3
Heurísticas para o Problema do Torneio com Viagens Espelhado

O problema do Torneio com Viagens (TTP) apresentado na seção anterior foi primeiramente definido em [20] e instâncias de teste estão disponíveis na página web do problema [59]. Algumas das instâncias estão baseadas na *Major League Baseball* (MLB) dos Estados Unidos, enquanto outras foram geradas com uma estrutura circular. No momento, mesmo instâncias pequenas do TTP com $n = 8$ equipes não puderam ser resolvidas de forma exata.

O Problema do Torneio com Viagens Espelhado (MTTP) não foi tratado na literatura antes desta tese. Alguns algoritmos baseados em metaheurísticas foram propostos para o TTP. Crauwels and Van Oudhuesden [10] propuseram uma heurística baseada em colônias de formigas, na qual o procedimento de construção consiste de uma estratégia de *backtracking* e um algoritmo de busca local, sem obter bons resultados. Cardemil e Durán [7, 8] desenvolveram um algoritmo de busca tabu com o qual melhoraram as soluções conhecidas para algumas instâncias de teste no momento de desenvolvimento desses trabalhos. Anagnostopoulos et al. [2] propuseram um algoritmo de *simulated annealing* que conseguiu as melhores soluções conhecidas para muitas das instâncias de teste. Os dois últimos algoritmos partem de soluções iniciais geradas aleatoriamente e os tempos de computação são da ordem de vários dias de processamento.

3.1
Uma heurística construtiva para o MTTP

Boas soluções iniciais podem melhorar o desempenho dos algoritmos baseados em metaheurísticas para problemas de programação de tabelas. Elmohamed et al. [24] comprova que quando uma configuração inicial aleatória é usada para problemas de programação de tabelas, algoritmos baseados em *simulated annealing* têm um desempenho muito fraco. Há uma

significativa melhoria no desempenho dos algoritmos quando uma etapa de preprocessamento é empregada para fornecer boas soluções de partida. Algoritmos rápidos são importantes para problemas de programação de tabelas porque os diferentes (e às vezes contrários) interesses dos decisores requerem uma certa interatividade. Muitas vezes, um dos decisores não vai se sentir satisfeito com a solução obtida e outras soluções terão que ser obtidas.

Propõe-se a seguir uma heurística construtiva de três etapas para o MTTP. Esta heurística explora o fato de que um torneio MDRR está composto por dois torneios SRR, sendo o segundo igual ao primeiro exceto pelos mandos de campo invertidos. Na primeira etapa constroi-se uma 1-fatoração ordenada de K_n que representa uma tabela de um torneio SRR entre n equipes abstratas (cada uma delas podendo ser associada a uma equipe real). Na segunda etapa, uma equipe real é associada a cada equipe abstrata correspondente a um nó de K_n, determinando-se uma tabela para um torneio entre as n equipes reais. Na última etapa, atribui-se uma sede a cada jogo, obtendo-se uma 1-fatoração ordenada e orientada de K_n.

3.1.1 Etapa 1: Construir uma 1-fatoração

O conhecido método do polígono (ver, por exemplo, [22]) é usado para construir uma 1-fatoração ordenada de K_n que corresponde a uma tabela para um torneio SRR com n equipes abstratas, sem atribuir sedes aos jogos.

As equipes abstratas $1, \ldots, n-1$ são inicialmente colocadas nos vértices de um polígono regular de $n-1$ vértices (chama-se a cada um desses vértices de nó 1, nó 2, etc.): a equipe 1 no nó 1, a equipe 2 no nó 2 e assim sucessivamente. A equipe abstrata n não é colocada no polígono. A cada rodada $k = 1, \ldots, n-1$, a equipe abstrata colocada no nó $\ell = 2, ..., n/2$ joga contra a equipe abstrata colocada no nó $n+1-\ell$. Em cada rodada, a equipe abstrata colocada no nó 1 joga contra a equipe abstrata n. Após cada rodada, cada equipe abstrata $1, \ldots, n-1$ é movida no sentido horário para o nó seguinte do polígono. A Figura 3.1 ilustra a aplicação deste procedimento para um torneio com $n = 6$ equipes.

Após este procedimento, obtém-se uma 1-fatoração de K_n chamada de canônica [13]. A tabela correspondente à 1-fatoração gerada é duplicada para se converter em uma tabela para o torneio DRR espelhado.

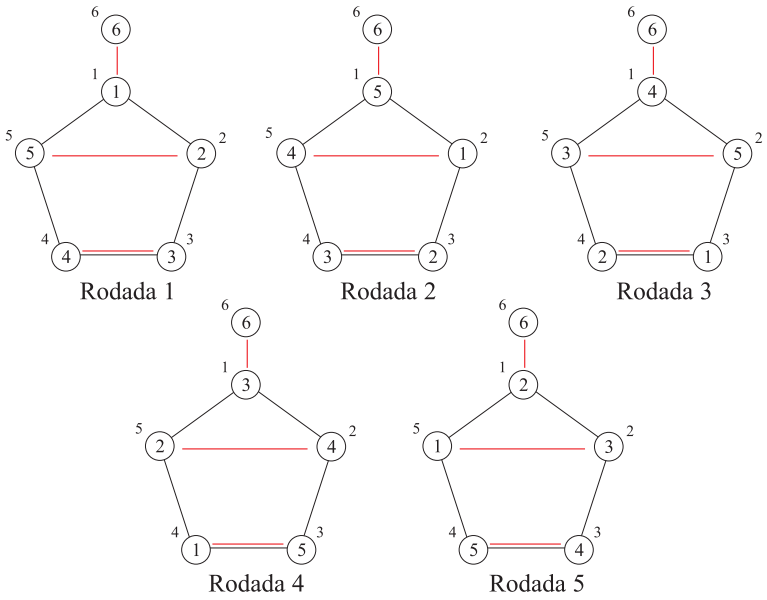

Figura 3.1: Método do polígono para $n = 6$.

3.1.2 Etapa 2: Associar equipes reais a equipes abstratas

A tabela gerada na etapa anterior é usada para gerar uma matriz quadrada de dimensão n chamada de matriz de oponentes consecutivos. Cada entrada (i, j) desta matriz é igual ao número de vezes que as equipes abstratas i e j são oponentes consecutivas de outras equipes durante o torneio. A Figura 3.2 mostra a matriz de oponentes consecutivos da tabela gerada pelo método do polígono para $n = 16$ equipes abstratas. Esta figura mostra que o método do polígono cria um certo padrão na tabela. Existem vários pares de equipes abstratas que são oponentes consecutivos de muitas outras equipes. Por exemplo, as equipes abstratas 8 e 10 são oponentes consecutivos de outras equipes 26 vezes durante o torneio. Equipes abstratas que são enfrentadas consecutivamente provavelmente deveriam ser associadas com equipes reais com distâncias curtas entre as suas cidades.

Pares de equipes reais com sedes próximas uma da outra idealmente deveriam ser associados a pares de equipes abstratas que são rivais consecutivas de muitas outras equipes durante o campeonato. Associar equipes abstratas a equipes reais (ou equipes a nós de uma 1-fatoração do grafo K_n) consiste basicamente em resolver um problema quadrático de atribuição.

Uma heurística rápida é usada nesta etapa. Primeiramente, os pares de equipes abstratas são ordenados em forma decrescente pelas entradas na matriz de oponentes consecutivos. Em seguida, as equipes reais são ordenados em forma crescente pela distância de sua sede até a de seu rival mais próximo e são associados com as equipes abstratas nessa ordem. Seja

$$\begin{pmatrix}
0 & 4 & 4 & 4 & 4 & 4 & 4 & 3 & 4 & 4 & 3 & 4 & 4 & 4 & 4 & 4 \\
4 & 0 & 2 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 2 \\
4 & 2 & 0 & 2 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 \\
4 & 25 & 2 & 0 & 2 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 25 & 2 & 0 & 2 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 25 & 2 & 0 & 2 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 25 & 2 & 0 & 2 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 25 & 2 & 0 & 2 & 26 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 25 & 2 & 0 & 1 & 26 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 1 & 0 & 2 & 25 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 26 & 2 & 0 & 2 & 25 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 2 & 0 & 2 & 25 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 2 & 0 & 2 & 25 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 2 & 0 & 2 & 25 \\
4 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 2 & 0 & 2 \\
4 & 2 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 25 & 2 & 0
\end{pmatrix}$$

Figura 3.2: Matriz de oponentes consecutivos para $n = 16$.

t_1 a próxima equipe a ser associada com uma equipe abstrata e seja t_2 a equipe com a sede mais perto da sede de t_1:

- Se t_2 já foi associada a uma equipe abstrata, então encontra-se o primeiro par (a, b) de equipes abstratas tal que a esteja associada a t_2 e b não tenha sido associada a uma equipe real. Neste caso, associa-se a equipe abstrata b à equipe real t_1.
- Se t_2 ainda não foi associada a uma equipe abstrata, então encontra-se o primeiro par (a, b) de equipes abstratas tal que nenhuma das duas tenha sido associada a uma equipe real. Neste caso, associa-se a equipe abstrata a ou b à equipe real t_1.

3.1.3 Etapa 3: Atribuir sedes aos jogos

Na ultima etapa, uma sede é atribuída a cada jogo. Para minimizar a distância total viajada, dever-se-ia programar a maior quantidade possível de jogos em cada seqüência fora de casa, para evitar que as equipes voltem as suas sedes após cada jogo. Propõe-se uma estratégia de dois passos para esta última etapa. No primeiro passo, constrói-se uma tabela viável, enquanto que, no segundo, tenta-se melhorar essa tabela reduzindo a distância total viajada usando uma busca local rápida.

As sedes são atribuídas de forma aleatória para os jogos da primeira rodada. A partir da segunda até a rodada $n-2$, é usada a seguinte estratégia

para atribuir uma sede ao jogo entre um par de equipes t_1 e t_2. Denota-se por n_{t_1} (resp. n_{t_2}) o número de jogos que a equipe t_1 (resp. t_2) jogou consecutivamente em casa ou fora de casa nas rodadas anteriores.

- Caso (1): $n_{t_2} > n_{t_1}$
  - Se a equipe t_2 jogou seu último jogo em casa, então programa-se o jogo para a sede da equipe t_1;
  - caso contrário, programa-se o jogo para a sede da equipe t_2.
- Caso (2): $n_{t_2} < n_{t_1}$
  - Se a equipe t_1 jogou seu último jogo em casa, então programa-se o jogo para a sede da equipe t_2;
  - caso contrário, programa-se o jogo para a sede da equipe t_1.
- Caso (3): $n_{t_2} = n_{t_1}$
  - Se a equipe t_1 jogou seu último jogo em casa e a equipe t_2 jogou seu último jogo fora, então programa-se o jogo para a sede da equipe t_2;
  - se a equipe t_2 jogou seu último jogo em casa e a equipe t_1 jogou seu último jogo fora, então programa-se o jogo para a sede da equipe t_1;
  - caso contrário, escolhe-se aleatoriamente uma das duas sedes.

A primeira rodada da segunda fase de um torneio MDRR é exatamente a primeira rodada da primeira fase com os mandos de campo invertidos. Em conseqüência, a última rodada e a primeira rodada com mandos de campos invertidos devem ser consideradas como rodadas consecutivas. As sedes são atribuídas aleatoriamente na última rodada. Se alguma dessas atribuições torna a tabela inviável, o mando de campo desse jogo é invertido. Se a tabela resultante é inviável, toda a atribuição de sedes é refeita até que se encontre uma tabela viável. Resultados computacionais mostraram que esta situação acontece apenas cerca de 6.3% das vezes em que a heurística é aplicada às instâncias de teste, sendo que para nenhuma instância esse valor supera 8%.

O passo final da etapa de atribuição de sedes consiste em uma rápida busca local usando a tabela anteriormente criada como solução inicial e a vizinhança Troca de Anfitriões descrita na Seção 3.2.1. O ótimo local obtido pela busca local é o resultado da heurística construtiva.

3.2 Vizinhanças

Dada uma determinada solução para um problema, define-se uma *vizinhança* dessa solução como o conjunto de soluções que podem ser atingidas efetuando-se um determinado tipo de modificação dos elementos da solução atual. Cada modificação é chamada de um movimento.

Quatro estruturas de vizinhanças são definidas a seguir. Estas serão usadas pela heurística baseada em GRASP e ILS descrita na Seção 3.4.

3.2.1 Vizinhança Troca de Anfitriões

Cada solução na vizinhança Troca de Anfitriões (TA) é obtida invertendo-se a sede de um único jogo. Dado que existam $n \cdot (n-1)/2$ jogos em cada fase de um torneio DRR, cada solução têm $n \cdot (n-1)/2$ vizinhas nesta vizinhança. Só as soluções vizinhas viáveis serão visitadas pela busca local descrita na Seção 3.3.

Quando as soluções do problema são representadas por 1-fatorizações orientadas e ordenadas de K_n, um movimento nesta vizinhança corresponde a inverter a orientação de uma das arestas de um único 1-fator. A Figura 3.3 ilustra um movimento nesta vizinhança para $n = 8$ equipes.

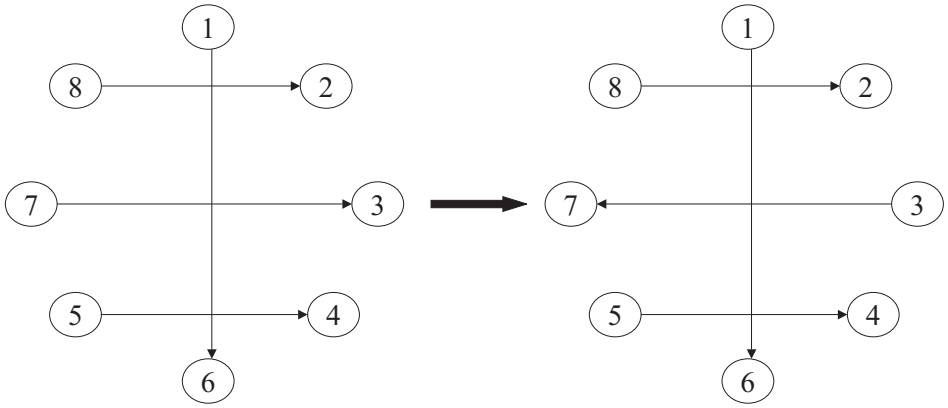

Figura 3.3: Movimento na vizinhança TA para $n = 8$, onde a sede do jogo $\{3, 7\}$ é mudada.

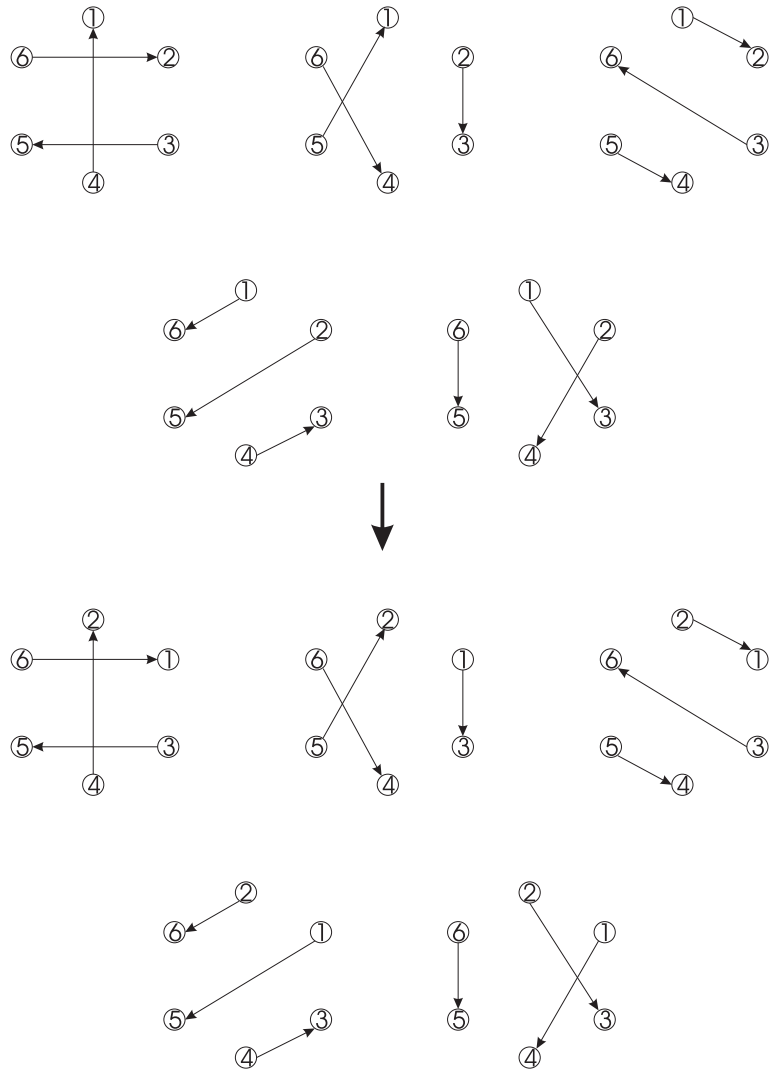

Figura 3.4: Movimento na vizinhança TE para $n = 6$, onde as equipes 1 e 2 foram trocadas.

3.2.2 Vizinhança Troca de Equipes

Cada solução na vizinhança Troca de Equipes (TE) é obtida trocando-se os adversários de um par de equipes t_1 e t_2 em todas as rodadas. Se a equipe t_3 joga com t_1 em casa (resp. fora) em uma determinada rodada, na solução vizinha jogará contra a equipe t_2 em casa (resp. fora) nessa mesma rodada. Dado que t_1 ficará com a seqüência de jogos que a equipe t_2 tinha e vice-versa, a sede do jogo entre as equipes t_1 e t_2 tem que ser invertida para garantir uma solução viável. Quando as soluções do problema são consideradas como 1-fatorizações orientadas e ordenadas de K_n, um movimento nesta vizinhança corresponde a trocar dois nós em todos os 1-fatores. A Figura 3.4 ilustra um movimento nesta vizinhança para $n = 6$ equipes.

3.2.3
Vizinhança Troca Parcial de Rodadas

Sejam t_1, t_2, t_3 e t_4 quatro equipes e sejam r_1 e r_2 duas rodadas tais que os jogos $\{t_1, t_3\}$ e $\{t_2, t_4\}$ acontecem na rodada r_1 e os jogos $\{t_1, t_4\}$ e $\{t_2, t_3\}$ acontecem na rodada r_2. Um movimento na vizinhança Troca Parcial de Rodadas (TPR) consiste em trocar as rodadas onde estes jogos acontecem. Os jogos $\{t_1, t_3\}$ e $\{t_2, t_4\}$ são programados para a rodada r_2 e os jogos $\{t_1, t_4\}$ e $\{t_2, t_3\}$ são programados para a rodada r_1. Todas as 16 combinações possíveis de atribuições de sedes para os quatro jogos são testadas e a melhor entre as viáveis é a escolhida.

Quando as soluções são consideradas como 1-fatorizações orientadas e ordenadas de K_n, procura-se na 1-fatoração dois 1-fatores tais que existam quatro nós com arcos entre eles nesses dois 1-fatores. Os dois arcos conectando os quatro nós nos dois 1-fatores podem ser trocados de 1-fator. O movimento é completado testando-se as 16 possíveis orientações para os quatro arcos envolvidos. A Figura 3.5 ilustra um movimento nesta vizinhança para $n = 8$ equipes.

As condições de aplicabilidade deste movimento (quatro equipes jogando entre elas em duas rodadas) não aparecem em tabela alguma com $n = 6$ equipes, dado que se isso acontecesse duas equipes deveriam se enfrentar duas vezes em cada turno do campeonato. Essas condições também não aparecem nas tabelas geradas pelo método do polígono para $n = 8, 12, 14, 16, 20$ e 24 equipes.

3.2.4
Vizinhança Rotação de Jogos

A vizinhança Rotação de Jogos (RJ) é uma generalização da vizinhança TPR e consiste em programar um determinado jogo em uma determinada rodada, seguido das modificações necessárias para evitar que as equipes joguem mais de um jogo por rodada. As modificações que têm que ser feitas geram um movimento de cadeia de ejeção. As cadeias de ejeção estão baseadas na idéia de efetuar movimentos pequenos de alguns elementos da solução que obrigam a que outros elementos sejam ejetados de seus estados atuais, posição ou atribuição de valor [29, 30].

Uma vez mais as soluções são representadas como 1-fatorizações orientadas e ordenadas de K_n. Primeiramente, são removidas as orientações de todos os arcos. Um movimento nesta vizinhança começa transferindo-se uma aresta (t_1, t_2) do 1-fator Z onde estava originalmente para um

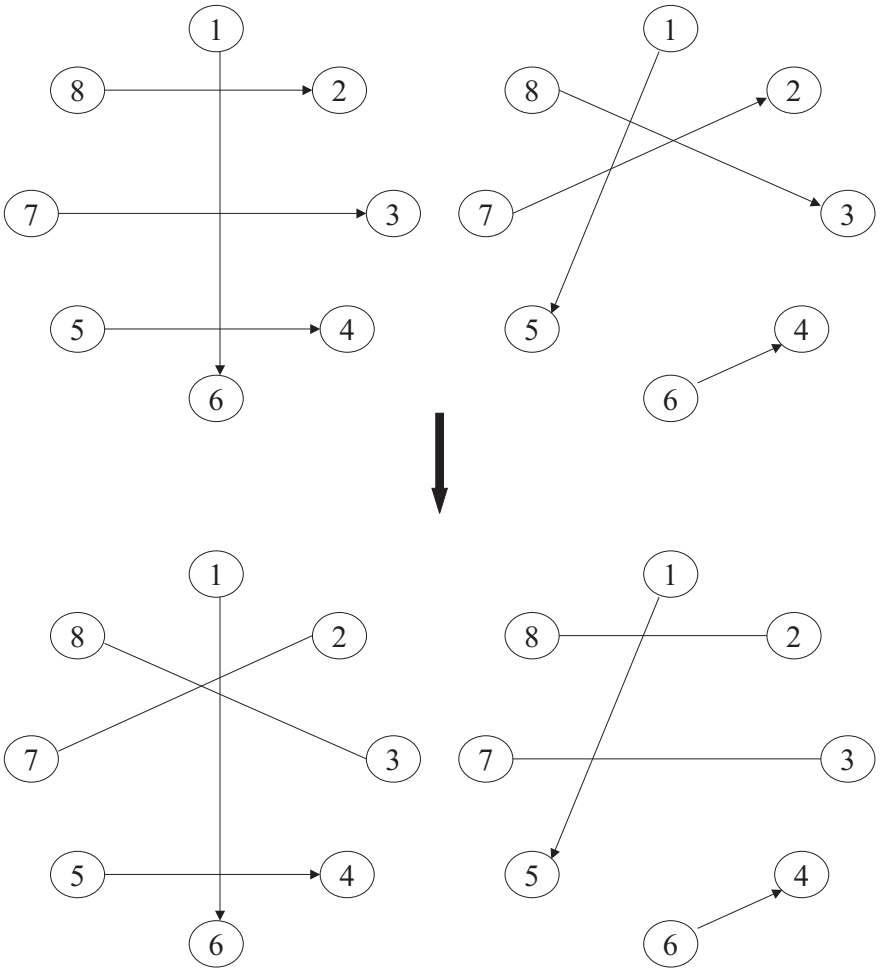

Figura 3.5: Movimento na vizinhança TPR para $n = 8$, onde os oponentes das equipes 2 e 3 em duas rodadas diferentes são trocados.

novo 1-fator A. Sejam t_3 e t_4 os nós originalmente adjacentes a t_1 e t_2, respectivamente, no 1-fator A. Se a aresta (t_1, t_2) foi movida para A, a aresta (t_3, t_4) também ter que ser movida para A, dado que as arestas (t_1, t_3) e (t_2, t_4) terão que ser transferidas para outro 1-fator. Seja B o 1-fator que originalmente tinha a aresta (t_3, t_4). Move-se a aresta (t_1, t_3) de A para B. Se $B = Z$, a aresta que falta em Z para ser um 1-fator é (t_2, t_4), a mesma que sobra em A. Transferindo-se (t_2, t_4) de A para Z cria-se uma nova 1-fatoração e a cadeia de ejeção termina. Se $B \neq Z$, t_4 tem grau 0 e t_1 tem grau 2 em B. Seja t' o nó originalmente adjacente a t_1 em B e seja C o 1-fator que tem a aresta (t', t_4). Transfere-se (t', t_4) de C para B e (t_1, t') de B para C. Se $C = Z$, a cadeia de ejeção termina transferindo-se (t_2, t_4) de A para Z. Caso contrário a cadeia de ejeção continua até que o 1-fator corrente seja Z.

Observe-se que a cadeia de ejeção sempre se move do 1-fator onde originalmente está a aresta (t_1, x) para o fator onde originalmente está a

aresta (x, t_4). Para que um dado 1-fator $F \neq A$, contendo as arestas (t_1, x) e (y, t_4), participe da cadeia de ejeção, esta deve passar pelo 1-fator onde está a aresta (t_1, y) no passo imediatamente anterior e pelo 1-fator onde está a aresta (x, t_4) no passo imediatamente seguinte. Então, para todo 1-fator F que é visitado pela cadeia de ejeção, existe um único 1-fator F' pelo qual a cadeia de ejeção passa imediatamente antes e um único 1-fator F'' pelo qual a cadeia de ejeção passa imediatamente a seguir.

Demonstra-se a seguir que a cadeia de ejeção anteriormente definida termina depois de visitar (no pior caso) todos os fatores uma única vez. O que se pretende mostrar é que se algum fator é visitado duas vezes pela cadeia de ejeção, então esse é o fator A (onde a cadeia de ejeção começa) e nesse momento o procedimento termina.

Suponha-se que a cadeia de ejeção visite mais de uma vez alguns 1-fatores diferentes de A. Então existe um desses 1-fatores que é visitado pela segunda vez antes de qualquer outro. Seja F esse 1-fator. Considere-se agora o 1-fator F' (aquele que é visitado imediatamente antes de F). Se $F' \neq A$, então F' tem que ser visitado pela segunda vez antes de F, o que é absurdo pela hipótese de que F é o primeiro 1-fator diferente de A que é visitado pela segunda vez. Se $F' = A$, então A tem que ser visitado pela segunda vez antes de F. Entretanto, como a cadeia de ejeção termina quando A é visitado pela segunda vez, isto também é um absurdo. Em conseqüência, todo 1-fator diferente de A é visitado pela cadeia de ejeção no máximo uma vez e o 1-fator A é visitado duas vezes (uma no começo e outra no final).

A atribuição de sedes para cada jogo deve ser refeita depois que a cadeia de ejeção termina. Uma opção válida é refazer-se a atribuição de sedes por completo, utilizando-se a terceira parte da heurística construtiva.

Propõe-se aqui uma estratégia mais eficaz, procurando aproveitar a atribuição de sedes da tabela à qual foi aplicada a cadeia de ejeção.

Todas as arestas correspondentes a jogos que não foram afetados pela cadeia de ejeção recuperam sua orientação original. A orientação de cada aresta afetada pela cadeia de ejeção é primeiramente determinada de forma aleatória. Se a orientação atribuída a uma destas arestas torna a tabela inviável, sua orientação é trocada. Se no final da atribuição a tabela fica inviável, aplica-se uma heurística simples de busca tabu [31] cujo objetivo é a minimização do número de inviabilidades. Para cada equipe, conta-se uma inviabilidade por cada jogo consecutivo em casa ou fora, acima do limite máximo de três. A heurística de busca tabu faz uso exclusivamente da vizinhança TA e termina quando o número de inviabilidades chega a zero. Está heurística é muito rápida, pois a quantidade

de inviabilidades usualmente observadas depois de feita a atribuição de sedes é muito pequena.

Só as vizinhanças TPR e RJ são capazes de mudar a estrutura da 1-fatoração construída pelo método do polígono. A vizinhança RJ tem um papel muito importante na procura de boas soluções para o MTTP. Os movimentos nesta vizinhança são capazes de encontrar soluções que não podem ser alcançadas através das outras vizinhanças. Em particular, movimentos na vizinhança TPR podem aparecer depois de uma movimento de cadeias de ejeção em situações onde nenhum movimento TPR existia. Se as cadeias de ejeção não fossem usadas, a heurística poderia ficar presa em tabelas com a estrutura das soluções construídas pelo método do polígono.

3.3 Busca local

Uma *busca local* é um procedimento que explora sistematicamente o espaço de soluções de um problema, utilizando o conceito de vizinhanças. Começando de uma solução inicial, a busca local explora a vizinhança na procura de uma solução melhor do que a corrente. A solução corrente muda sucessivamente, até chegar-se a um ótimo local.

Uma busca local é do tipo *melhor aprimorante* se percorre sempre toda a vizinhança de uma solução e a solução corrente sempre muda para a melhor solução vizinha. Uma busca local é do tipo *primeiro aprimorante* se a solução corrente muda para a primeira solução vizinha melhor do que ela.

3.3.1 Busca local para o MTTP

Quatro estruturas de vizinhança para o MTTP foram definidas na seção anterior. As três primeiras (TA, TE e TPR) são exploradas pela busca local. A vizinhança RJ, baseada em cadeias de ejeção, será explorada apenas em uma estratégia de diversificação, através de movimentos aplicados com menor freqüência na heurística descrita na próxima seção, devido ao alto custo computacional da geração e avaliação destes movimentos.

O algoritmo de busca local usa uma estratégia do tipo primeiro aprimorante similar à do procedimento VND [33, 43]. TE é a primeira vizinhança explorada (as soluções a partir das quais a busca local é aplicada sempre são ótimos locais com respeito à vizinhança TA). Uma vez que um

ótimo local com respeito a esta vizinhança é atingido, uma busca local rápida na vizinhança TA é executada na procura de uma melhor atribuição de sedes. Em seguida, a vizinhança PRS é investigada. Como no caso anterior, uma vez que um ótimo local com respeito a esta vizinhança é encontrado, o algoritmo executa uma rápida busca local na vizinhança TA à procura de uma melhor atribuição de sedes. Este esquema se repete até que um ótimo local com respeito às três vizinhanças seja obtido. As soluções vizinhas são visitadas de forma aleatória dentro de cada vizinhança.

3.3.2 Recálculo do custo de soluções vizinhas

Dada uma solução corrente e seu custo (isto é, a distância viajada pelas equipes), procura-se determinar o custo de uma solução vizinha com o menor esforço computacional possível. O recálculo eficiente do custo de soluções vizinhas é fundamental no desempenho de algoritmos baseados em busca local.

Para cada equipe, armazena-se na memória a distância viajada para jogar em cada uma das rodadas e a distância viajada para voltar à sua cidade ao final do torneio (funções d_c, d_i e d_f no capítulo anterior). Também é armazenada a distância total viajada por cada equipe.

Com estas estruturas, o custo de uma solução vizinha na vizinhança TA pode ser avaliado em tempo $O(1)$. Sejam t_1 e t_2 as duas equipes envolvidas no movimento e seja q a rodada em que elas se enfrentam. Depois de trocar a sede do jogo $\{t_1, t_2\}$, só é preciso recalcular a distância viajada pelas duas equipes para jogar as rodadas q e $q+1$, nos dois turnos do campeonato.

O custo de uma solução vizinha na vizinhança TE pode ser avaliado em tempo $O(n)$. É preciso reavaliar por completo (em tempo $O(n)$) as distâncias viajadas pelas duas equipes envolvidas no movimento. Para as demais equipes, é preciso avaliar as distâncias viajadas antes e depois das duas rodadas em que se enfrentam com as equipes envolvidas no movimento ($O(1)$ para cada uma das $n-2$ equipes, isto é, $O(n)$).

O custo de uma solução vizinha na vizinhança TPR pode ser avaliado em tempo $O(1)$. É preciso reavaliar as distâncias viajadas pelas quatro equipes envolvidas no movimento, antes e depois das duas rodadas envolvidas no movimento. Está avaliação é feita 16 vezes (uma vez para cada combinação de atribuição de sedes aos jogos envolvidos no movimento). Considerando-se que a maioria das instâncias de teste têm menos de 16

equipes, é normal que na prática a reavaliação do custo das soluções vizinhas na vizinhança TE seja quase sempre mais rápida que na vizinhança TPR.

3.4 Heurística híbrida GRASP com ILS

A metaheurística GRASP (do inglês *Greedy Randomized Adaptive Search Procedure*) [25, 26, 47] é um processo iterativo, no qual cada iteração consiste em duas fases: construção e busca local. A fase de construção cria uma solução viável, cuja vizinhança é investigada até que um ótimo local seja encontrado durante a fase de busca local. A melhor solução obtida é guardada e retornada ao final do processo iterativo. Um extenso *survey* da literatura relacionada é apresentado em [27].

A metaheurística ILS (do inglês *Iterated Local Search*) [36, 38, 37] começa de um ótimo local. Uma perturbação aleatória é aplicada à solução corrente seguida de uma busca local. Se o ótimo local obtido após esses passos satisfaz um determinado critério de aceitação, ele é aceito como a nova solução corrente, caso contrário a solução corrente não muda. A melhor solução encontrada é eventualmente atualizada e os passos acima são repetidos até que um determinado critério de parada seja atingido.

Propõe-se uma hibridação das metaheurísticas GRASP e ILS em uma heurística para o MTTP. Basicamente, substitui-se a fase de busca local da metaheurístca GRASP por um procedimento ILS. O fato de haver algumas vizinhanças pequenas e de exploração rápida (para serem usadas em um procedimento de busca local) e uma vizinhança maior e mais difícil de ser avaliada (para ser usada como perturbação) justifica a escolha do método. O pseudo-código do Algoritmo 1 resume a heurística `GRILS-MTTP` para encontrar boas soluções viáveis para o problema do torneio com viagens espelhado.

O ciclo externo no Algoritmo 1 realiza `MaxIter` iterações GRASP (nos experimentos computacionais, o número máximo de iterações foi substituído por um limite no tempo de processamento). Cada iteração começa com a construção de uma solução inicial S obtida a partir da heurística descrita na Seção 3.1, implementada pelo algoritmo `ConstrucãoGulosaAleatorizada`. O critério guloso utilizado na segunda etapa dessa heurística (ver Seção 3.1.2) é aleatorizado, aplicando-se $n/8$ trocas na ordem na qual as equipes reais serão associadas às equipes abstratas.

```
Procedimento GRILS-MTTP();
Dados : MaxIter
Resultado: Solução S*
for i = 1, ..., MaxIter do
    S ← ConstrucãoGulosaAleatorizada();
    S̲, S ← BuscaLocal(S);
    repeat
        S' ← Perturbacão(S);
        S' ← BuscaLocal(S');
        S ← CritérioDeAceitacão(S, S');
        S* ← AtualizarMelhorSolucão(S, S*);
        S̲ ←
        AtualizarMelhorSolucãoNaIteracão(S, S̲);
    until CritérioDeReinicializacão;
end
```

Algoritmo 1: Pseudo-código de uma heurística híbrida GRASP com ILS para minimização.

Em seguida, o algoritmo `BuscaLocal` é aplicado à solução inicial, retornando uma nova solução corrente S. Esta solução também é usada para inicializar a melhor solução $\underline{S}$ na iteração GRASP corrente. O procedimento de busca local descrito na Seção 3.3 é implementado pelo algoritmo `BuscaLocal`.

O ciclo interior começa com a aplicação de uma perturbação na solução corrente S usando o algoritmo `Perturbacão` e obtendo a nova solução S'. Este algoritmo implementa uma movimento aleatório na vizinhança RJ (baseada em cadeias de ejeção) descrita na Seção 3.2.4, seguido de uma busca local na vizinhança TA. O algoritmo `BuscaLocal` é aplicado a S'.

A nova solução S' é aceita ou não como a nova solução corrente, dependendo do resultado do critério de aceitação implementado pelo algoritmo `CritérioDeAceitacão` usando o parâmetro β. Este parâmetro é inicializado com 0.001 ao começo de cada iteração GRASP e reinicializado com o mesmo valor cada vez que a solução corrente muda. A solução S' é aceita como a nova solução corrente se seu custo é menor do que $(1 + \beta)$ vezes o custo da solução corrente S. Se a solução corrente não muda após um determinado número de iterações (usou-se $12n$ na implementação), o valor de β é dobrado (i. e., o critério de aceitação é relaxado).

A melhor solução obtida S^* e a melhor solução na iteração GRASP corrente são eventualmente atualizadas e um novo ciclo começa com uma perturbação da solução corrente, até que o critério de reinicialização implementado pelo algoritmo `CritérioDeReinicializacão` seja atingido. Neste caso, uma nova iteração GRASP começa se 50 movimentos não aprimorantes

foram aceitos desde a última vez em que a melhor solução na iteração GRASP foi atualizada. A reinicialização ocorre se demasiadas perturbações seguidas de busca local foram executadas sem melhorar a melhor solução na iteração GRASP. Desta forma, procura-se fazer com que a iteração GRASP não seja interrompida se a solução corrente S ainda está melhorando.

3.5 Resultados computacionais

Os algoritmos descritos neste capítulo foram codificados em C++ e compilados com a versão 2.96 do compilador `gcc` com a opção de otimização -O3. Os experimentos foram executados em um computador Pentium IV com um relógio de 2.0 GHz e 512 Mbytes de memória RAM.

3.5.1 Instâncias de teste

Duas classes de instâncias foram originalmente propostas e descritas em [20].

A primeira classe é composta por instâncias circulares, geradas artificialmente para testar se os problemas do torneio com viagens são mais fáceis quando o problema do caixeiro viajante na mesma instância é trivial. O nome `circ`n é usado para denotar a instância circular com $4 \leq n \leq 20$ equipes. Os nós são colocados em uma circunferência de tal forma que a distância de um nó a seus dois nós vizinhos é igual a 1. Os nós são numerados crescentemente $0, 1, \ldots, n-1$. Então, a distância entre os nós i e j com $i > j$ é dada pelo comprimento do caminho mais curto entre eles e é igual ao mínimo entre $i - j$ e $j - i + n$.

A segunda classe de instâncias foi criada usando-se as distâncias entre as cidades de um sub-conjunto das equipes da *National League* da MLB dos Estados Unidos. Denota-se `nl`n a instância da *National League* com $4 \leq n \leq 16$ equipes. Todas estas instâncias estão disponíveis em [59]. Para os experimentos, foram desconsideradas as instâncias menores, com $n = 4$ e $n = 6$ equipes.

3.5.2 Resultados numéricos

Foram executados três experimentos computacionais concernentes à heurística construtiva descrita na Seção 3.1 e à heurística `GRILS-MTTP` baseada em GRASP e ILS.

No primeiro experimento, a versão aleatorizada da heurística construtiva foi executada 1000 vezes com diferentes sementes para cada instância. Os resultados numéricos são relatados na Tabela 3.1. A primeira coluna mostra o nome da instância. Na segunda, terceira e quarta colunas, são relatados o pior custo, o custo médio e o melhor custo, respectivamente, das soluções obtidas nas 1000 execuções da heurística. Na quinta coluna, é relatada a diferença porcentual entre o custo da melhor solução obtida pela heurística e o custo da melhor solução conhecida para a versão não espelhada do problema no momento de obtenção destes resultados. Também é fornecido o tempo total, em segundos, das 1000 execuções da heurística.

Instância	Pior	Média	Melhor	Dif. (%)	Tempo (s)
`circ8`	214	180	156	18.2	0.16
`circ10`	390	344	306	20.5	0.27
`circ12`	652	573	486	15.7	0.45
`circ14`	1012	892	748	11.6	0.67
`circ16`	1508	1329	1138	16.6	0.94
`circ18`	2086	1880	1584	11.5	1.26
`circ20`	2818	2532	2234	17.1	1.67
`nl8`	59485	50478	44902	13.0	0.18
`nl10`	90530	80103	71092	19.3	0.30
`nl12`	167414	146365	127534	14.6	0.48
`nl14`	293675	266216	241361	27.2	0.71
`nl16`	436137	382032	329990	23.5	1.01

Tabela 3.1: Resultados computacionais obtidos pela heurística construtiva (1000 execuções).

A heurística construtiva é muito rápida. Por exemplo, 1000 execuções da instância `nl16` podem ser feitas em aproximadamente um segundo. A média da diferença porcentual com respeito à melhor solução conhecida para a versão não espelhada do problema no momento de obtenção destes resultados é 17.4% . A heurística construtiva executa em tempos computacionais várias ordens de magnitude inferiores aos de outras heurísticas para a versão não espelhada do problema. As soluções rapidamente obtidas pela heurística construtiva são, em alguns casos, melhores do que as encontradas

Instância	Não-espelhada	Espelhada	Dif. (%)	Iteração	Tempo (s)
circ8	132	140	6.1	5	1.4
circ10	254	276	8.7	289	276.0
circ12	420	456	8.6	7	8.5
circ14	670	714	4.7	1	1.1
circ16	976	1004	2.9	13	115.3
circ18	1420	(*) 1364	-3.9	19	284.2
circ20	1908	(*) 1882	-1.4	40	578.3
nl8	39721	41928	5.6	1	0.7
nl10	59583	63832	7.1	471	643.9
nl12	111248	120655	7.4	22	24.0
nl14	189766	208086	9.5	30	69.9
nl16	267194	285614	6.9	31	514.2

(*) Novas melhores soluções conhecidas para instâncias não-espelhadas.

Tabela 3.2: Resultados computacionais obtidos pela heurística GRILS-MTTP.

após vários dias de processamento pela heurística de colônia de formigas com *backtracking* e busca local de Crauwels e Van Oudheusden [10].

No segundo experimento, a heurística GRILS-MTTP foi executada uma única vez para cada instância. O tempo máximo de processamento foi fixado em 15 minutos. Para cada instância, relata-se na Tabela 3.2 o valor da melhor solução conhecida para o TTP não-espelhado no momento de obtenção destes resultados e o valor da solução obtida pela heurística GRILS-MTTP. Esses valores são seguidos da diferença porcentual entre eles. Também são relatados a iteração GRASP em que a melhor solução foi observada e o correspondente tempo de execução.

A heurística GRILS-MTTP é muito robusta. A maior diferença porcentual observada entre o valor da solução obtida pela heurística para o MTTP e a melhor solução conhecida para o TTP não necessariamente espelhado é de 9.5%. Em particular, a nova heurística foi capaz de obter soluções que melhoram as melhores soluções conhecidas para o problema mais relaxado, o TTP não-espelhado, para as instâncias circ18 e circ20 em 3.9% e 1.4%, respectivamente.

Os tempos de processamento se comparam favoravelmente quando comparados aos de outras heurísticas. Por exemplo, o tempo relatado por Anagnostopoulos et al. [2] para obter a melhor solução para a instância não espelhada nl14 é de 418358.2 segundos (quase cinco dias) em um processador AMID Athlon a 1544 MHz, enquanto o tempo dado à heurística GRILS-MTTP é de 15 minutos em uma máquina similar (i.e., aproximadamente 0.2 % do tempo gasto em [2]).

No experimento seguinte o limite de 15 minutos de execução foi substituído por um limite de dois dias, com o objetivo de obter-se soluções da melhor qualidade possível. A Tabela 3.3 mostra os valores das soluções que foram melhoradas executando-se o algoritmo com um limite de dois dias de execução. Apresenta-se também, a nova diferença porcentual com respeito à melhor solução conhecida para o TTP não-espelhado.

Instância	Melhor espelhada	Dif. (%)
circ16	990	1.4
circ18	1358	-4.4
nl16	281161	5.2

Tabela 3.3: Melhores resultados obtidos pela heurística GRILS-MTTP.

No último experimento, isola-se e quantifica-se a melhoria adicional obtida pela utilização da cadeia de ejeção envolvida na vizinhança RJ. Implementou-se uma heurística GRASP pura usando a mesma busca local usado pela heurística GRILS-MTTP. Esta busca local não faz uso da vizinhança baseada em cadeias de ejeção. O mesmo tempo de processamento de 15 minutos foi dado para os dois algoritmos. Para cada instância, relata-se na Tabela 3.4 o valor da melhor solução obtida e o tempo de processamento necessário para encontrá-la para uma única execução de cada heurística.

Instância	GRASP		GRILS-MTTP	
	Valor	Tempo (s)	Valor	Tempo (s)
circ8	154	1.6	140	1.4
circ10	282	755.7	276	276.0
circ12	458	145.0	456	8.5
circ14	714	746.3	714	1.1
circ16	1072	107.9	1004	115.3
circ18	1446	563.3	1364	284.2
circ20	2008	620.2	1882	578.3
nl8	43742	8.8	41928	0.7
nl10	64515	360.4	63832	643.9
nl12	121477	72.3	120655	24.0
nl14	215358	637.7	208086	69.9
nl16	297004	662.3	285614	514.2

Tabela 3.4: GRASP vs. heurística GRILS-MTTP.

A contribuição da vizinhança baseada em cadeias de ejeção é clara. As soluções encontradas pela heurística GRILS-MTTP são sistematicamente melhores ou iguais às encontradas pela heurística GRASP. A diferença porcentual média entre os valores das soluções obtidas é de 4.2 %. Os tempos

de processamento também se comparam favoravelmente para a heurística GRILS-MTTP. Ou seja, a heurística GRILS-MTTP não apenas obtém soluções melhores, como também as encontra mais rapidamente do que a versão que não utiliza cadeias de ejeição.

Além do artigo publicado com os resultados descritos neste capítulo da tese [51], o único artigo que trata do MTTP foi apresentado por Henz [34]. Neste trabalho procura-se combinar busca local com programação por restrições [39]. Os resultados apresentados são bastante inferiores aos resultados relatados nesta tese. Para nl8, nl10 e nl12 são encontradas soluções com custos 42142, 65464 e 135950, respectivamente. Os custos das soluções obtidas nesta tese são menores nos três casos. Não são relatados resultados para outras instâncias em [34].

3.6 Conclusões

Neste capítulo investigou-se a versão espelhada do problema do torneio com viagens (MTTP). Essa estrutura de torneio é muito comum na América Latina.

A primeira contribuição deste capítulo é uma heurística construtiva rápida para o MTTP, que executa em aproximadamente um milésimo de segundo para as maiores instâncias de teste em um computador Pentium IV com 2.0 GHz. Embora os valores das soluções obtidas estejam na média 17.4% acima das melhores soluções conhecidas para o TTP não-espelhado, as melhores tabelas espelhadas encontradas pela heurística em 1000 execuções são, em algumas ocasiões, melhores do que as obtidas por outras heurísticas para o TTP não-espelhado em vários dias de execução. A heurística construtiva é uma ferramenta eficiente para construir soluções iniciais para algoritmos mais complexos.

A segunda contribuição é a vizinhança Rotação de Jogos, baseada em cadeias de ejeção. Está vizinhança é essencial na busca de boas soluções para o MTTP, sendo capaz de levar à obtenção de soluções não atingíveis apenas com estruturas de vizinhanças mais simples. Esta importância foi comprovada por experimentos computacionais.

Finalmente, a terceira contribuição deste capítulo consistiu em uma heurística híbrida para o MTTP, baseada nas metaheurísticas GRASP e ILS. A heurística GRILS-MTTP obteve soluções muito boas para o MTTP. Para duas instâncias de teste, a nova heurística obteve soluções melhores do que as previamente conhecidas para a versão mais relaxada do TTP na

qual as tabelas podem ou não ser espelhadas. Os tempos de processamento da nova heurística são bem menores que os de outros algoritmos para o TTP não-espelhado. A heurística `GRILS-MTTP` é muito mais rápida do que o algoritmo *simulated annealing* considerado atualmente como o melhor algoritmo para o TTP não-espelhado. Por exemplo, a nova heurística executa aproximadamente 500 vezes mais rápida para a instância `nl114`.

Como conseqüência do trabalho desenvolvido neste capítulo da tese foram publicados os artigos [51, 49].