Após busca local inicial	188.302,0	188.302,0	188.302,0	188.302,0	188.302,0	188.302,0
	Solução final	174.830,0	173.885,0	171.665,0	173.269,0	175.895,0	176.121,0
	Redução %	7,15	7,66	8,84	7,98	6,59	6,47
2	Após busca local inicial	192.257,0	192.257,0	192.257,0	192.257,0	192.257,0	192.257,0
	Solução final	172.025,0	171.560,0	175.570,0	170.701,0	174.433,0	172.423,0
	Redução %	10,52	10,77	8,68	11,21	9,27	10,32
3	Após busca local inicial	190.186,0	190.186,0	190.186,0	190.186,0	190.186,0	190.186,0
	Solução final	171.615,0	173.413,0	177.055,0	176.808,0	174.159,0	172.203,0
	Redução %	9,76	8,82	6,90	7,03	8,43	9,46
4	Após busca local inicial	180.807,0	180.807,0	180.807,0	180.807,0	180.807,0	180.807,0
	Solução final	168.218,0	167.999,0	170.711,0	169.392,0	168.093,0	165.128,0
	Redução %	6,96	7,08	5,58	6,31	7,03	8,67
5	Após busca local inicial	184.791,0	184.791,0	184.791,0	184.791,0	184.791,0	184.791,0
	Solução final	169.754,0	174.360,0	166.745,0	166.745,0	172.307,0	170.600,0
	Redução %	8,14	5,64	9,77	9,77	6,76	7,68
Média		8,51	7,99	7,95	8,46	7,61	8,52

Tabela 11: Diferentes ordens de precedência das estruturas de vizinhança na perturbação.

No entanto, realizando-se novos experimentos com essas três ordens de precedência, mas com tempo limite de execução de 900 segundos, obtêm-se resultados que indicam que a maior redução da distância total é obtida quando a ordem de precedência dessas estruturas é deslocamento com troca de dias, seguido de troca de times e troca de semanas, como mostra a Tabela 12.

Execução		Desloc.+ troca dias, troca times, troca semanas Distância	Desloc. + troca dias, troca semanas, troca times Distância	Troca times, troca semanas, desloc. + troca dias Distância
1	Após busca local inicial Solução final Redução %	188.302,0 168.626,0 10,45	188.302,0 168.379,0 10,58	188.302,0 167.759,0 10,91
2	Após busca local inicial Solução final Redução %	192.257,0 168.943,0 12,13	192.257,0 167.457,0 12,90	192.257,0 163.774,0 14,82
3	Após busca local inicial Solução final Redução %	191.997,0 167.526,0 12,75	191.997,0 177.339,0 7,63	191.997,0 170.553,0 11,17
4	Após busca local inicial Solução final Redução %	189.772,0 166.045,0 12,50	189.772,0 169.413,0 10,73	189.772,0 169.605,0 10,63
5	Após busca local inicial Solução final Redução %	184.516,0 164.613,0 10,79	184.516,0 165.078,0 10,53	184.516,0 166.703,0 9,65
Média		11,72	10,48	11,43

Tabela 12: Melhores ordens de precedência das estruturas de vizinhança na perturbação.

Quanto ao ajuste dos parâmetros γ, α, β e Δ, foram realizados experimentos utilizando-se cinco diferentes conjuntos. No primeiro conjunto tem-se $\gamma = 3$, $\alpha = 15$, $\beta = 18$ e $\Delta = 21$. No segundo, utilizou-se $\gamma = 4$, $\alpha = 20$, $\beta = 25$ e $\Delta = 30$. No terceiro, foram definidos $\gamma = 5$, $\alpha = 25$, $\beta = 30$ e $\Delta = 40$. No quarto conjunto, utilizou-se $\gamma = 10$, $\alpha = 50$, $\beta = 100$ e $\Delta = 150$. No último conjunto, tem-se $\gamma = 20$, $\alpha = 100$, $\beta = 200$ e $\Delta = 300$. Como a ordem de precedência que apresentou os melhores resultados foi deslocamento com troca de dias, seguido de troca de times e por último a troca de semanas, esta foi a seqüência utilizada. A condição de parada se manteve em 300 segundos e utilizou-se *maxit* =100.000. A Tabela 13 apresenta os resultados obtidos ao final de dez execuções do algoritmo proposto com esses diferentes conjuntos de parâmetros. Em todas as execuções, todas as soluções obtidas depois da busca local inicial e ao final do algoritmo foram viáveis. Por essa razão, é exibida apenas a distância total média das soluções. A primeira coluna apresenta os parâmetros utilizados. A segunda coluna apresenta a distância total média das soluções obtidas após a busca local inicial (solução a partir da qual será efetuada a perturbação). A terceira coluna exibe a distância total média obtida ao final da execução do algoritmo. A quarta coluna apresenta a redução percentual

obtida entre a distância média após a busca local inicial (segunda coluna) e a distância média obtida no final da execução para cada conjunto de parâmetros (primeira coluna).

Observa-se que, na medida em que os valores de γ, α, β e Δ aumentam, cresce a redução percentual obtida. A maior redução é obtida quando $\gamma = 10$, $\alpha = 50$, $\beta = 100$ e $\Delta = 150$. Quando aumentam-se os valores de γ, α, β e Δ para 20, 100, 200 e 300, respectivamente, diminui o ganho na redução percentual.

Parâmetros	Após a busca local inicial	Solução final	Redução (%)
$\gamma = 3$, $\alpha = 15$, $\beta = 18$ e $\Delta = 21$	189.112,2	172.082,0	9,01
$\gamma = 4$, $\alpha = 20$, $\beta = 25$ e $\Delta = 30$	190.151,9	172.743,5	9,15
$\gamma = 5$, $\alpha = 25$, $\beta = 30$ e $\Delta = 40$	188.409,0	170.716,7	9,39
$\gamma = 10$, $\alpha = 50$, $\beta = 100$ e $\Delta = 150$	188.444,3	168.009,9	10,84
$\gamma = 20$, $\alpha = 100$, $\beta = 200$ e $\Delta = 300$	189.324,7	169.498,1	10,47

Tabela 13: Resultados obtidos com diferentes conjuntos de parâmetros da perturbação.

Este comportamento pode ser observado no gráfico da Figura 20, em que cada barra apresenta a redução percentual da distância para cada conjunto de parâmetros. Pode-se observar que as reduções percentuais crescem com o aumento dos valores dos parâmetros até o conjunto $\gamma = 10$, $\alpha = 50$, $\beta = 100$ e $\Delta = 150$ (quarta barra no gráfico), onde é obtida a maior redução percentual.

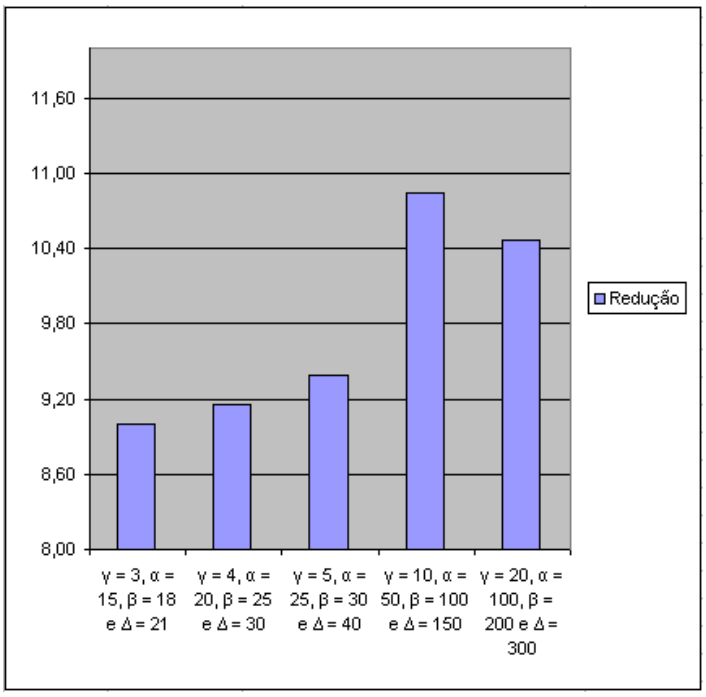

Figura 20: Comportamento da perturbação com diferentes conjuntos de parâmetros.

4.3 Experimentos com a oscilação estratégica

Durante a fase de oscilação estratégica tem-se como parâmetros δ e ω, que representam, respectivamente, a quantidade de quilômetros acrescentada à solução a cada violação das restrições fracas e o peso dessa penalidade. Foram realizados experimentos a fim de identificar a melhor configuração de tais parâmetros. Em todos os experimentos ω variou durante a fase de oscilação estratégica entre um, dois ou quatro. Nesse sentido, quando a fase de oscilação estratégica é iniciada, $\omega = 1$. Esse valor permanece até que o número de iterações sem melhoria seja igual ao parâmetro θ. Quando esse valor é alcançado, ω passa a ser igual a dois. Quando o número de iterações sem melhoria atinge novamente o valor do parâmetro θ, ω passa a ser igual a quatro, o que permanece até que o número de iterações sem melhoria atinge novamente o valor θ. Quando isso ocorre, encerra-se a fase de oscilação estratégica, não se aplicando nenhuma penalidade para cada violação das restrições fracas.

No primeiro experimento utilizou-se $\theta = 5$. No procedimento de perturbação utilizou-se $\gamma = 10$, $\alpha = 50$, $\beta = 100$ e $\Delta = 150$, tendo sido utilizada a seguinte ordem de precedência: deslocamento com troca de dias, troca de times e troca de semanas. A busca local inicial teve 100.000 iterações como limite (*maxit* = 100.000). A condição de parada foi de 300 segundos. A Tabela 14 apresenta os resultados obtidos a partir de dez execuções completas do algoritmo onde todas as soluções obtidas foram viáveis, tanto ao final do algoritmo, quanto ao final da busca local inicial. Por essa razão são omitidos os números de violações das restrições fortes e fracas. Os valores de δ testados foram 1.000, 500, 150 e 50. A primeira coluna exibe os valores de δ testados. A segunda coluna apresenta a distância média obtida ao final das dez execuções para cada valor de δ. A terceira coluna apresenta a menor distância obtida nas dez execuções do algoritmo para cada valor de δ.

Observa-se que valores muito altos de δ não produzem boas soluções na média nem no melhor caso, pois uma penalidade muito alta não permite que a distância seja minimizada em prol da violação de restrições fracas. Entretanto, penalidades baixas permitem que um movimento que viole uma ou mais restrições fracas e minimizem a distância sejam aceitos na solução corrente, o que faz com que a distância total da solução seja minimizada, mas haja violações de restrições fracas. Essas violações passam, em fases posteriores do algoritmo, a ter um peso maior, o que faz com que o algoritmo volte a priorizar as restrições fracas, isto é, movimentos que reduzem o número de violações de restrições fracas são aceitos, independente da distância.

δ	Distância final média	Melhor distância final
1000	168.510,9	164.508,0
500	166.960,0	160.527,0
150	165.132,1	158.712,0
50	168.010,7	163.115,0

Tabela 14: Experimentos com diferentes valores de penalidade.

Esse comportamento do algoritmo é ilustrado de maneira mais clara no gráfico da Figura 21, onde são apresentadas duas barras de valores para cada valor de δ utilizado. A barra da esquerda ilustra a distância média, enquanto a barra da direita mostra a menor distância. Os melhores valores foram obtidos quando $\delta = 150$.

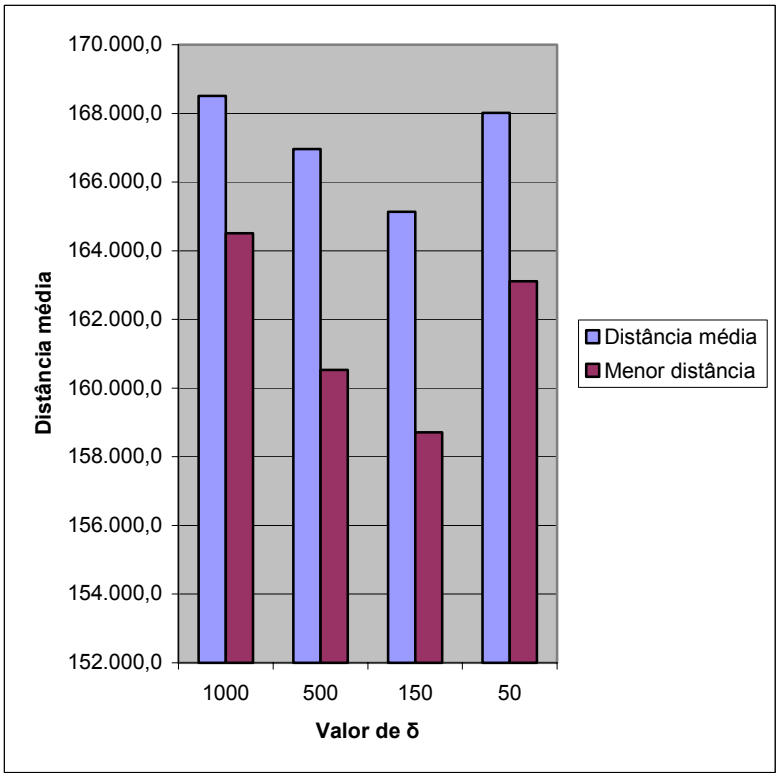

Figura 21: Experimentos com diferentes valores de penalidade.

4.4 Resultados finais

Para melhor compreensão da eficácia do algoritmo proposto, nesta seção são apresentados resultados obtidos com diferentes tempos de limite de execução do algoritmo com os melhores conjuntos de parâmetros apresentados nas seções anteriores deste capítulo. Sendo assim, os experimentos foram realizados com *maxit* = 100.000 para que a maioria das soluções geradas antes de cada iteração do algoritmo fosse viável. Além disso, foram definidos no procedimento de perturbação $\gamma = 10$, $\alpha = 50$, $\beta = 100$ e $\Delta = 150$, com a ordem de precedência das estruturas de vizinhança definida por deslocamento com troca de dias, troca de times e troca de semanas. Utilizou-se $\delta = 150$ como penalidade durante a fase de oscilação estratégica.

A Tabela 15 apresenta os resultados das soluções obtidas utilizando-se 300 segundos como tempo limite de execução do algoritmo. O algoritmo foi executado dez vezes, tendo sido obtidas soluções viáveis em todos os casos. A segunda coluna apresenta a distância total obtida ao final da execução do algoritmo, sendo a média apresentada na segunda linha, a pior solução (maior distância total) na terceira linha, a melhor solução (menor distância total) na quarta linha e a distância total da tabela original da CBB na última linha. A terceira coluna apresenta a redução percentual da distância total obtida pelo algoritmo comparando-se com a distância total da tabela divulgada pela CBB no início da competição, onde observa-se uma redução de 23,7% na média. A melhor solução obtida representa 72,8% da distância total percorrida pelos times na tabela da CBB (redução de 27,2%). Ainda deve se considerar que todas soluções geradas pelo algoritmo proposto são viáveis, o que não ocorre com a tabela da CBB, onde há uma restrição fraca violada, já que falta um jogo no último dia (violando a restrição 9).

É importante salientar que a solução da CBB que está sendo considerada é a tabela original que foi divulgada pela CBB no início da competição, ou seja, a tabela que foi criada pelos seus organizadores a fim de cumprir as restrições estabelecidas e reduzir as distâncias percorridas pelos times participantes. Durante a competição, a tabela original foi alterada, não cumprindo uma série de restrições, o que tornaria essa comparação pouco relevante.

	Distância final	Redução (%)
Média	165.649,6	23,7
Pior	171.515,0	21,0
Melhor	158.097,0	27,2
CBB	217.171,0	

Tabela 15: Resultados obtidos com tempo limite de 300 segundos.

Em outro experimento foram mantidos os valores dos parâmetros do experimento anterior, exceto o tempo limite de execução do algoritmo que passou a ser de 600 segundos. Mais uma vez executou-se o algoritmo dez vezes. A Tabela 16 mostra os resultados obtidos com essa modificação no critério de parada.

Todas as soluções obtidas também são viáveis. Em relação à distância total, pode-se observar que os resultados são ainda melhores, pois a redução média obtida foi de 25,4 %. Na melhor solução obtida também houve ganho em relação ao experimento anterior (com tempo limite de 300 segundos), pois chega-se a uma redução de 28,4%.

	Distância final	Redução (%)
Média	162.088,7	25,4
Pior	167.633,0	22,8
Melhor	155.470,0	28,4
CBB	217.171,0	

Tabela 16: Resultados obtidos com tempo limite de 600 segundos.

Em mais dois experimentos com o mesmo conjunto de parâmetros dos experimentos anteriores, modificou-se o tempo limite de execução para 900 e 1200 segundos, conforme relatado nas Tabelas 17 e 18, respectivamente. Os resultados obtidos são ainda melhores do que nos experimentos anteriores, pois apesar de não haver diferença significativa entre os resultados obtidos com 600 e 900 segundos, com 1200 segundos obtêm-se uma redução na distância de 26,8% na média. Na melhor solução encontrada a redução chega a 28,8%, o que corresponde a 154.598 km.

	Distância final	Redução (%)
Média	162.033,8	25,5
Pior	170.457,0	23,3
Melhor	156.398,0	28,5
CBB	217.171,0	

Tabela 17: Resultados obtidos com tempo limite de 900 segundos.

	Distância final	Redução (%)
Média	159.012,6	26,8
Pior	163.654,0	24,6
Melhor	154.598,0	28,8
CBB	217.171,0	

Tabela 18: Resultados obtidos com tempo limite de 1200 segundos.

O gráfico da Figura 22 apresenta as reduções obtidas nos experimentos com tempo limite de 300, 600, 900 e 1200 segundos. São apresentadas seqüências de valores para a redução média, para a maior redução e para a pior redução. Pode-se verificar que quando o tempo limite muda de 300 para 600 segundos, há uma melhoria significativa na redução média, assim como na melhor e na pior redução. Na modificação de 600 para 900 segundos não há melhorias significativas na redução média e na melhor redução. Nesse caso, a melhoria mais significativa ocorre na pior redução, onde a redução passa de 22,8% para 23,3%. Entretanto, quando altera-se o tempo limite de 900 para 1200 segundos, pode-se observar que a redução média passa de 25,5% para 26,8%. Quanto à maior redução, não se observam mudanças significativas que passa de 28,5% para 28,8%.

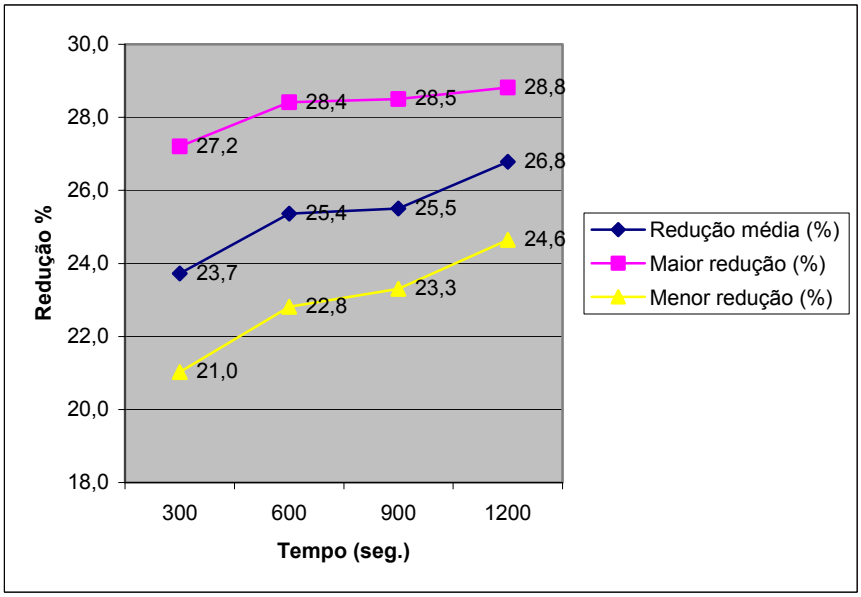

Figura 22: Gráfico com as reduções obtidas para diferentes tempos limite.

Para verificar a contribuição da fase de oscilação estratégica no resultado final do algoritmo, realizaram-se experimentos com e sem a aplicação dessa fase. Os resultados obtidos por esse experimento são relatados na Tabela 19, onde as três últimas colunas apresentam os resultados obtidos sem a oscilação estratégica. Mais uma vez, o algoritmo foi

executado dez vezes com os mesmos parâmetros do experimento anterior. Nota-se que as soluções geradas pelo algoritmo com a oscilação estratégica são melhores do que as soluções geradas pelo mesmo algoritmo sem uso da oscilação estratégica, principalmente considerando-se a redução média da distância. Esta diferença torna-se mais significativa quando crescem os tempos limite de execução do algoritmo, pois quanto maior a duração do algoritmo, melhor é explorada a oscilação estratégica. Quando o algoritmo executa durante 900 segundos, por exemplo, a diferença percentual média é de 3,5%. Quando modifica-se a duração de execução para 1200 segundos, a diferença percentual média passa a ser de 3,8%.

	Com oscilação estratégica			Sem oscilação estratégica		
Tempo (seg.)	Redução média (%)	Maior redução (%)	Menor redução (%)	Redução média (%)	Maior redução (%)	Menor redução (%)
300	23,7	27,2	21,0	21,3	25,0	18,5
600	25,4	28,4	22,8	22,2	25,1	19,7
900	25,5	28,5	23,3	22,0	25,0	19,8
1200	26,8	28,8	24,6	23,0	25,1	20,9

Tabela 19: Resultados com e sem oscilação estratégica.

Capítulo 5

Conclusões

O trabalho desenvolvido refere-se à programação de jogos de competições esportivas e constitui-se no desenvolvimento de um algoritmo para otimização da tabela do Campeonato Nacional Masculino de Basquete de 2004, organizado pela Confederação Brasileira de Basquete. Tal otimização visa obter uma tabela para a fase de classificação do campeonato, de tal forma que a soma das distâncias percorridas pelos times para realização dos jogos seja minimizada, satisfazendo a um conjunto de restrições.

O algoritmo desenvolvido é baseado na metaheurística *Iterated Local Search* (ILS), sendo composto por uma heurística construtiva, duas buscas locais diferentes, um procedimento de perturbação, um critério de aceitação e uma fase de oscilação estratégica. A heurística construtiva visa obter uma solução aleatória de forma que algumas restrições sejam satisfeitas. A busca local inicial é acionada uma única vez, tendo como objetivo tornar a solução viável (com todas as restrições atendidas) e reduzir a soma das distâncias percorridas pelos times. A segunda busca local é executada a cada iteração do algoritmo, tendo como objetivo obter um ótimo local a partir da solução corrente. Em ambas buscas utilizou-se o movimento *deslocamento*, que consiste em modificar o dia de um jogo. No procedimento de perturbação foram utilizadas outras estruturas de vizinhança, que se baseiam em movimentos de *troca de dias*, *troca de semanas* e *troca de times*. O critério de aceitação tem uma particularidade que consiste em aceitar soluções equivalentes, ou seja, que possuem o mesmo número de violações de restrições fortes e fracas e a mesma distância total, o que permite que o algoritmo explore melhor o espaço de busca. A fase de oscilação estratégica foi acrescentada ao algoritmo para estender a capacidade de explorar o espaço de busca e minimizar a soma das distâncias da solução, fazendo com que soluções inviáveis fossem temporariamente aceitas em prol da minimização das distâncias, voltando a tornar a solução

viável posteriormente. Comprovou-se experimentalmente que a inclusão dessa fase melhora os resultados obtidos pelo algoritmo proposto.

Na implementação do algoritmo houve a preocupação em realizar o recálculo das restrições com a menor complexidade computacional possível, fazendo com que a maioria das restrições fossem calculadas em tempo constante, o que é fundamental no desempenho de todo algoritmo de busca local, já que a avaliação de soluções vizinhas é o procedimento mais executado nesse tipo de algoritmo.

Esta dissertação mostra que, embora seja trabalhoso, é possível desenvolver heurísticas de busca local eficientes para problemas reais com muitos tipos de restrições.

Os experimentos computacionais realizados comprovaram que a tabela obtida pelo algoritmo proposto nesse trabalho é melhor do que a tabela utilizada pela CBB para o Campeonato Nacional Masculino de Basquete de 2004. Observou-se que, além de não violar restrição alguma do problema (o que não ocorre na tabela da CBB), as soluções obtidas pelo algoritmo proposto reduzem as distâncias percorridas pelos times em até 28,81%, obtendo uma redução média de 26,80% quando comparadas com a tabela da CBB. É importante ressaltar que tais resultados são obtidos em um tempo computacional reduzido, conforme os experimentos apresentados no Capítulo 4.

A programação de competições esportivas é uma tarefa árdua realizada pela maioria das federações nacionais e internacionais devido à própria complexidade do problema e à falta de métodos e recursos adequados para tal tarefa. Pode-se observar que as soluções obtidas pelo algoritmo proposto, se utilizadas pelas federações, poderiam contribuir para diminuir o esforço necessário para programação dos jogos e permitir mais tempo de descanso aos atletas, o que proporciona maior performance individual e coletiva. Além disso, poderia contribuir para diminuir os custos com passagens dos clubes para realizarem seus jogos.

Como perspectivas de trabalhos futuros, pode-se citar a inclusão de fatores financeiros como custos com passagens aéreas e terrestres para realização das viagens, bem como custos associados à hospedagem das equipes participantes. Além disso, pode-se adaptar o algoritmo desenvolvido para problemas semelhantes como a programação dos jogos da Nossa Liga de Basquete, assim como de competições de outros esportes.

Referências

1. Anagnostopoulos, A.; Michel, L.; Van Hentenryck, P.; Vergados, Y.: **A Simulated annealing approach to the traveling tournament problem.** Journal of Scheduling v. 9, p. 177-193, 2006.

2. Armstrong, J.; Willis, R. J.: **Scheduling the cricket world cup - A case study**. Journal of the Operational Research Society, v. 44, p. 1067–1072, 1993.

3. Bean, J.C.; Birge, J.R.: **Reducing travel costs and player fatigue in the NBA**. Interfaces, v. 10, p. 98-102, 1980.

4. Costa, D.: **An evolutionary tabu search algorithm and NHL scheduling problem**. INFOR, v. 33, p. 161-178, 1995.

5. De Werra, D.: **Geography, games and graphs**. Discrete Applied Mathematics, v. 2, p. 327-337, 1980.

6. Easton, K.; Nemhauser, G.; Trick, M.: **The traveling tournament problem: Description and benchmarks**. Em: Walsh, T., editor, "Principles and Pratice of Constraint Programming", Lecture Notes in Computer Science, v. 2239, p. 580-584. Springer, 2001.

7. Easton, K.; Nemhauser, G.; Trick, M.: **Sports scheduling**. Em: Leung, J., editor, "Handbook of Scheduling: Algorithms, Models, and Performance Analysis", p. 52.1-52.19. CRC Press, 2004.

8. Ferland, J.A., Fleurent, C.: **Computer aided scheduling of a sports league**. INFOR, v. 29, p. 14-24., 1991.

9. Glover, F.: **Multi-start and strategic oscillation methods - Principles to exploit adaptive memory**. Em: Laguna, M.; Gonzáles-Velarde, J.L., editores, "Computing Tools for Modeling, Optimization and Simulation", Interfaces in Computer Science and Operations Research, p. 1-24. Kluwer, 2000.

10. Hamiez, J. P.; Hao, J. K.: **Solving the sports league scheduling problem with tabu search**. Lecture Notes in Artificial Intelligence, v. 2148, p. 24-36, Springer, 2001.

11. Hansen, P.; Mladenovic, N.: **Variable Neighborhood Search**. Em: Glover, F.; Kochenberger, G., editores, "Handbook of MetaHeuristics", p. 321-353. Kluwer Academic Publishers, 2002.

12. Henz, M.: **Scheduling a major college basketball conference**. Operations Research, v. 49, p. 163-168, 2001.

13. Lajolo, M.: **Jogadores do nacional de basquete cruzam o país de ônibus.** Referência on-line em http://www1.folha.uol.com.br/folha/esporte/ult92u71683.shtml, última visita em 3 de julho de 2006.

14. Lourenço, H. R.; Martin O.; Stützle, T.: **Iterated local search**. Em: Glover, F.; Kochenberger, G., editores, "Handbook of MetaHeuristics", p. 321-353. Kluwer Academic Publishers, 2002.

15. McAloon, K; Tretkoff, C.; Wetzel, G.: **Sports league scheduling**. Proceedings of ILOG Optimization Suite International Users's Conference, 1997.

16. Miyashiro, R.; Iwasaki, H.; Matsui, T.: **Characterizing feasible pattern sets with a minimum number of breaks**. Em: Burke, E.; Carter, M., editores, "Practice and Theory of Automated Timetabling IV: Selected Papers from the 4th International Conference for the Practice and Theory of Automated Timetabling", Lecture Notes in Computer Science, v. 2740, p. 78-99. Springer, 2003.

17. Nemhauser, G. L.; Trick, M. A.: **Scheduling a major college basketball conference**. Operations Research, v. 24, p. 1097-1100, 1997.

18. Reeves C.R.: **Genetic algorithms.** Em: C.R. Reeves, editor, "Modern Heuristic Techniques for Combinatorial Problems", p. 151-196. Blackwell Scientific Publications, 1993.

19. Resende M.; Ribeiro C.: **Greedy randomized adaptive search procedures**. Em: Glover, F.; Kochenberger, G.; editores. "Handbook of MetaHeuristics", p. 219-249. Kluwer, 2002.

20. Souza, M.J.F.; Biajoli, F.L; Mine, O.M.; Chaves, A.A.: **Escala de jogos em torneios esportivos: Uma abordagem via simulated annealing**. Anais do XXXV Simpósio Brasileiro de Pesquisa Operacional, 2003, Natal, p. 1295–1306, 2003.

21. Terril, B.J., Willis, R.J.: **Scheduling the australian state cricket season using simulated annealing**. Journal of the Operational Research Society, v. 45, p. 276–280, 1994.

22. Trick, M.: **A schedule-and-break approach to sports scheduling**. Lecture Notes in Computer Science, v. 2079, p. 242-253, 2000.

23. Urrutia, S.: **Otimização em esportes: Programação de tabelas e problemas de classificação**, Tese de Doutorado, Departamento de Informática, PUC-Rio, 2005.

24. Wright, M.: **Timetabling country cricket fixtures using a form of tabu search**, Journal of the Operational Research Society, v. 45, p. 758-770, 1994.

25. Yang, J.T.; Huang, H.-D.; Yang, J.; Horng, J.-T.: **Devising a cost effective baseball scheduling by evolutionary algorithms**. Proceedings of the 2002 Congress on Evolutionary Computation, p. 1660–1665, 2002.

26. Zhang, H.: **Generating College Conference Basketball Schedules by a SAT Solver**. Proceedings of the 5th International Symposium on the Theory and Applications of Satisfiability Testing, p. 281–291, 2002.

27. **Confederação Brasileira de Basquetebol**. Referência on-line em http://www.cbb.com.br/competicoes/cnbm2004/tabela.asp?fase=10, última visita em 29 de junho de 2006.

28. **Aonde Fica**. Referência on-line em http://www.aondefica.com/, última visita em março de 2005.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo