LIMITES PARA O PROBLEMA DO TORNEIO COM VIAGENS

Victor Hugo Rodrigues do Nascimento

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Luidi Gelabert Simonetti

Rio de Janeiro
Março de 2019

LIMITES PARA O PROBLEMA DO TORNEIO COM VIAGENS

Victor Hugo Rodrigues do Nascimento

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Luidi Gelabert Simonetti, D.Sc.

Prof. Abilio Pereira de Lucena Filho, D.Sc.

Prof. Rafael Augusto de Melo, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2019

Nascimento, Victor Hugo Rodrigues do

Limites para o Problema do Torneio com Viagens/Victor Hugo Rodrigues do Nascimento. – Rio de Janeiro: UFRJ/COPPE, 2019.

X, 51 p.: il.; 29, 7cm.

Orientador: Luidi Gelabert Simonetti

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2019.

Referências Bibliográficas: p. 47 – 51.

1. Traveling Tournament Problem. 2. Problema do Torneio com Viagens. 3. Geração de Colunas. I. Simonetti, Luidi Gelabert. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Inicialmente agradeço a minha mãe Irene Rodrigues do Nascimento, a quem amo muito e que sempre me deu apoio, carinho e atenção. Agradeço também a minha avó Josefa da Silva Souza, que sempre ajudou a mim e à minha mãe, durante toda a nossa vida. Devo a elas não só este trabalho como tudo que conquistei até hoje. Agradeço à minha irmã Laura e a todos os meus familiares que são quem sempre me apoiam na vida.

Agradeço ao meu orientador Luidi, pelo apoio durante o mestrado e pela compreensão nos períodos de dificuldade que passei. Agradeço também aos membros da banca, os professores Abilio e Rafael, por suas críticas construtivas e sugestões ao trabalho aqui apresentado.

Agradeço aos muitos amigos que fiz durante a graduação, que sempre me ajudam e me fazem viajar até os confins da baixada fluminense, para que possamos nos encontrar e dar risadas e também discutir sobre assuntos aleatórios.

Por fim, agradeço aos amigos do Labotim pelo convívio e pelo suporte oferecido durante o mestrado. Agradeço também à UFRJ e ao PESC pelo suporte e à CAPES pelo suporte financeiro nos primeiros anos do mestrado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

LIMITES PARA O PROBLEMA DO TORNEIO COM VIAGENS

Victor Hugo Rodrigues do Nascimento

Março/2019

Orientador: Luidi Gelabert Simonetti

Programa: Engenharia de Sistemas e Computação

O problema do torneio com viagens (TTP, do inglês *Traveling Tournament Problem*) é um importante problema da área de agendamento esportivo. Dados n times e as distâncias entre os estádios dos times, o TTP consiste em encontrar um torneio com $2(n-1)$ rodadas onde cada time deve jogar $n-1$ rodadas em casa e deve viajar nas outras $n-1$ rodadas para enfrentar cada um de seus rivais. A distância percorrida por todos os times deve ser mínima. Este trabalho consiste na implementação de um algoritmo de geração de colunas para o TTP e como contribuições originais dois algoritmos de *pricing* são propostos. Estes algoritmos são adaptados do problema de roteamento de veículos capacitado (CVRP). Duas técnicas de estabilização para a geração de colunas também foram implementadas: uma baseada na ideia de *box* de estabilidade e a outra utiliza combinações convexas das soluções duais obtidas. Por fim, uma heurística de *pricing* e uma metaheurística ILS (*Iterated Local Search* - Busca Local Iterada) foram propostas. Os novos algoritmos de *pricing* propostos permitiram a execução da geração de colunas para as instâncias do problema com tamanho de até $n = 24$ times. Os limites inferiores encontrados eram mais fracos em comparação com os conhecidos, porém foram obtidos de forma mais rápida.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

BOUNDS FOR THE TRAVELING TOURNAMENT PROBLEM

Victor Hugo Rodrigues do Nascimento

March/2019

Advisor: Luidi Gelabert Simonetti

Department: Systems Engineering and Computer Science

The traveling tournament problem (TTP) is a important problem in sports scheduling. Given n teams and the distances between the team's venues, the TTP consists in finding a tournament with $2(n-1)$ rounds where each team must play $n-1$ rounds at his home venue and must travel to each of its rivals venues in the other $n-1$ rounds. The distance traveled by all teams must be minimum. This work consists on a column generation algorithm to the TTP and as original contributions two pricing algorithms are proposed. These algorithms are adapted from the capacitated vehicle routing problem (CVRP). Two stabilization techniques for column generation were also implemented: one based on the idea of stability box and the other uses convex combinations of the obtained dual solutions. Finally, a pricing heuristic and an ILS metaheuristic were proposed. The new pricing algorithms proposed allowed running the column generation for instances with size up to $n = 24$ teams. The lower bounds found were weaker compared to those previously known, but were obtained more quickly.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O mercado esportivo movimenta milhões ao ano. Em 2017, o clube de futebol do Flamengo obteve uma receita total de R$ 648,7 milhões e clubes como Palmeiras e São Paulo obtiveram receitas de R$ 500 milhões cada [1]. Grande parte desta receita corresponde à direitos de transmissão dos jogos e à ingressos no estádio. Em 2017, somente por direitos de transmissão na TV aberta, o Flamengo recebeu R$ 170 milhões, enquanto Palmeiras e São Paulo receberam R$ 100 milhões cada[2]. Frente a esse mercado milionário, as organizações de campeonatos como o Brasileirão e outros campeonatos ao redor do mundo precisam garantir uma agenda de jogos economicamente interessantes. É interessante que jogos clássicos, que rendem maior audiência, devem ser agendados de forma a maximizar essa audiência. Por outro lado, em algumas ligas esportivas amadoras a ordem dos jogos pode influenciar os custos de viagens para os times, e é interessante que os jogos sejam agendados de forma a minimizar esses custos. A resolução deste e de outros problemas relacionados deu origem à área de pesquisa conhecida como *sports scheduling*. Diversos problemas já foram estudados em vários esportes: futebol, basquetebol, críquete, basebol, hóquei, tênis, etc [3].

1.1 Torneios *Round-Robin*

A maior parte dos problemas em *sports scheduling* envolve construir uma agenda de jogos para n times, onde n é par. Em eventos como a fase de grupos da Copa do Mundo FIFA, onde todos os times jogam numa mesma área, sem o conceito de mando de jogo, diz-se que o torneio é do formato ***Round-Robin* simples (SRRT -Simple Round Robin Tournament)**. Neste tipo de agendamento, cada par de times se enfrenta uma só vez. Com n times tem-se $n-1$ rodadas e em cada rodada ocorrem $\frac{n}{2}$ jogos. A tabela 1.1 ilustra o agendamento SRRT do grupo E da Copa do Mundo FIFA 2018. Quando o número de times participantes n é ímpar, cada time deve ficar sem jogar em alguma rodada (cada time tem uma rodada "*bye*"). Esta

condição é modelada adicionando um time nulo $n+1$, tornando assim o número de times par. O time t que for agendado pra jogar contra $n+1$, deve ficar sem jogar naquela rodada.

Time\Rodada	1	2	3
Brasil	Suíça	Costa Rica	Sérvia
Costa Rica	Sérvia	Brasil	Suíça
Sérvia	Costa Rica	Suíça	Brasil
Suíça	Brasil	Sérvia	Costa Rica

Tabela 1.1: Grupo E da Copa do Mundo FIFA 2018. As linhas correspondem aos $n-1$ confrontos de cada um dos n times.

O Campeonato Brasileiro de futebol é outro exemplo de competição que utiliza torneios *Round-Robin*. Em cada jogo, um time detém o chamado mando de jogo. Normalmente a partida é disputada no estádio do time com o mando de jogo e o time detém maior parte das cadeiras na arquibancada, sendo conhecido popularmente como o "jogar em casa". Para que, no confronto entre os times A e B, cada time tenha a chance de obter o mando de jogo, um torneio ***Round-Robin*** **duplo (DRRT - Double Round Robin Tournament)** deve ser elaborado. Neste caso, o torneio é dividido em dois turnos: se no primeiro turno A foi o mandante na partida A vs B, no segundo turno A e B devem se enfrentar novamente, desta vez com o mando de jogo atribuído ao time B. Um torneio *Round-Robin* **espelhado** é construído com um SRRT no primeiro turno e considerando, para o segundo turno, as mesmas partidas do primeiro turno com o mando de jogo invertido. A tabela 1.2 ilustra um torneio DRRT espelhado com $n = 4$ times, numerados de 1 a 4. As entradas negativas na tabela representam que o time da linha i sai de casa para jogar contra o time da entrada negativa. Por exemplo, a primeira partida do time 1 ocorre no estádio do time 4. Nos dois exemplos anteriores, os torneios são ditos compactos. Isto é, em cada rodada todos os times jogam e assim o torneio possui um número mínimo de rodadas.

T\R	1	2	3	4	5	6
1	-4	3	2	4	-3	-2
2	-3	4	-1	3	-4	1
3	2	-1	4	-2	1	-4
4	1	-2	-3	-1	2	3

Tabela 1.2: DRRT espelhado. Note que no segundo turno (rodadas 4,5 e 6) a ordem é a mesma das 3 primeiras rodadas, com o mando de jogo invertido.

Dependendo dos interesses da organização, o campeonato não precisa ser DRRT espelhado. Nesse caso, o primeiro e segundo turnos podem ter uma ordem de confrontos diferentes.

Uma forma alternativa de analisar os torneios *Round-Robin* consiste em separar um torneio em dois componentes: Uma agenda de jogos e um padrão de mando de jogos. O padrão de mando de jogos (HAP: *Home-Away Pattern*) de um time t consiste numa string composta por H's nas rodadas onde t joga em casa e por A's nas rodadas onde t joga fora de casa. A agenda de jogos consiste numa matriz com as partidas de cada rodada, sem as especificações de mando de jogo. Em [4], a tarefa de construir um torneio *Round-Robin* é dividida em construir um HAP e uma agenda de jogos. A tabela 1.3 exibe os HAP's do torneio da tabela 1.2.

Tabela 1.3: Padrões de mando de jogo para o torneio da tabela 1.2

Time	HAP
1:	AHHHAA
2:	AHAHAH
3:	HAHAHA
4:	HAAAHH

1.2 Principais Restrições

Para construir um torneio *Round-Robin*, a única restrição necessária é garantir que todos os times se enfrentem uma única vez. As organizações podem impor restrições adicionais, que garantam um torneio mais justo, competitivo e lucrativo. Duas das principais restrições utilizadas nos trabalhos da literatura são:

- ***AtMost***: Para manter o campeonato justo, é necessário que todos os times se alternem de forma aproximadamente igual entre partidas dentro e fora de casa. Quando um time passa k rodadas jogando em casa, diz-se que ele está em um ***home stand*** a k rodadas. Alternativamente, quando um time está há k rodadas jogando fora de casa, diz-se que este time está em uma ***road trip*** a k rodadas. Na restrição ***AtMost*** nenhuma *road trip* e/ou *home stand* deve exceder U rodadas. Em todos os trabalhos da literatura, este incluso, define-se $U = 3$.

- ***NoRepeat***: Para muitas organizações não é interessante que partidas se repitam em rodadas consecutivas. Na restrição ***NoRepeat*** se quaisquer times A e B se enfrentam na rodada s, o segundo confronto A vs B (com o mando de jogo invertido) não pode ocorrer na rodada $s + 1$. Note que se o torneio a ser construído for do tipo espelhado, a restrição é naturalmente respeitada.

Na prática, outras restrições podem ser adicionadas como fixar jogos importantes em certas rodadas. Por exemplo, é interessante que partidas com rivalidades clássicas

ocorram próximo ao final do campeonato. Um outro exemplo de restrição possui natureza geográfica: Em [5] é realizado o agendamento do campeonato chileno de futebol. Devido ao fato do território do Chile ser muito estreito de leste a oeste e longo de norte a sul, são definidas três regiões: Norte, central e sul. Para a resolução do problema foi estabelecida a restrição de que se na rodada s o time t jogou na zona norte (ou sul), na rodada $s+1$ ele não pode jogar na região sul (ou norte). Estas são apenas alguns exemplos de restrições que podem surgir durante a construção de um torneio.

1.3 *Round-Robin* e Teoria dos Grafos

Utilizando teoria dos grafos, é possível representar torneios *Round-Robin* de duas formas: através de coloração de arestas em grafos e da decomposição de grafos em 1-fatores. Dados n times, considere o grafo completo K_n, onde cada vértice corresponde a um time e cada aresta uv representa um jogo entre u e v. Uma coloração de arestas válida com $n-1$ cores equivale a um SRRT, onde cada cor representa uma rodada. Para representar um DRRT, basta considerar o K_n direcionado e encontrar uma coloração de arcos válida com $2(n-1)$ cores. Um conceito equivalente ao de coloração de arestas é a decomposição do K_n em 1-fatores. Dado um grafo $G=(V,E)$, um **1-fator** é um subgrafo de G onde todos os vértices possuem grau 1. Se $|V|=n$, um 1-fator também pode ser visto como um conjunto de $\frac{n}{2}$ arestas não adjacentes. Ao particionar o conjunto de arestas em 1-fatores disjuntos, tem-se uma **1-fatoração**. Construir um SRRT é equivalente à encontrar $n-1$ 1-fatores diferentes em K_n, e consequentemente, uma 1-fatoração. Para representar um DRRT, basta determinar uma 1-fatoração no grafo K_n dirigido. A figura 1.1 ilustra uma coloração de arestas do torneio da tabela 1.2, onde cada estilo de arco indica uma cor diferente. O arco (a,b) indica que o time a viaja até o estádio do time b. Note no exemplo, que a coloração de arcos é equivalente à encontrar $2(n-1)=6$ 1-fatores, pois cada cor define um subgrafo com arcos não adjacentes.

1.4 O Método do Círculo

O algoritmo mais conhecido e utilizado para construção de um SRRT viável é o chamado método do círculo, também conhecido como método do polígono, descrito pela primeira vez em 1847 [6]. Para n times, define-se um time (o pivô) e forma-se um círculo com os $n-1$ outros times. O pivô fica no centro do círculo formado. A figura 1.2 ilustra a execução do algoritmo para $n=4$. Na primeira rodada o pivô (vértice 1) joga com o time na primeira posição no círculo. Os outros times que estão em posições opostas no círculo se enfrentam. Na segunda rodada o pivô

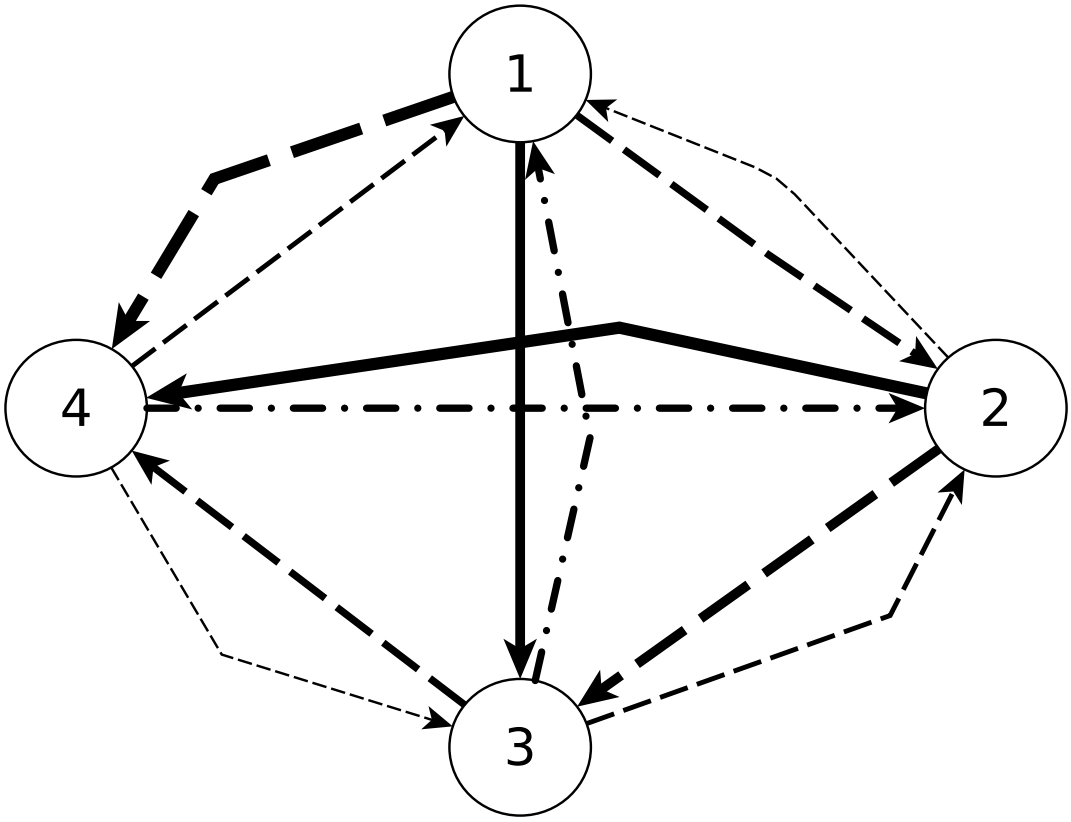

Figura 1.1: Exemplo de DRRT usando coloração de arestas. Cada conjunto de arestas com o mesmo tracejado forma um 1-fator.

enfrenta o próximo time do círculo, no sentido horário. Os outros times enfrentam seus opostos no círculo, e esse processo é repetido durante $n - 1$ iterações. Este algoritmo garante a atribuição das partidas de cada rodada. Uma prática comum é a execução do método do círculo considerando times anônimos. Após a agenda de jogos ser calculada, times reais são atribuídos, de forma aleatória ou considerando os custos na atribuição. Note que após a execução do método do círculo,

1.5 O *Traveling Tournament Problem*

Proposto em 2001 por Easton et al[7], o *Traveling Tournament Problem* (TTP), abstrai as principais características comuns aos problemas de Sports Scheduling e considera um cenário onde o custo das viagens realizadas pelos times são importantes. O TTP consiste em encontrar um DRRT onde o somatório dos custos das viagens realizadas pelos times é mínimo. Todos os times começam em suas respectivas casas e devem retornar à ela após a última rodada. As restrições *AtMost* (No máximo 3 rodadas fora/dentro de casa) e *NoRepeat* (a vs b não pode ser seguido por b

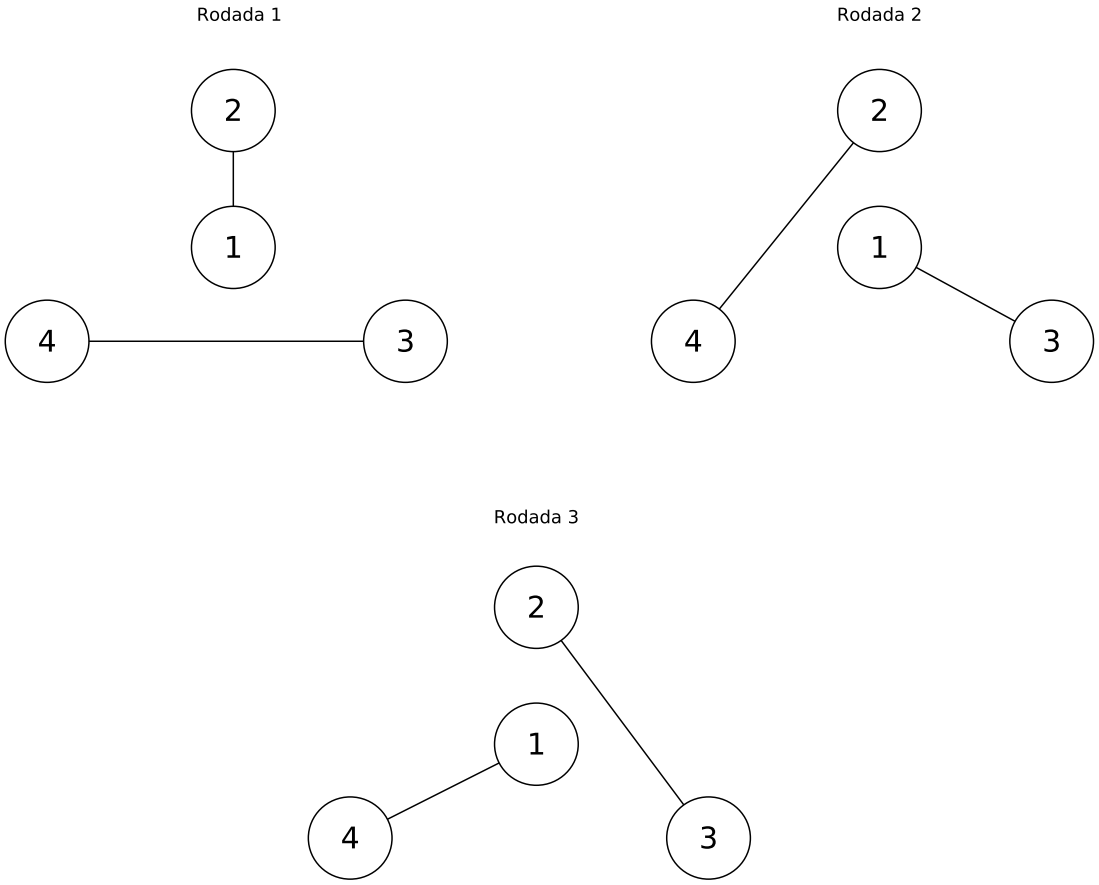

Figura 1.2: Método do Círculo. A cada iteração, um 1-fator do grafo é obtido.

vs a) também devem ser satisfeitas. Na sua versão clássica, os torneios não são espelhados. A variante espelhada, o *Mirrored Traveling Tournament Problem* (MTTP), foi proposta em 2007 por Ribeiro e Urrutia[8]. Em 2009 surge uma segunda variante do TTP: O TTPPV (*TTP with Predefined Venues*) [9]. Nesta variante, o padrão de mando de jogo de cada time é dado, e o objetivo é construir uma agenda de jogos que minimiza o custo total das viagens. Uma instância do TTP consiste num conjunto de times e os custos das viagens entre os estádios destes times. Ao longo destes 18 anos, o TTP vem sendo abordado por *constraint programming*, métodos exatos, heurísticos e aproximativos. Na url https://mat.tepper.cmu.edu/TOURN/ é possível realizar o *download* de instâncias, verificar soluções e encontrar os melhores limites superiores e inferiores cadastrados para várias instâncias. Entre as principais famílias de instâncias encontram-se: nl, con e circ. As instâncias com prefixo 'nl' baseiam-se na Liga Nacional de Basebol dos EUA. Nas instâncias com prefixo 'con', a distância entre qualquer par de times é igual a 1. Nesse caso, o TTP se torna o problema de minimizar o número de viagens realizadas. Note que encontrar as rotas individuais com menor custo de cada time pode ser visto como um problema do

caixeiro viajante (TSP). As instâncias 'circ' são elaboradas de forma que haja uma única solução trivial para o TSP. Desta forma, é possível analisar empiricamente a dificuldade do TTP considerando que o TSP subjacente é mais simples. Em [8], foi gerada a instância br24, com os times do campeonato brasileiro de 2003 e as distâncias entre as cidades dos mesmos. Na url supracitada ainda há outras famílias de instâncias: Super, NFL e Galaxy. A tabela 1.4 contém a instância nl4, com 4 times. A tabela 1.5 contém uma solução ótima para esta instância. Por coincidência esta solução também é ótima para o MTTP. Na última coluna consta o custo da rota executada por cada time. O custo total da solução é o somatório dos custos de todas as rotas.

Tabela 1.4: Instância nl4.

	1	2	3	4
1	0	745	665	929
2	745	0	80	337
3	665	80	0	380
4	929	337	380	0

Tabela 1.5: Exemplo de solução ótima para a instância nl4. O custo da solução é 8276.

T\R	1	2	3	4	5	6	Custo da Rota:
1	3	2	4	-3	-2	-4	2011
2	4	-1	-3	-4	1	3	2127
3	-1	4	2	1	-4	-2	2127
4	-2	-3	-1	2	3	1	2011

O primeiro trabalho sobre o TTP apresenta também um limite inferior, considerando as rotas individuais de cada time. Cada time deve visitar todos os outros uma única vez e deve jogar em casa $n-1$ vezes. Como mencionado anteriormente, esse problema é equivalente ao TSP, e podemos utilizar as técnicas deste para calcular a rota ótima a ser executada por um determinado time t. Esta rota ótima é chamada de **limite inferior independente (ILB)**. O somatório dos limites inferiores independentes é denominado **limite independente (IB)**. O IB é um limite inferior para o custo da solução do TTP. Infelizmente, o TSP é NP-Difícil, e o cálculo do limite independente pode ser custoso para instâncias maiores, o que torna o cálculo eficiente de um limite inferior um objeto de pesquisa em alguns trabalhos presentes na literatura. Em [10] encontra-se um exemplo do cálculo do IB, considerando o CVRP onde todos os veículos tem capacidade $C = U$, sendo U o máximo de rodadas consecutivas considerado na restrição *AtMost*.

Entre as contribuições originais desta dissertação constam dois algoritmos de *pricing* para o TTP, que são adaptações de algoritmos utilizados no CVRP. Estes

novos algoritmos de *pricing* podem ser utilizados para obter um limite mais fraco, porém de forma mais rápida que o IB. Outra contribuição original é a implementação de dois algoritmos de estabilização da geração de colunas aplicados ao TTP. O próximo capítulo contém um resumo dos principais trabalhos sobre o TTP.

Capítulo 2

Revisão Bibliográfica

No primeiro trabalho sobre o TTP os autores utilizaram *Constraint Programming* (CP) para geração de soluções viáveis. Uma formulação de programação inteira, a partir da enumeração exaustiva de todas as rotas dos times também é mencionada. Com estes dois métodos, os autores somente conseguiram encontrar soluções ótimas para as instâncias nl4 e nl6 ($n = 4$ e $n = 6$, respectivamente). Para a instância nl8, os autores conseguiram estabelecer um gap de dualidade de 5%.

No trabalho proposto por Benoist et al. [11], uma implementação de relaxação lagrangeana é combinada com um modelo de CP. O modelo CP é usado para guiar uma busca branch and bound. A relaxação lagrangeana fornece limites inferiores para os nós da árvore de busca. O sub-problema lagrangeano é um problema do caixeiro viajante, executado para cada time. Utilizando esta combinação, os autores foram capazes de encontrar soluções ótimas para as instâncias nl4 e nl6. Para a instância nl10, os autores conseguiram estabelecer um gap de dualidade de 17,7% em 24 horas. Encontrar a solução ótima para nl6 também levou cerca de um dia.

No trabalho proposto por Nemhauser et al.[12], os autores implementaram um algoritmo branch and price (ver Capítulo 3) com paralelismo. O problema de *pricing* é resolvido com CP, onde rotas são geradas para cada um dos times. Uma heurística primal baseada em CP, que busca agendamentos considerando todos os times, também é executada em paralelo à execução da busca na árvore. Um algoritmo polinomial para o problema de encontrar a rota de um único time também é apresentado, com a restrição de que o tamanho máximo de *home stands* e *road trips* seja igual a 2. Com esta abordagem, foi possível reduzir o gap da instância nl8 para 2%. Os testes nesta instância levaram mais de 100 horas.

No trabalho proposto por Anagnostopoulos et al. [13], os autores implementaram uma metaheurística *Simulated Annealing* (SA). As soluções iniciais são construídas aleatoriamente e o trabalho introduz cinco tipos de vizinhanças. O espaço de busca consiste de todos os torneios válidos, relaxando as restrições *NoRepeat* e *AtMost*. A função objetivo foi modificada, introduzindo uma penalidade dinâmica às violações

das restrições. O SA também utilizou a estratégia de reaquecimento, permitindo uma exploração maior do espaço de busca. O trabalho conseguiu melhorar os limites superiores para as instâncias de $n \in \{10, 12, 14, 16\}$. O tempo necessário para obter estes resultados foi de cerca de 11 horas para a instância nl10 e de mais de 90 horas para nl16.

No trabalho proposto por Di Gaspero et al. [14], os autores implementaram uma metaheurística Busca Tabu aplicada ao TTP. O método do círculo foi utilizado na construção de soluções iniciais aleatórias e parte das vizinhanças introduzidas em [13] também foram utilizadas. A função objetivo foi modificada com uma penalidade dinâmica, de forma similar ao trabalho proposto em [13]. Apesar de não conseguir melhorar os limites superiores já encontrados, a diferença entre as soluções obtidas no teste e os resultados de trabalhos anteriores foi de no máximo 6%. Os resultados foram obtidos com tempos, em média, 10 vezes menores que os obtidos em trabalhos anteriores.

No trabalho proposto por Lim et al. [15], os autores combinaram *Simulated Annealing* e *Hill Climbing* para encontrar soluções viáveis para o TTP. As soluções foram divididas em agendamentos válidos e HAPs. O espaço de busca das HAPS é explorado pelo *Simulated Annealing* e dos agendamentos é explorado pelo *Hill Climbing*. As soluções iniciais são construídas com o método proposto em [4]. No *Simulated Annealing*, foram consideradas as vizinhanças definidas em [13], e utilizadas em outros trabalhos. Neste trabalho os movimentos são realizados de forma a não violar as restrições *NoRepeat* e são adicionadas penalidades na função objetivo, referentes às restrições *AtMost*. No *Hill Climbing*, soluções iniciais são construídas atribuindo aleatoriamente os times aos HAPs e para diversificar a busca, K reinícios são utilizados. As soluções encontradas em um dos algoritmos também são passadas como entrada para o outro na tentativa de melhorar a qualidade das mesmas. A abordagem melhorou vários resultados das instâncias circulares em até 5% e obteve resultados cerca de 3% próximos aos melhores resultados para as instâncias da liga nacional.

No trabalho proposto por Ribeiro e Urrutia [8], foi utilizada uma versão híbrida das metaheurísticas GRASP e ILS. O problema considerado foi o MTTP (TTP espelhado). Uma nova vizinhança foi introduzida e as vizinhanças de trabalhos anteriores também foram utilizadas. Para a construção de soluções, foi utilizado o método do círculo, com uma modificação de forma a aproveitar os padrões gerados pelo método para obter soluções de menor custo. O padrão de mando de jogos é decidido de forma a manter a viabilidade, ou aleatoriamente quando não é possível ocorrer violação. Na heurística proposta, a fase de busca local do GRASP é substituída pela ILS. A busca local segue a idéia do VND utilizando as vizinhanças propostas. Ao contrário de trabalhos anteriores, apenas soluções viáveis são consi-

deradas e os autores utilizam uma busca tabu rápida para remover as inviabilidades. Neste trabalho foi introduzida a instância br24. Ao comparar as distâncias viajadas no campeonato brasileiro de 2003 com a melhor solução obtida, os autores constataram que a solução gerada pelo método proposto era 52% melhor que a implementada no campeonato. Além disso, com esta metaheurística híbrida foi possível melhorar o resultado de algumas instâncias circulares e obter soluções até 10% próximas dos melhores resultados conhecidos até 10 vezes mais rápido.

Em um segundo trabalho proposto por Di Gaspero et al. [16], foi utilizada uma implementação da busca tabu com uma vizinhança composta. O trabalho utilizou as cinco vizinhanças apresentadas em [13]. Para a busca tabu, foi considerada uma lista de tamanho aleatório e variável em um intervalo de valores. Novamente, uma função objetivo com penalidades dinâmicas, referentes às restrições *NoRepeat* e *AtMost*, foi utilizada. O algoritmo implementado obteve resultados próximos dos melhores conhecidos nas instâncias clássicas, com tempos entre 20 e 50% melhores. Nas instâncias CON, novos limites superiores foram obtidos. Na instância do campeonato brasileiro de tamanho $n = 24$, entretanto, não foi possível obter um resultado próximo ao melhor já conhecido.

No trabalho proposto por Lee et al. [17], um modelo de programação inteira é apresentado e uma nova abordagem de busca tabu é implementada. Utilizando o branch and bound tradicional somente nas instâncias de tamanho $n = 4$ foi possível encontrar a solução ótima. Devido a esse baixo desempenho, os autores optaram por implementar uma busca tabu. Os autores definem dois conceitos que permitem explorar de maneira mais eficiente as vizinhanças. As estruturas de vizinhança consideradas são as mesmas utilizadas em trabalhos anteriores, como [13]. As soluções iniciais foram construídas de forma aleatória utilizando *constraint programming*. Apesar de não conseguir soluções melhores que as já conhecidas, o tempo médio para encontrar soluções de qualidade próxima às melhores foi cerca de 50% menor em comparação com [15] e [13].

Em um segundo trabalho de Ribeiro e Urrutia [18], foram obtidos limites inferiores e superiores para a classe específica de instâncias constantes (onde todas as distâncias entre times são iguais a 1) do TTP espelhado. Neste caso específico o problema de minimizar as distâncias percorridas é equivalente à maximizar o número de quebras no agendamento. Neste trabalho as quebras são definidas como *home stands* e *road trips* de tamanho maior ou igual a 2. Para o TTP espelhado sem as restrições *AtMost*, foi apresentado um algoritmo polinomial que resolve o problema. Além disso, foram apresentados limites superiores e inferiores para o TTP espelhado com a restrição *AtMost*. Os limites são obtidos à partir da equivalência entre o TTP com instâncias constantes e o problema de maximizar o número de quebras. A partir desses limites, foi executada a metaheurística proposta pelos mesmos autores em

[8] e foi provada a otimalidade para as instâncias constantes com tamanho n até 16 (exceto $n = 14$).

Em um terceiro trabalho de Urrutia et al. [10], foi apresentado um algoritmo que calcula um limite inferior para o TTP. Este limite inferior é mais forte que o limite independente (IB), apresentado em [7]. O cálculo é realizado a partir do IB, do limite inferior obtido em [18] e também utiliza o ótimo conhecido de cada instância constante, ou o melhor limite inferior conhecido. O IB é obtido a partir do problema de roteamento de veículos capacitado (CVRP). Com este algoritmo foi possível melhorar o limite inferior de várias instâncias clássicas do problema, com tamanho até $n = 22$. Para algumas instâncias cujo limite inferior era desconhecido, o trabalho possibilitou definir o mesmo. O tempo necessário pra encontrar cada limite inferior não foi informado.

No trabalho proposto por Fujiwara et al. [19], foram apresentados um novo limite inferior e dois algoritmos construtivos para instâncias constantes do TTP. Em ambos os métodos construtivos, a idéia é criar várias tabelas parciais, e concatená-las para formar um torneio completo. O primeiro é uma modificação do método do círculo, com complexidade $O(n^2)$, que gera soluções melhores que as encontradas em [18]. No segundo método, foi utilizada programação inteira para gerar um torneio inicial, respeitando alguns teoremas sobre o número de quebras possíveis em um torneio [20] e considerando que o torneio simples obtido obedece a certas condições. Este algoritmo também retorna soluções melhores que as obtidas por Ribeiro e Urrutia [18]. Nos testes numéricos, foi possivel provar a otimalidade de soluções para instâncias de distâncias constantes com tamanho até $n = 50$. Alguns limites inferiores conhecidos também foram melhorados.

No trabalho proposto por Cheung [21], foi desenvolvido um algoritmo de duas fases para resolver as instâncias de tamanho $n = 8$ do TTP espelhado. A primeira fase consiste em enumerar todas as possíveis tabelas de jogos não redundantes, à partir de seis 1-fatorações. Os autores calcularam que existem 15.724.800 tabelas de jogos para $n = 8$ times. Na segunda fase, o padrão de mando de jogos ótimo é obtido à partir de *constraint programming* e programação inteira, considerando as tabelas encontradas na primeira fase. Foi constatado que existem somente 72 padrões de mando de jogo viáveis para uma instância com 8 times. O tempo de execução foi contabilizado para cada uma das seis 1-fatorações, e o tempo para encontrar a solução ótima variou entre aproximadamente 30 minutos e 30 horas.

No trabalho proposto por Melo et Al. [9], o TTPPV foi apresentado, juntamente com três formulações de programação inteira, além de estratégias de enumeração e de soluções aproximadas. Para instâncias maiores que $n = 16$, as formulações não conseguiram obter soluções viáveis. Os autores então apresentam mais duas formulações e estratégias para refinar a busca por soluções viáveis, onde os resultados

reportados indicam que foi possível encontrar soluções viáveis para instâncias com $n = 18$ e $n = 20$.

No trabalho proposto por Irnich [22], foi apresentado um algoritmo Branch and Price para o TTP. Uma nova formulação compacta foi apresentada, e também uma decomposição Dantzig-Wolfe baseada na mesma (A decomposição é explicada com detalhes no capítulo 3). O problema de *pricing* é resolvido como um problema de caminho mínimo com restrição de recursos (ESPPRC - *Elementary Shortest Path with Resource Constraints*). Várias estratégias de branching foram testadas, assim como estratégias para acelerar o problema de *pricing*. Entre os resultados obtidos, provou-se a otimalidade da instância nl8 e foram obtidos novos limites inferiores para as instâncias nl e circ com n até 12 e 8, respectivamente.

No trabalho proposto por Uthus et al. [23], um algoritmo DFS* foi apresentado para o TTP, combinando os algoritmos IDA* (uma variante do A*) e DFB&B (Depth First Branch & Bound), com possibilidade de execução da busca em paralelismo. Este método encontrou resultados similares aos encontrados em [22], porém em um tempo muito menor. Além disso a otimalidade da instância CIRC8 foi provada. Neste trabalho as instâncias do tipo Super foram introduzidas, com soluções ótimas para n até 10.

No trabalho de Misir et al. [24], os autores utilizaram o TTP para exemplificar um novo critério de busca em hiper heurísticas. Hiper heurísticas combinam elementos de metaheurísticas como a busca em vizinhanças com estratégias de aprendizado, tentando melhorar sua efetividade. O trabalho também considera as vizinhanças clássicas definidas em [13]. Este método encontrou soluções próximas às melhores soluções conhecidas para as instâncias NL, e conseguiu alcançar os melhores resultados já conhecidos para as instâncias Super de tamanho n até 8. Nas instâncias Super de tamanho maior que $n = 8$, limites superiores foram obtidos em muito menos tempo (quase 10 vezes menos tempo na instância Super14) que trabalhos anteriores.

No trabalho proposto por Miyashiro et al. [25], foi apresentado um limite inferior e um algoritmo aproximativo com fator constante menor que $2+\frac{3}{4}$. O limite inferior, é mais fraco que o limite inferior independente [7] porém mais fácil de computar. Foi demonstrado que uma solução ótima do TTP é maior ou igual à $\frac{2S}{3}$, onde S é o somatório de todas as distâncias entre os n times. A partir deste limite inferior também foi demonstrado que para o valor z^* de uma solução ótima, nenhuma solução viável z é maior que $3z^*$. O algoritmo aproximativo é obtido a partir do método do círculo modificado (MCM), adaptado de [19]. Uma demonstração foi apresentada, garantindo a taxa de aproximação do algoritmo proposto (para instâncias com $n \geq 4$).

No trabalho proposto por Thielen et al. [26], foi demonstrado que o TTP é

de fato NP-Difícil. A demonstração consiste em obter uma redução do TTP para o problema 3-SAT onde o número de cláusulas é múltiplo de 6 e o número de ocorrências de uma variável x_i é igual ao número de ocorrências da variável $\bar{x}_i$. Os autores também demonstraram que essa variante do 3-SAT também é NP-Hard.

No segundo trabalho de Uthus et al. [27], um algoritmo IDA* é apresentado. O trabalho é uma evolução do que foi apresentado em [23], incorporando novos conceitos. O algoritmo foi paralelizado podendo ser utilizado em máquinas com memória distribuída ou compartilhada. Com esta abordagem foi possível resolver as instâncias NL e CIRC de tamanho $n = 10$ e encontrar soluções ótimas já conhecidas para outras instâncias. O conjunto de instâncias Galaxy, baseadas nas distâncias tridimensionais entre os planetas, foi introduzido, com soluções ótimas para instâncias de até 10 times.

No trabalho de Goerigk et al. [28], um algoritmo de busca tabu foi combinado à modelos de programação inteira (PI). Na busca tabu, considera-se as vizinhanças clássicas e a função objetivo modificada utilizada em outros trabalhos anteriores. A solução encontrada é submetida a dois modelos de PI, um que visa otimizar o padrão de mandos de jogo e outro que tenta otimizar o agendamento das partidas. A execução dos modelos se alterna até que não seja possível melhorar a solução. Após isso a melhor solução é submetida à busca tabu novamente e o ciclo se repete até que nenhum dos métodos consiga melhorar a solução. O método conseguiu melhorar os limites superiores para mais da metade das instâncias testadas (família Galaxy).

No trabalho de Westphal et al. [29], um algoritmo aproximativo foi apresentado com um fator de 5,875 quando o comprimento máximo de *home stands* e *road trips* for $k \geq 4$. O algoritmo consiste numa modificação do método do círculo, utilizando uma tour do TSP para a atribuição dos times. A partir dos torneios construídos, os autores implementaram uma busca local, a fim de testar o desempenho do algoritmo de construção. A combinação do algoritmo aproximativo e busca local conseguiu obter limites superiores para as instâncias Galaxy com tamanho até $n = 40$.

No trabalho de Goerigk et al. [30], uma estratégia híbrida foi implementada, envolvendo *packings* P_3 mínimos, busca tabu e um pós processamento com um modelo de programação inteira. Seja G um grafo, um *packing* P_3 é um conjunto de caminhos disjuntos, cada um com 3 vértices, que cobrem os vértices de G. Este conceito é utilizado para construir rotas com *road trips* de baixo custo. A busca tabu e o modelo de PI refinam as soluções construídas. Com esta abordagem, foi possível melhorar os limites superiores das instâncias NFL com tamanho 28, e das instâncias Galaxy com até 40 times.

Capítulo 3

Método Proposto

Este trabalho consiste na avaliação de limites para o TTP através de um algoritmo de geração de colunas, com novos algoritmos de *pricing* adaptados do problema de roteamento de veículos capacitado (CVRP). A escolha pela abordagem de geração de colunas se deu pelo fato de que há poucos trabalhos na literatura que visam resolver de forma exata o TTP, e novas contribuições na geração de colunas podem levar a descoberta de soluções exatas de forma eficiente. Com a implementação destes algoritmos de *pricing* esperava-se obter limites inferiores para as instâncias maiores do TTP em menor tempo que as abordagens anteriores. A formulação do problema mestre foi adaptada de [22] assim como os algoritmos de *pricing* lá utilizados, para fim de comparação. Esta formulação foi escolhida pois com ela pudemos adaptar conceitos do CVRP e avaliar sua eficiência no TTP. Dois procedimentos para estabilização da geração de colunas também foram implementados além de uma heurística para o problema de *pricing* e uma metaheurística para o TTP com o objetivo de fornecer um *warm start* para a geração de colunas. Nas próximas seções e subseções deste capítulo constam as abordagens implementadas em cada um dos algoritmos. No capítulo 4 constam os resultados dos testes realizados nos algoritmos. No capítulo 5 apresentamos as conclusões e possíveis melhorias para trabalhos futuros.

3.1 Geração de Colunas

O algoritmo de geração colunas é empregado para resolver problemas de programação linear (PPL) com um número muito grande de variáveis [31]. Considere o PPL P(descrito nas restrições 3.1, 3.2) como sendo tal problema. No simplex clássico, a decisão de qual variável entrará na base em uma determinada iteração é realizada avaliando os custos reduzidos de todas as variáveis não básicas. Como P possui um número muito grande de variáveis, tal abordagem não é eficiente e na geração de colunas a escolha é realizada de forma implícita, resolvendo um problema

auxiliar. O algoritmo de geração de colunas inicia considerando somente um pequeno subconjunto das variáveis do problema P, o suficiente para obter uma base primal viável. Chamaremos este problema reduzido de $\tilde{P}$. Ao resolver $\tilde{P}$, uma solução dual $\tilde{\pi}$ é obtida. Para adicionar uma nova variável à $\tilde{P}$ o seguinte subproblema, denominado problema de *pricing*, deve ser resolvido: $\tilde{c} = min_{x \in N}\{(c_x - \tilde{\pi}A)x\}$, onde N é o conjunto das variáveis que não estão em $\tilde{P}$. Se $\tilde{c} < 0$ então x é adicionada a $\tilde{P}$, que é então reotimizado, gerando uma nova solução $\tilde{\pi}$. Se $\tilde{c} \geq 0$, então a última solução encontrada em $\tilde{P}$ é ótima para P e o algoritmo termina.

$$\begin{aligned} \min \quad & c^T x \\ \text{s.a.} \quad & Ax = b \qquad (3.1) \\ & x \geq 0, x \in \mathbb{R}^n \qquad (3.2) \end{aligned}$$

Outro conceito relacionado a geração de colunas é a decomposição de Dantzig-Wolfe. Esta decomposição é utilizada para reformular problemas de programação inteira (PPI). Seja o problema (P_2): $z = min\{\sum_{k=1}^{K} c^k x^k : \sum_{k=1}^{K} A^k x^k = b, x^k \in X^k, k = 1, .., K\}$, onde $X^k = \{x^k \in \mathbb{Z}^{n_k} : D^k x^k \leq d\}$ para $k = 1, ..., K$ são subconjuntos da região viável. Considerando que todos os X^k são limitados, os elementos de X^k podem ser reescritos a partir da combinação convexa de todos os N_k elementos de X^k:

$$X^k = \{x^k : x^k = \sum_{i=1}^{N_k} \lambda_{k,i} x^{k,i}, \sum_{i=1}^{N^k} \lambda_{k,i} = 1, \lambda_{k,i} \in \{0,1\}, i = 1, ..., N_k\}$$

Onde $x^{k,i}$ representa um elemento i do conjunto X^k. Considerando essa representação de x^k, o problema P_2 é reescrito como o problema (PM), denominado problema mestre. Este problema mestre tem uma quantidade enorme de variáveis (geralmente, exponencial em relação a entrada do problema) e a geração de colunas pode ser aplicada, iniciando o problema mestre restrito (PMR) com pelo menos uma variável $\lambda_{k,i}$ para cada X^k. Ao resolver o PMR, uma solução dual $(\tilde{\pi}, \tilde{\mu})$ é obtida, com $\tilde{\pi}$ referente às restrições (3.3) que agregam os conjuntos X^k e $\tilde{\mu}$ referente às chamadas restrições de convexidade (3.4). A cada iteração K problemas de *pricing* $\tilde{c}_k = min\{(c^k - \tilde{\pi}A^k)x - \tilde{\mu}_k\}$ devem ser resolvidos e as variáveis encontradas que possuem $\tilde{c}_k$ negativos são adicionadas ao PMR. A relaxação linear de PM é obtida quando todos os $\tilde{c}_k$ encontrados são maiores ou iguais a zero.

$$
\begin{aligned}
(PM)\min \quad & \sum_{k=1}^{K}\sum_{i=1}^{N_k}(c^k x^{k,i})\lambda_{k,i} & & \\
\text{s.a.} \quad & \sum_{k=1}^{K}\sum_{i=1}^{N_k}(A^k x^{k,i})\lambda_{k,t} = b & & (3.3)\\
& \sum_{i=1}^{N_k}\lambda_{k,i} = 1 & k = 1,...,K & (3.4)\\
& \lambda_{k,i} \in \{0,1\} & i = 1,...,N_k; k = 1,...,K & (3.5)
\end{aligned}
$$

3.1.1 Geração de Colunas no TTP

A mesma formulação utilizada em [22] é considerada, onde o caminho percorrido por cada time é modelado em um grafo discretizado por rodadas. Seja $T = \{0, 1, ..., \bar{n}\}$ o conjunto dos times e $S = \{1, 2, ..., 2\bar{n}\}$ o conjunto das rodadas, onde $\bar{n} = n - 1$. Seja também $N^t = (V^t, A^t)$ um grafo direcionado, referente ao time $t \in T$. O conjunto de vértices é dado por $V^t = \{v_{is} : i \in T, s \in S\} \cup \{v_{t0}, v_{t,2\bar{n}+1}\}$, onde v_{is} denota que o time t está no estádio do time i na rodada s e $\{v_{t0}, v_{t,2\bar{n}+1}\}$ são vértices artificiais que indicam que o time t começa o torneio em casa (rodada 0) e deve retornar pra casa após o fim do torneio (rodada $2\bar{n} + 1$). O conjunto de arestas é dado por $A^t = \{(t, j, 0) : j \in T\} \cup \{(i, t, 2\bar{n}) : i \in T\} \cup \{(i, j, s) : i, j \in T(i \neq j \text{ ou } i = j = t), s \in S \setminus \{2\bar{n}\}\}$, onde (i, j, s) denota que o time t sai do estádio do time i na rodada s para jogar no estádio do time j na rodada $s + 1$. Em outras palavras, a aresta (i, j, s) liga o vértice v_{is} ao vértice $v_{j,s+1}$. A figura 3.1 mostra um exemplo do grafo N^t para o time $t = 0$ em uma instância com quatro times. Note que a informação sobre o adversário i de t, quando t joga em casa, está implícita e é dada pelo grafo do time i.

O modelo (3.6) - (3.11) considera tal grafo. O conjunto $B^t = \{(i, j, s) \in A^t : (i = j = t, s \neq 0, 2\bar{n}) \text{ ou } i \neq j, i \neq t, j \neq t\} \subset A^t\}$ corresponde aos arcos que induzem um *break*: arcos que indicam que o time t se manteve em casa ou se manteve viajando nas rodadas s e $s+1$. Por exemplo, se em uma rota realizada pelo time $t = 0$ a aresta $(0, 0, 4)$ foi percorrida, então ocorreu um *break* pois a utilização da aresta indica que t se manteve em casa na quarta e na quinta rodada. De forma análoga, na aresta $(1, 2, 1)$ também ocorre um *break* pois o time t se manteve fora de casa nas duas primeiras rodadas do torneio. As rodadas artificiais $s = 0$ e $s = 2\bar{n}$ não induzem um *break* pois são as viagens de inicio e fim do torneio, independentes do agendamento. Assim como na maioria dos trabalhos da literatura, considera-se $U = 3$ como a quantidade máxima de rodadas seguidas que um time pode permanecer em casa ou em viagem à outros estádios. A variável x^t_{ijs} indica que a aresta (i, j, s) é selecionada

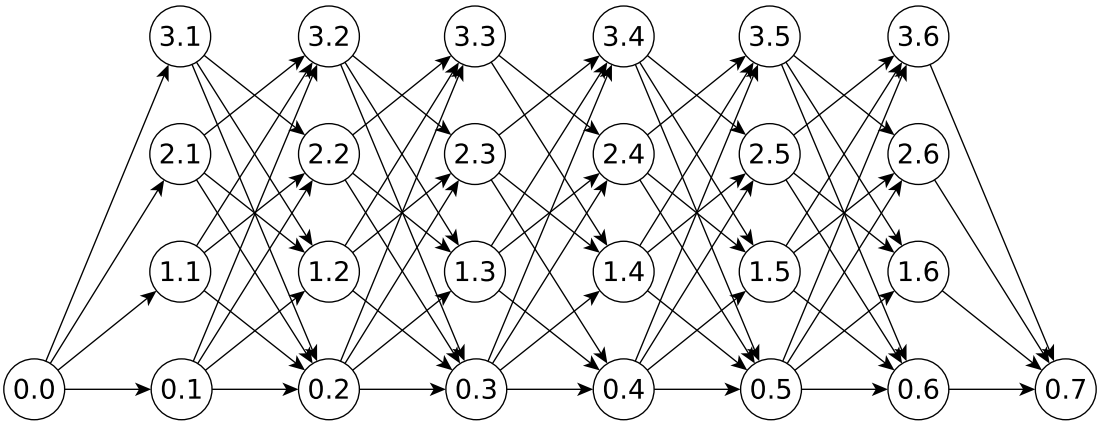

Figura 3.1: N^t referente ao time 0 em uma instância com $n = 4$. Os vértices v_{is} estão indicados por $i.s$ nos vértices da figura.

para a rota percorrida pelo time t, implicando que t deve jogar no estádio do time i na rodada s e no estádio do time j na rodada $s+1$. O coeficiente d_{ij} indica o custo da viagem de i para j. As restrições (3.6) tratam da conservação de fluxo, garantindo que todo time que entra em um estádio deve sair dele, após a partida. As restrições (3.7) garantem que todos os times devem visitar os estádios de cada um de seus oponentes exatamente uma vez. As restrições *AtMost* são garantidas em (3.8) e as restrições (3.9) agregam as rotas de todos os times, garantindo que em cada rodada s, cada time t ou joga fora de casa contra algum time i(primeiro somatório) ou joga em casa contra algum time t' (segundo somatório). Por fim, as restrições *NoRepeat* constam em (3.10). Para implementar as restrições (*NoRepeat*) basta definir que $x^t_{tt's} + x^{t'}_{tt's} \leq 1$, porém ao inverter o papel de t e t' na restrição verifica-se que tais arestas também são incompatíveis na solução, uma vez que isso levaria ou à um desencontro entre t e t', ou a um agendamento onde t ou t' jogam duas partidas em

uma rodada. A restrição é então ampliada para a versão considerada no modelo.

$$\min \quad \sum_{t\in T}\sum_{(i,j,s)\in A^t} d_{ij}x^t_{ijs}$$

$$\text{s.a.} \quad \sum_{i:(i,j,s-1)\in A^t} x^t_{ij,s-1} - \sum_{i:(j,i,s)\in A^t} x^t_{jis} = 0 \qquad \forall t, j \in T, s \in S \qquad (3.6)$$

$$\sum_{s\in S}\sum_{j:(i,j,s)\in A^t} x^t_{ijs} = 1 \qquad \forall t \in T, i \in T \setminus \{t\} \qquad (3.7)$$

$$\sum_{u=0}^{U-1}\sum_{(i,j,s+u)\in B^t} x^t_{ij,s+u} \leq U - 1 \qquad \forall t \in T, s \in S : s \leq 2\bar{n} - U \qquad (3.8)$$

$$\sum_{i\in T\setminus\{t\}}\sum_{j:(i,j,s)\in A^t} x^t_{ijs} + \sum_{t'\in T\setminus\{t\}}\sum_{j:(t,j,s)\in A^{t'}} x^{t'}_{tjs} = 1 \qquad \forall t \in T, s \in S \qquad (3.9)$$

$$x^t_{tt's} + x^{t'}_{tt's} + x^{t'}_{t'ts} + x^t_{t'ts} \leq 1 \qquad \forall t, t' \in T, s \in S : t < t', s \neq 2\bar{n} \qquad (3.10)$$

$$x^t_{ijs} \in \{0, 1\} \qquad \forall t \in T, (i, j, s) \in A^t \qquad (3.11)$$

Uma decomposição de Dantzig-Wolfe deste modelo, as restrições 3.6 - 3.8 e 3.11 definem o domínio dos problemas de *pricing* de cada time t. As restrições 3.9 e 3.10 (*NoRepeat*) envolvem mais de um time e serão adaptadas no problema mestre. Esta decomposição consta em 3.12 - 3.15. A variável binária λ^t_p indica se a rota $p \in P^t$ está presente na solução. O conjunto P^t contém as rotas viáveis do time t, isto é: $P^t = \{p = (x^t_{tj0}, x^t_{ij1}, .., x^t_{it,2\bar{n}}) : p$ não viola as restrições 3.6 - 3.8 e 3.11$\}$. O custo de uma rota $p \in P^t$ é dado por $c_p = \sum_{(i,j,s)\in p} d_{ij}x^t_{ijs}$. O subconjunto $P^t_{t's} \subset P_t$ contém as rotas onde t joga fora de casa contra t' na rodada s. Por sua vez, $P^t_{ijs} \subset P^t$ representa o subconjunto das rotas que contém o arco $(i, j, s) \in A^t$. As restrições 3.12 são similares à 3.9 e garantem que as rotas escolhidas na solução formam um DRRT. Note que a restrição 3.12 equivale à restrição 3.8 porém a igualdade foi substituída por maior ou igual para evitar variáveis duais irrestritas. As restrições 3.13 garantem que somente uma rota por time é selecionada. As restrições *NoRepeat* constam em 3.14. Na próxima seção veremos como adicionar variáveis dinâmicamente ao referido problema mestre.

$$\min \quad \sum_{t \in T} \sum_{p \in P^t} c_p \lambda_p^t$$

$$\text{s.a.} \quad \sum_{t' \in T \setminus \{t\}} \sum_{p \in P_{t's}^t} \lambda_p^t + \sum_{t' \in T \setminus \{t\}} \sum_{p \in P_{ts}^{t'}} \lambda_p^{t'} \geq 1 \qquad \forall t \in T, s \in S \quad (3.12)$$

$$\sum_{p \in P^t} \lambda_p^t = 1 \qquad \forall t \in T \quad (3.13)$$

$$\sum_{p \in (P_{tt's}^t \cup P_{t'ts}^t)} \lambda_p^t + \sum_{p \in (P_{tt's}^{t'} \cup P_{t'ts}^{t'})} \lambda_p^{t'} \leq 1 \quad \forall t, t' \in T, s \in S \setminus \{2\bar{n}\} : t \neq t' \quad (3.14)$$

$$\lambda_p^t \in \{0, 1\} \qquad \forall t \in T, p \in P^t \quad (3.15)$$

3.2 *Pricing*

O problema de *pricing* para a formulação 3.12 - 3.15 é modelado como uma variante do problema do caminho mínimo com restrição de recursos (*ESPPRC - Elementary Shortest Path Problem with Resource Constraints*). O objetivo é encontrar um caminho mínimo p, para um time t, de v_{t0} à $v_{t,2\bar{n}+1}$, onde o time t seja visitado $n-1$ vezes e cada outro time seja visitado exatamente uma vez, respeitando também a restrição *AtMost*. As restrições *NoRepeat* consideram rotas de dois times distintos e portanto não são consideradas durante o *pricing*. Este problema é NP-Difícil e em trabalhos anteriores a resolução do problema na geração de colunas só foi viável até instâncias de tamanho $n = 10$. No presente trabalho, vários algoritmos de *pricing* foram avaliados, entre eles um modelo de programação inteira para o *pricing* e um algoritmo de *labeling* bidirecional, ambos apresentados em [22]. Como contribuições originais, este trabalho apresenta uma heurística de *pricing* e dois algoritmos que relaxam a restrição de elementaridade nos caminhos encontrados: *q-routes* e *ng-routes*. A relaxação tem como inspiração abordagens similares utilizadas nos problemas de *pricing* do roteamento de veículos capacitado (CVRP - Ver [32, 33]). Em todos os algoritmos de *pricing*, o custo de cada aresta é calculado da seguinte forma: Sejam π_{ts}, μ_t e β_{ijs} as variáveis duais referentes às restrições 3.12, 3.13 e 3.14 respectivamente. O custo $\tilde{c}_{ijs}^t$ referente à viagem realizada pelo time t, do estádio do time i ao estádio do time j na rodada s é dado por:

$$\tilde{c}_{ijs}^t = d_{ij} - \begin{cases} \pi_{ts} + \pi_{is} \text{ se } i \neq t \\ 0, \text{ caso contrário} \end{cases} - \begin{cases} \beta_{ijs} \text{ se } i < j, s < 2\bar{n} \\ \beta_{jis} \text{ se } i > j, s < 2\bar{n} \\ 0, \text{ caso contrário} \end{cases} \quad (3.16)$$

e o custo reduzido da variável λ_p^t é dado por $\tilde{c}_p^t = \sum_{(i,j,s) \in p} \tilde{c}_{ijs}^t - \mu_t$. Note que no TTP espelhado a restrição *NoRepeat* nunca é violada pois não há como ocorrer dois

jogos entre a e b em rodadas consecutivas. As variáveis β_{ijs} não são consideradas neste caso.

3.2.1 *Pricing* via Programação Inteira

Em [22], o modelo 3.17 - 3.20 é introduzido. Este modelo é executado para cada time t. A variável binária x_{ijs} indica se a aresta (i, j, s), com seu custo reduzido $\tilde{c}^t_{ijs}$, está na solução. As restrições 3.17 garantem a conservação de fluxo e as restrições 3.18 garantem que o time t visita todos os outros times somente uma vez. As restrições *AtMost* são garantidas em 3.19. Se o custo reduzido total $z - \mu_t$ for negativo então uma variável do problema mestre referente a rota representada pela rota encontrada deve ser adicionada ao problema mestre.

$$\min \quad z = \sum_{(i,j,s)\in A^t} \tilde{c}^t_{ijs} x_{ijs}$$

$$\text{s.a.} \quad \sum_{i:(i,j,s-1)\in A^t} x_{ij,s-1} - \sum_{i:(j,i,s)\in A^t} x_{jis} = 0 \qquad \forall j \in T, s \in S \qquad (3.17)$$

$$\sum_{s\in S} \sum_{j:(i,j,s)\in A^t} x_{ijs} = 1 \qquad \forall i \in T \setminus \{t\} \qquad (3.18)$$

$$\sum_{u=0}^{U-1} \sum_{(i,j,s+u)\in B^t} x_{ij,s+u} \leq U - 1 \qquad \forall s \in S : s \leq 2\bar{n} - U \qquad (3.19)$$

$$x_{ijs} \in \{0, 1\} \qquad \forall (i, j, s) \in A^t \qquad (3.20)$$

3.2.2 *Pricing* Bidirecional

A principal abordagem do trabalho apresentado em [22] consiste num algoritmo de *pricing* que transforma o ESPPRC em um problema de caminho mínimo num grafo estendido. Este grafo estendido é gerado a partir de extensões de caminhos partindo do nó origem v_{t0}. Uma *label* $l = (i, s, b, N)$ representa um caminho no grafo estendido, através de quatro componentes:

- $i \in T$: Time sendo atualmente visitado no caminho. As *labels* são estendidas, quando possível, aos vizinhos do vértice v_{is} referente ao time i.
- $s \in S \cup \{0, 2n-1\}$: Indica a rodada s referente ao vértice v_{is} atual do caminho.
- $b \in \{0, 1, 2\}$: Contador de *breaks*. Mantém a informação de quantas rodadas seguidas o time t está jogando em casa ou viajando. Sempre que ocorre uma extensão do vértice v_{ts} para o vértice $v_{t,s+1}$ (o time t joga em casa em ambas as rodadas s e $s+1$) o contador de *breaks* b é incrementado. Da mesma forma, b é incrementado sempre que ocorre uma extensão do vértice v_{is} para o vértice

$v_{j,s+1}$, com i e j ambos diferentes de t (o time t joga fora de casa nas rodadas s e $s+1$). Nos dois casos, as extensões de *labels* que tornariam $b = 3$ são proibidas, devido a restrição *AtMost*. Por fim, se ocorre uma extensão do vértice v_{ts} ao vértice $v_{i,s+1}$ ou do vértice $v_{i,s}$ ao vértice $v_{t,s+1}$, b é reiniciado em 0. A extensão partindo da *label* origem e para a *label* destino não incrementam o contador de *breaks* pois as rodadas artificiais 0 e $2\bar{n}$ não são consideradas no agendamento final.

- $N \in 2^{T\setminus\{t\}}$: Conjunto de times já visitados no caminho representado pela *label*. Todo time $i \neq t$ visitado durante a extensão de um caminho deve ser adicionado ao conjunto N da respectiva *label*. As extensões à vertices de times já visitados são proibidas.

O custo da aresta que liga a *label* (i, s, b, N) à *label* $(j, s+1, b', N')$ é dado pelo custo reduzido $\tilde{c}^t_{ijs}$ definido em 3.16. Após o processo de extensão, o caminho mínimo da *label* $(t, 0, 0, \emptyset)$ à *label* $(t, 2\bar{n}, 0, T)$ encontrado é equivalente à solução do ESPPRC, que é o problema de *pricing* do problema mestre utilizado. Como a quantidade de *labels* sendo estendidas durante a execução do método é exponencial em relação à n, uma boa estratégia é realizar a extensão de forma bidirecional (conforme discutido em [34]). No TTP, esta abordagem levou a uma redução de até 30% na quantidade de *labels* analisadas. A extensão bidirecional pode ser realizada pois as regras de extensão podem ser utilizadas em qualquer *label* $u = (i, s, b, N)$ sendo estendida de forma progressiva no arco (i, j, s) e de forma regressiva no arco $(j, i, 2\bar{n} - s)$. A única diferença reside na possibilidade dos custos $\tilde{c}^t_{ijs}$ e $\tilde{c}^t_{ji,2\bar{n}-s}$ serem diferentes. A ideia é estender as *labels* progredindo a partir da origem v_{t0} com a *label* $o = (t, 0, 0, \emptyset)$ até as *labels* na rodada $\bar{n}$ (o "meio"do caminho), considerando também as respectivas extensões regressivas. Após a realização das extensões, uma fase final de concatenação é executada para analisar as junções válidas entre as *labels* da rodada $\bar{n}$. Uma *label* $u = (i, \bar{n}, b, N)$ pode ser concatenada à outra *label* $v = (j, \bar{n}, b', N')$ através da aresta $(i, j, \bar{n})$ se e somente se: (i)$N \cap N' = \emptyset$ e $N \cup N' = T \setminus \{t\}$, (ii)$(i, j, \bar{n}) \notin B^t$ ou $b + b' + 1 \leq U - 1 = 2$. Seja $\tilde{c}_{fw}$ o custo da extensão progressiva da origem até a *label* u e $\tilde{c}_{bw}$ o custo da extensão regressiva até a *label* v. O custo total do caminho obtido é dado por $\tilde{c}_{fw} + \tilde{c}^t_{ij\bar{n}} + \tilde{c}_{bw}$. No algoritmo 1, consta um resumo do *pricing* bidirecional.

Pricing Bidirecional e Redução de Simetria

Além do algoritmo de *pricing* bidirecional, em [22] é proposto um algoritmo para eliminar soluções quando há simetria nas distâncias da instância. A redução é realizada da seguinte forma: na fase de concatenação dos caminhos, à cada uma das K *labels* referentes à rodada $n-1$ (o meio do caminho) é atribuído um número k.

Algoritmo 1 *Pricing Label* Bidirecional: Determina o caminho mínimo para o time t

1. Inicie a fila de *labels* Q.
2. Inicie um mapa P de predecessores e um mapa S de sucessores.
3. Inicie os mapas C_{fw} e C_{bw} de custo para as duas direções.
4. Adicione a *label* $o = (t, 0, 0, \emptyset)$ à Q.
5. $C_{fw}(o) \leftarrow C_{bw}(o) \leftarrow 0$
6. $P(o) \leftarrow nil$, $S(o) \leftarrow nil$
7. **enquanto** a fila Q não está vazia **faça**
8. Remova a *label* $u = (i, s, b, N)$ de Q_{fw}
9. **se** $s < \bar{n}$ **então**
10. **para** Cada vizinho $v_{j,s+1}$ de v_{is} **faça**
11. **se** $i \neq t$ e $j \neq t$ e $j \notin N$ e $b < 2$ **então**
12. Crie a *label* $v = (j, s+1, b+1, N \cup \{j\})$
13. **senão se** $i = t$ e $j = t$ e $b < 2$ **então**
14. Crie a *label* $v = (j, s+1, b+1, N)$
15. **senão se** $i = t$ e $j \neq t$ e $j \notin N$ **então**
16. Crie a *label* $v = (j, s+1, 0, N \cup \{j\})$
17. **senão se** $i \neq t$ e $j = t$ **então**
18. Crie a *label* $v = (j, s+1, 0, N)$
19. **senão**
20. Extensão inválida. Continue para o próximo vizinho.
21. **fim se**
22. $c_{fw} \leftarrow \tilde{c}^t_{ijs}$
23. $c_{bw} \leftarrow \tilde{c}^t_{ij,2\bar{n}-s}$
24. **se** $C_{fw}(u) + c_{fw} < C_{fw}(v)$ **então**
25. $C_{fw}(v) \leftarrow C_{fw}(u) + c_{fw}$
26. $P(v) \leftarrow u$
27. **fim se**
28. **se** $C_{bw}(u) + c_{bw} < C_{bw}(v)$ **então**
29. $C_{bw}(v) \leftarrow C_{bw}(u) + c_{bw}$
30. $S(v) \leftarrow u$
31. **fim se**
32. Adicione v à fila Q
33. **fim para**
34. **fim se**
35. **fim enquanto**
36. $z \leftarrow \infty$
37. $menor_u \leftarrow menor_v \leftarrow nil$
38. **para** Cada par de *labels* $u = (i, \bar{n}, b, N), v = (j, \bar{n}, b', N')$ **faça**
39. **se** $C_{fw}(u) + \tilde{c}^t_{ij\bar{n}} + C_{bw}(v) < z$ e a concatenação das *labels* u e v é viável **então**
40. $z \leftarrow C_{fw}(u) + \tilde{c}^t_{ij\bar{n}} + C_{bw}(v)$
41. $menor_u \leftarrow u$, $menor_v \leftarrow v$
42. **fim se**
43. **fim para**
44. Construa o caminho p, concatenando os caminhos $P(menor_u)$ e $S(menor_v)$
45. Retorne o caminho mínimo p e o seu custo reduzido $z - \mu_t$

Um caminho progressivo l só é conectado a um caminho regressivo m se $k(l) \leq k(m)$. Acredita-se que esta redução de simetria depende da implementação, uma vez que a cada iteração não há como garantir qual sequência de times será cortada: A cada iteração um caminho pode receber uma numeração diferente, dependendo de como as *labels* são armazenadas na memória. Ao implementar esse procedimento, foi obtido um valor da relaxação diferente do reportado naquele trabalho. Além disso a relaxação obtida não era consistente com o *pricing* via modelo também proposto no artigo. Acredita-se que a diferença reside no fato do procedimento não fixar um time em específico e sim *labels* que representam caminhos e que podem variar ao longo das iterações da geração de colunas. Além disso no *pricing* via modelo proposto a redução de simetria não está presente, o que dificultaria ainda mais as comparações. Por esta discrepância nos valores da relaxação optou-se por não utilizar esta redução de simetria.

3.2.3 Grafo *q-routes/ng-routes*

Os algoritmos de *pricing q-routes* e *ng-routes* utilizam o mesmo grafo, que será definido nesta subseção. O grafo é discretizado por rodadas de forma semelhante ao utilizado nas seções anteriores. Cada vértice é representado por uma tripla (i, s, b): i representa o time referente ao vértice, s representa a rodada na qual o time i é visitado, b representa o contador de *breaks*: a quantas rodadas t está jogando fora de casa se $i \neq t$ e a quantas rodadas t está jogando em casa se $t = i$. Considerando uma instância com $n = 4$, a figura 3.2 representa o grafo do time 0. As arestas só ocorrem entre os vértices se o incremento nos indicadores de rodada s e nos indicadores de *break* b forem corretos. Seja o vértice $v = (i, s, b)$, $i \neq t$. Se $b = 1, 2$, v é ligado aos vértices $(j, s + 1, b + 1)$ e $(t, s + 1, 1)$, $j \neq t$. Se $b = 3$, v só pode ser ligado ao vértice $(t, s + 1, 1)$. De forma análoga, seja $v = (t, s, b)$. Se $b = 1, 2$, v é ligado aos vértices $(j, s + 1, 1)$ e $(t, s + 1, b + 1)$, com $j \neq t$. Caso $b = 3$, v é ligado somente aos vértices $(j, s+1, 1)$, $j \neq t$. O uso dos contadores b garantem a restrição *AtMost*, uma vez que um vértice (i, s, b) com $b > 3$ não existe no grafo. Como as rodadas 0 e $2\bar{n}$ não contribuem para a contagem de *breaks*, na primeira rodada o vértice $(t, 0, 0)$ é conectado à todos os vértices $(i, 1, 1)$ da primeira rodada, com $i \in T$. Na última rodada, todos os vértices são conectados à $(t, 2\bar{n}, 0)$ independente de seus respectivos contadores b. Como os problemas de *pricing* são executados individualmente para cada time e o grafo não indica qual o time rival sendo enfrentado a cada rodada não há como garantir as restrições *NoRepeat*. Tal fato não é um problema uma vez que as restrições *NoRepeat* são garantidas no problema mestre.

O custo da aresta entre (i, s, b) e $(j, s + 1, b')$ é dado por $\tilde{c}^t_{ijs}$ 3.16. Assim como em [32], *q-routes* e *ng-routes* utilizam uma matriz M, preenchida via programação

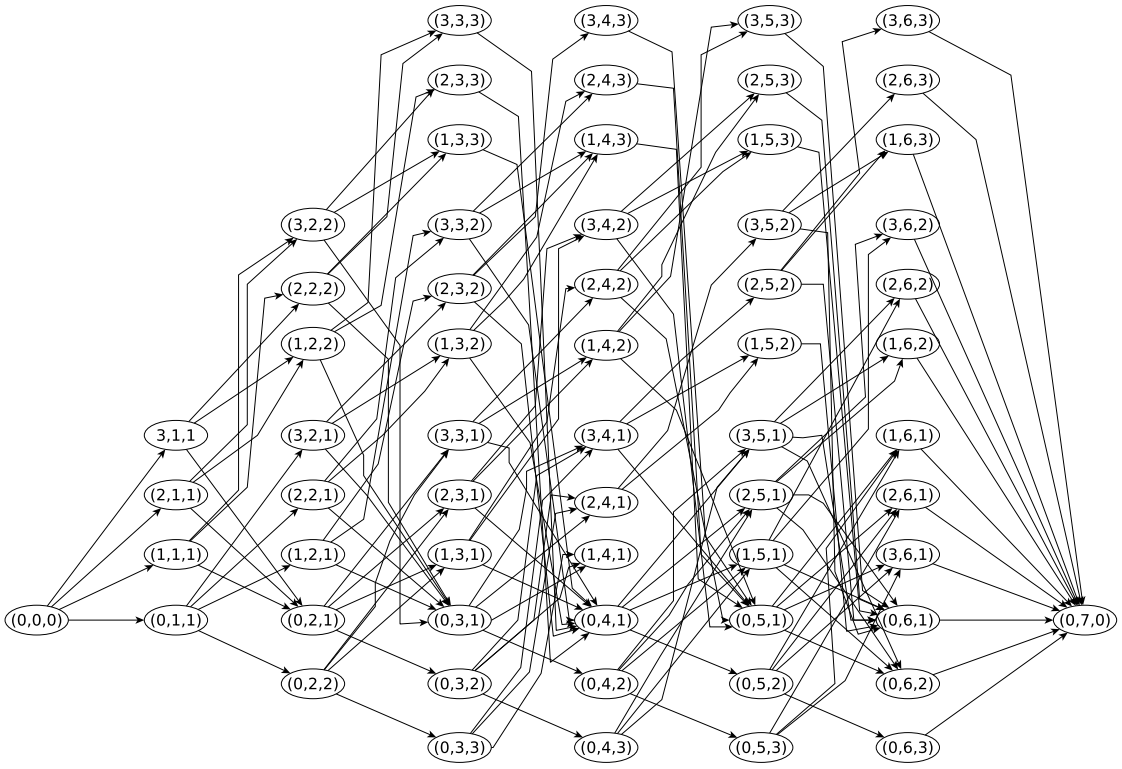

Figura 3.2: Grafo referente ao time 0 em uma instância com $n = 4$.

dinâmica, contendo os caminhos construídos no algoritmo. As colunas representam os vértices visitados nos caminhos. No problema de *pricing* utilizado no CVRP, cada cliente i possui uma demanda d_i e o objetivo é construir um caminho mínimo, percorrido por um veículo com capacidade C onde o somatório das demandas atendidas seja menor que C. Na adaptação deste problema para o TTP, cada time $i \neq t$ possui uma demanda de uma unidade e o objetivo é construir um caminho mínimo percorrido pelo time t onde o somatório das demandas é exatamente igual à $C = n - 1$. Para isto, as linhas são enumeradas de 0 à $n - 1$, indicando o consumo de recursos nos caminhos. Dada esta estrutura, a forma mais simples de obter um caminho mínimo do vértice origem ao vértice destino do grafo exposto em 3.2 é armazenar em cada célula $M[q, u]$ da matriz uma única *label* que representa o menor caminho da origem ao vértice u que faça exatamente q visitas à times rivais(diferentes de t). O algoritmo inicia inserindo uma *label* de custo zero na célula $M[0, u = (t, 0, 0)]$ e *labels* nulas com custo infinito em todas as outras células. Após esta etapa inicial a matriz é preenchida percorrendo as linhas em ordem crescente. Seja $c(M[q, u])$ o custo da *label* presente na célula $M[q, u]$ onde $u = (i, s, b)$. Para cada vizinho $v = (j, s + 1, b')$ de u, o custo da extensão é calculado da seguinte forma: Se $c(M[q, u]) + \tilde{c}^t_{ijs} < c(M[q + a, v])$ então a célula $M[q + a, v]$ é atualizada com uma *label* cujo custo seja $c(M[q, u]) + \tilde{c}^t_{ijs}$ e o predecessor de v é atualizado

para u. Se $j \neq t$ então $a = 1$, indicando que uma unidade de recurso foi consumida ao visitar j. Quando $j = t$, nenhum recurso é consumido e a atualização é realizada considerando $a = 0$. Ao final deste algoritmo, a *label* presente na célula $M[n-1, (t, 2\bar{n}, 0)]$ indica a rota de custo mínimo. O problema com este algoritmo é o fato de que, utilizando somente uma *label* por célula, não há como proibir que um time $t^{'}$ seja visitado mais de uma vez no caminho mínimo encontrado (e consequentemente algum outro time deixe de ser visitado), o que não é permitido nas rotas de uma solução do TTP. Como encontrar uma rota onde não haja repetições nos times visitados demanda muito tempo, uma das formas de relaxar a restrição é impor que um determinado time não pode ser revisitado sem que tenham se passado uma determinada quantidade de rodadas. Os principais algoritmos propostos por este trabalho são a aplicação de duas regras para impor a restrição mencionada: O *q-routes*, que elimina ciclos de tamanho 2 e o *NG-routes*, que propõe um outro método para proibir ciclos menores nas rotas encontradas. Estes algoritmos são explicados em detalhes a seguir.

3.2.4 *Q-routes* com eliminação de 2-ciclos

No trabalho sobre o CVRP descrito em [35], é introduzido o conceito de *q-rota*. Uma *q-rota* é uma rota partindo do depósito, visitando uma sequência de clientes com demanda até C e retornando para o depósito. Note que uma q-rota é uma relaxação das rotas válidas do CVRP, pois em uma q-rota os clientes podem ser visitados mais de uma vez, o que não ocorre numa rota válida do CVRP. Neste trabalho realizaremos uma adaptação do conceito de q-rotas para o TTP: uma q-rota para um time t é uma rota que visita t $n-1$ vezes e onde outros times são visitados $n-1$ vezes. Como no CVRP, um time $i \neq t$ pode ser visitado mais de uma vez, e portanto as q-rotas dos times são relaxações para as rotas válidas no TTP.

Ao eliminar ciclos de tamanho 2, são eliminadas do conjunto de q-rotas sequências de times do tipo $i - j - i$, onde i e j são diferentes de t. Para realizar essas eliminações dois passos devem ser executados. Seja i o vértice sendo visitado atualmente e $pred(i)$ o seu predecessor no caminho representado pela *label* l, e seja l a *label* sendo processada. O primeiro passo é proibir que l estenda o caminho atual para $pred(i)$. Como a casa do time t deve ser visitada $n-1$ vezes, a checagem do predecessor desconsidera t, proibindo não só ciclos do tipo $i - j - i$, assim como os ciclos $i - j - t - i$, $i - t - j - i$ e $i - t - j - t - i$, onde i, j e t são três times diferentes. Devido a esta proibição na extensão das *labels*, ao armazenar somente uma *label* na matriz, perde-se a garantia de encontrar o caminho mínimo no algoritmo mencionado anteriormente. Suponha que o time j está sendo visitado atualmente no caminho referente à *label* l e que i é o seu predecessor. Como a

extensão ao time i está proibida na *label* l, não há garantia de que haja outra *label* l' cujo caminho leva à i e resulta em menor custo. Portanto é necessário armazenar mais de uma *label* em cada célula da matriz. No caso específico dos ciclos de tamanho 2, basta armazenar 2 *labels*, uma vez que para quaisquer 3 *labels* da matriz, uma é sempre dominada em relação às outras. Como exemplo, considere o *q-routes* para o time 0 em uma instância com $n = 4$ e considere também uma *label* l referente ao caminho que atualmente se encontra no vértice $(j = 2, 3, 2)$ tendo como predecessor $pred(l) = 1$. Esta *label* l se encontra em $M[(2, 3, 2), 2]$ e durante seu processamento o caminho é estendido gerando duas *labels*: l_0 e l_1. Neste caso, l_0 com custo $c(l_0) = c(l) + \tilde{c}^0_{203}$ pode ser inserida em $M[(0, 4, 1), 2]$. Se já existe uma *label* l' com predecessor i na célula e $c(l_0) < c(l')$ então l_0 substitui l' na célula. Caso contrário, l_0 substitui a *label* de maior custo na célula ou é descartada se ela mesma possui o maior custo. Temos que $pred(l_0) = pred(l)$ pois esta extensão representa um retorno para casa. Note que a informação que a aresta ligando $(2, 3, 2)$ à $(0, 4, 1)$ também deve ser armazenada para que o caminho seja reconstruído no final do algoritmo. Como $pred(l) = 1$, a extensão à $(1, 4, 3)$ está proibida. A *label* l_1 com custo $c(l_1) = c(l) + \tilde{c}^t_{233}$ e $pred(l_1) = 2$ pode ser inserida em $M[(3, 4, 3), 3]$. O mesmo teste de dominância é realizado: considerando l_1 e as duas *labels* presentes em $M[(3, 4, 3), 3]$, l_1 será inserida se o seu custo estiver entre os dois menores, substituindo a *label* presente que também possua o time 2 como predecessor, se houver, ou a *label* de maior custo, caso contrário. Note que a cada extensão a informação da aresta utilizada também é armazenada para que a q-rota possa ser construída no final do algoritmo. O processamento das *labels* segue na ordem crescente das linhas de M e a q-rota mínima é obtida em $M[(0, 7, 0), 3]$.

3.2.5 *NG-routes*

Em [36] é definido o conceito de *NG-rota*. Nas NG-rotas a ideia é que após a visita à um time i em uma rota, não haja outra visita a ele antes que algum time fora da "vizinhança"de i seja visitado. Para isto, a informação de que i foi visitado é propagada até que a rota do time t saia da "vizinhança"de i. Uma vez que a rota está distante o suficiente de i, o mesmo pode ser visitado novamente. Para cada time $i \in T$ define-se N_i como o conjunto NG de i, que representa esta "vizinhança". Este conjunto é definido como os $k = |N_i|$ vizinhos mais próximos de i. Com k grande o suficiente, é possível que várias rotas geradas não possuam times rivais à t repetidos (rotas sejam elementares). Para cada *label* l na extensão dos caminhos define-se também o conjunto $\Pi(l)$ com os times para os quais a extensão do caminho é proibida. Para um caminho $P = (t_0, t_1, t_2, ..., t_s)$ representado pela *label* l, temos $\Pi(l) = \{t_i \in P \setminus t_s : t_k \in \bigcap_{i=k+1}^{s} N_{t_i}\} \cup \{t_s\}$. A extensão da *label* l referente a um time

i gera uma nova *label* m referente a um time j, onde o conjunto $\Pi(m)$ é obtido a partir de $(\Pi(l) \cap N_j) \cup \{j\}$. Como o time t deve ser visitado $n-1$ vezes, t não possui um conjunto N_t e durante uma visita ao time t considera-se $\Pi(t) = \Pi(pred(t))$, onde $pred(t)$ é o predecessor de t no caminho considerado. Desta forma, os retornos para casa são sempre permitidos e após um retorno pra casa a informação dos conjuntos Π é mantida. Para garantir a otimalidade do caminho mínimo encontrado é necessário armazenar até $2^{|N_i|-1}$ nas células da matriz M referentes ao time i ([33]). Uma *label* l domina outra *label* m somente se o custo $c(l)$ do caminho representado por l é menor ou igual a $c(m)$ e se $\Pi(l) \subseteq \Pi(m)$. Somente *labels* não dominadas são armazenadas em M. O *NG-routes* é pseudo-polinomial quando os conjuntos $|N_i|$ tem tamanho fixo independente de n. Durante o preenchimento da matriz M, o *NG-routes* substitui a etapa de atribuição dos predecessores pela etapa de atribuição dos conjuntos Π. Por exemplo, considere o *NG-routes* para o time $t = 0$ com $n = 4$ e os seguintes conjuntos NG: $N_1 = \{2, 3\}$, $N_2 = \{1, 3\}$ e $N_3 = \{1, 2\}$. Partindo do vértice de origem $(0, 0, 0)$ são criadas as *labels* l_i referentes aos vértices $(i, 1, 1)$, com $i \neq t$. Nestas *labels* temos $\Pi(l_i) = (\Pi(t) \cap N_t) \cup \{i\} = (\emptyset \cap \emptyset) \cup \{i\} = \{i\}$. Consideremos agora o caso onde $i = 1$. A extensão a partir do vértice $(1, 1, 1)$ gera as *labels* m_0 referente ao vértice $(0, 2, 1)$ com $\Pi(m_0) = \{1\}$ (como o time t retornou para casa, as extensões proibidas são as mesmas do predecessor), m_2 referente ao vértice $(2, 2, 2)$ com $\Pi(m_2) = \{1, 2\}$ e m_3 referente ao vértice $(3, 2, 2)$ com $\Pi(m_3) = \{1, 3\}$. A *label* m_2 só pode ser estendida aos vértices $(3, 3, 3)$ e $(0, 3, 1)$ pois $\Pi(m_2)$ proibe um retorno à um vértice do time 1. Neste exemplo específico, ao continuar com as extensões de caminhos em respeito aos conjuntos Π, obtém-se somente rotas sem ciclos (sem revisitas a times diferentes de t), porém no caso geral do *NG-routes* não há como garantir a eliminação de ciclos de tamanhos 2 ou maiores, embora resultados anteriores indiquem que quanto maior os conjuntos NG forem, menores as chances de ciclos pequenos na rota obtida.

3.2.6 Heurística de *Pricing*

Com o intuito de reduzir ainda mais o tempo gasto na resolução do problema de *pricing*, uma heurística gulosa foi implementada. Dado um time t, o algoritmo consiste em construir uma rota para t de forma gulosa, minimizando o custo das viagens e também o número de viagens realizadas. Para isto o time t alterna entre o máximo de rodadas possíveis jogando em casa e o máximo de rodadas possíveis jogando fora. Essa alternância é realizada levando em conta o número de rodadas restantes e o número de rodadas em que t pode jogar em casa, para que não haja inviabilidades. Nas rodadas onde t está fora de casa, as viagens também são escolhidas de forma gulosa: a próxima viagem é sempre a de menor custo. Se a solução obtida desta

forma possui custo reduzido negativo, então a mesma é retornada como solução para o problema de *pricing*. Caso contrário, novas rotas são obtidas ao realizar trocas entre as partidas das rodadas s_1 e s_2, $\forall s_1, s_2 \in S = \{1, ..., 2(n-1)\}$. Estas trocas, que não violam as restrições *AtMost*, ocorrem até que não seja possível reduzir o custo da rota. Se em qualquer etapa deste processo for encontrada uma rota com custo reduzido negativo então esta é retornada como solução para o problema de *pricing*. Se ao fim do processo não for possível encontrar uma rota com custo reduzido negativo, então o *pricing* via modelo é executado.

3.3 Estabilização da Geração de Colunas

Com frequência, o algoritmo de geração de colunas demora a encontrar o valor da relaxação linear. Acredita-se que um dos fatores responsáveis por esta característica é o fato de que a cada iteração do algoritmo, os valores das variáveis duais apresentam grande oscilação. Este trabalho apresenta a aplicação de duas estratégias de estabilização aplicadas ao TTP: uma variante do método *Box Step* e o método de suavização de Neame.

3.3.1 Variante do método *Box Step*

Em [37], é apresentada uma abordagem para a estabilização que envolve a ideia de caixas de estabilidade. A vantagem do método apresentado é o fato de o mesmo permanecer dentro do contexto da programação linear. Seja o programa linear primal P (3.21 - 3.22) e o seu dual D (3.23).

$$
\begin{aligned}
(P)\min \quad & c^T x \\
\text{s.a.} \quad & Ax = b && (3.21)\\
& x \geq 0 && (3.22)
\end{aligned}
$$

$$
\begin{aligned}
(D)\max \quad & b^T \pi \\
\text{s.a.} \quad & A^T \pi \leq c && (3.23)
\end{aligned}
$$

O método de estabilização consiste em adicionar vetores de variáveis de excesso e de folga ao problema P, respectivamente y_- e y_+, com limites superiores ϵ_- e ϵ_+. Na função objetivo, as variáveis de folga são penalizadas com os vetores δ_- e δ_+. Estas modificações levam aos problemas $\tilde{P}$ (3.24 - 3.27) e $\tilde{D}$ (3.28 - 3.31).

$$
\begin{aligned}
(\tilde{P})\min\, & c^T\tilde{x} - \delta_-^T y_- + \delta_+^T y_+ \\
\text{s.a.} & A\tilde{x} - y_- + y_+ = b && (3.24)\\
& y_- \leq \epsilon_- && (3.25)\\
& y_+ \leq \epsilon_+ && (3.26)\\
& \tilde{x}, y_-, y_+ \geq 0 && (3.27)
\end{aligned}
$$

$$
\begin{aligned}
(\tilde{D})\max\quad & b^T\tilde{\pi} - \epsilon_-^T w_- - \epsilon_+^T w_+ \\
\text{s.a.}\quad & A^T\tilde{\pi} \leq c && (3.28)\\
& -\tilde{\pi} - w_- \leq -\delta_- && (3.29)\\
& \tilde{\pi} - w_+ \leq \delta_+ && (3.30)\\
& w_-, w_+ \geq 0 && (3.31)
\end{aligned}
$$

Reorganizando as restrições 3.29 e 3.30 em $\tilde{D}$, temos que $\delta_- - w_- \leq \tilde{\pi} \leq \delta_+ + w_+$. Logo, sempre que os valores das variávies duais $\tilde{\pi}$ estiverem fora do intervalo $[\delta_-, \delta_+]$ ocorrerá uma penalidade definida por ϵ_- e ϵ_+ e quantificada por w_- e w_+. Seja x^*, π^* soluções ótimas de P e D e $(\tilde{x}^*, y_-^*, y_+^*), (\tilde{\pi}^*, w_-^*, w_+^*)$ soluções ótimas de $\tilde{P}$ e $\tilde{D}$. Para que os problemas P e $\tilde{P}$ sejam iguais, uma das duas condições devem ocorrer: $\epsilon_- = \epsilon_+ = 0$ ou $\delta_- < \tilde{\pi}^* < \delta_+$. Estas condições definem um critério de parada para o algoritmo de estabilização. Os parâmetros $\epsilon_-, \epsilon_+, \delta_-, \delta_+$ são atualizados a cada iteração, da seguinte forma:

- Se a solução dual $\tilde{\pi}$ atual está dentro do *box* $[\delta_-, \delta_+]$: defina $\tilde{\pi}$ como o centro e reduza o *box*, diminuindo também as penalidades ϵ_-, ϵ_+. Neste trabalho o tamanho do *box* e as penalidades caem pela metade neste caso.

- Se $\tilde{\pi}$ atual está fora do *box* $[\delta_-, \delta_+]$: defina $\tilde{\pi}$ como o centro e aumente o *box*, aumentando também as penalidades ϵ_-, ϵ_+. O tamanho do *box* e a penalidade são dobrados neste caso.

O intervalo $[\delta_-^0, \delta_+^0]$ inicial é obtido através da execução da primeira iteração da geração de colunas, onde o *box* é construído ao redor da solução dual encontrada.

3.3.2 Método de Estabillização de Neame

Uma alternativa na estabilização das variáveis duais, reduzindo a oscilação, é utilizar no problema de *pricing* uma solução dual modificada, que combina o valor da solução

dual atual com soluções duais obtidas em iterações anteriores [38]. Seja π^k a solução dual obtida em uma iteração k. Ao invés de utilizar π^k como entrada para o problema de *pricing*, utiliza-se $\tilde{\pi}^k = \alpha\pi^k + (1-\alpha)\tilde{\pi}^{k-1}$. Desta forma $\tilde{\pi}^k$ se torna uma soma ponderada das iterações anteriores: $\tilde{\pi}^k = \sum_{i=0}^{k} \alpha(1-\alpha)^{k-i}\pi^k$, o que implica que com o passar das iterações da geração de colunas, soluções duais de iterações antigas se tornam obsoletas e deixam de ser consideradas. Quanto maior α, $\tilde{\pi}^k$ se torna mais próximo da solução dual atual e as soluções duais anteriores contribuem menos.

3.4 Metaheurística ILS

Com o objetivo de gerar soluções viáveis para o TTP e fornecer um conjunto inicial de colunas de boa qualidade para o problema mestre, este trabalho introduz uma metaheurística ILS para o TTP. A construção da solução é feita em duas etapas: a definição da agenda de jogos e a definição dos padrões de mando de jogos (HAP). A agenda de jogos é definida executando o método do círculo uma vez para cada etapa do torneio (rodadas 1 à $n-1$ e n à $2(n-1)$). Os HAPs da primeira etapa são definidos de forma a minimizar o número de viagens, maximizando o número de *breaks*. Os HAPs da segunda etapa são definidos a partir da inversão dos mandos de jogos obtidos na primeira. Nesta metaheurística, as restrições *NoRepeat* e *AtMost* são relaxadas e a função objetivo é modificada para penalizar as violações. O método de construção pode gerar violações na restrição *NoRepeat*, porém estas violações são ignoradas devido à penalidade imposta na função objetivo. A função objetivo modificada consta em 3.32. $f(S)$ corresponde ao somatório dos custos das viagens realizadas na solução S, v indica o número de violações das restrições *NoRepeat* e *AtMost* e ω indica o custo de sair da região viável. A ideia é penalizar com maior intensidade somente a primeira violação e atribuir uma penalização menor à violações subsequentes, por isto ω é multiplicado pelo termo sublinear $(1+\sqrt{v}ln(v)/2)$. Durante o algoritmo o parâmetro ω é aumentado sempre que uma nova solução inviável de menor custo é encontrada e reduzido sempre que uma nova solução viável de menor custo é encontrada.

$$\bar{f}(S) = \begin{cases} f(S) & \text{Se a solução é viável} \\ \sqrt{f(S)^2 + \omega(1+\sqrt{v}ln(v)/2)} & \text{, caso contrário} \end{cases} \tag{3.32}$$

Para a fase de busca local, as estruturas de vizinhança clássicas da literatura, introduzidas em [13], são: *Swap Homes, Swap Rounds, Swap Teams, Partial Swap Rounds, Partial Swap Teams.*

- ***Swap Homes***: Dados dois times a e b cujas partidas ocorrem nas rodadas

s_1 no estádio de a e s_2 no estádio de b. Um movimento na vizinhança *Swap Homes* consiste em tomar uma nova solução onde a partida que ocorre em s_1 passa a ser no estádio de b e a partida da rodada s_2 passa a acontecer no estádio de a.

- ***Swap Rounds***: Dadas duas rodadas s_1 e s_2, uma nova solução é obtida trocando todas as partidas que ocorrem em s_1 e s_2. Considerando um DRRT como uma matriz $S_{n\times 2(n-1)}$, um movimento *Swap Rounds* consiste em trocar as colunas s_1 e s_2 de S.

- ***Swap Teams***: Dados dois times a e b, é realizada uma troca entre todas as partidas dos mesmos, exceto nas rodadas onde a e b se enfrentam. Considerando a mesma matriz $S_{n\times(2n-1)}$, um movimento *Swap Teams* consiste em realizar uma troca entre as linhas a e b de S, exceto nas duas colunas onde a e b se enfrentam. Para cada rodada s, uma troca adicional deve ser feita nas rotas dos times que enfrentam a e b. Por exemplo, se uma troca é realizada entre $S_{as} = x$ e $S_{bs} = y$ (a enfrenta x e b enfrenta y na rodada s.), então as trocas entre $S_{xs} = a$ e $S_{ys} = b$ também devem ser realizadas.

- ***Partial Swap Rounds***: Dados um time t e duas rodadas s_i e s_j, é realizada uma troca entre as partidas realizadas por t nas rodadas s_i e s_j. Ao realizar a troca, mudanças em outras partidas das rodadas s_i e s_j devem ser realizadas para manter a viabilidade do DRRT. Se considerarmos um grafo com os n times como vértices e arestas entre os times se eles se enfrentam nas rodadas s_i e s_j então todos os times na mesma componente conexa de t precisam ter suas partidas trocadas. A tabela 3.1, adaptada de [13], mostra uma modificação num DRRT de $n = 6$ times. A ideia neste movimento é que, ao contrário da *Swap Rounds*, somente parte das colunas s_i e s_j sejam modificadas, evitando que a solução receba uma perturbação muito grande.

- ***Partial Swap Teams***: Dados dois times a e b e uma rodada s, os jogos de a e b na rodada s são trocados. Assim como na vizinhança anterior, modificações em outras partidas de a e b são necessárias para manter a viabilidade. Ao realizar o movimento, o time a recebe uma partida contra o time y, que jogava contra b e perde uma partida contra x, que passa a jogar contra b. Com esta troca a possui agora duas partidas contra y e b possui duas partidas contra x, então uma troca na outra rodada onde uma partida a vs y deve ocorrer. As trocas prosseguem desta forma até que o time a receba a partida contra x que foi perdida na troca inicial. Assim como no *Partial Swap Rounds*, a intenção desta vizinhança é induzir uma modificação menor que a vizinhança

Swap Teams. A tabela 3.1 mostra um exemplo de movimento *Partial Swap Teams*.

Tabela 3.1: Exemplo do movimento *Partial Swap Rounds* considerando $t = 6$, $s_1 = 2$ e $s_2 = 9$. Na direita temos o DRRT após a realização do movimento, onde os times afetados estão destacados.

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

T\R	1	2	3	4	5	6	7	8	9	10
1	6	4	2	3	-5	-4	-3	5	-2	-6
2	5	-6	-1	-5	4	3	6	-4	1	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	-1	-3	-6	-2	1	5	2	6	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	2	-5	4	-3	5	-2	3	-4	1

O algoritmo 2 detalha a metaheurística proposta. Dois passos são realizados sempre que uma nova solução viável S é encontrada: o limite superior UB para o TTP é atualizado e as rotas de S são adicionadas ao problema mestre. Quando uma nova solução inviável S_{inv} é encontrada, cada rota de S_{inv} é verificada e somente as rotas viáveis (rotas que não violam a restrição *AtMost*) são adicionadas. O método busca soluções por uma quantidade de tempo pré definida. As buscas locais são realizadas na vizinhança *Partial Swap Teams*. A fase de perturbação é realizada executando um movimento aleatório na vizinhança *Swap Homes*, que é o movimento mais rápido de ser realizado.

Tabela 3.2: Exemplo de movimento na vizinhança *Partial Swap Teams* com $a = 2$, $b = 5$ e $s = 6$. Em destaque no lado direito constam as partidas modificadas pelo movimento.

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	-6	6	-4	3	-3
3	-4	5	4	-1	6	-5	1	-6	-2	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	3	-4	-1	-6	4
6	-1	-4	-5	4	-3	2	-2	3	5	1

Algoritmo 2 Metaheurística ILS para o TTP.

1. $UB \leftarrow menor\_custo \leftarrow \infty$
2. Construa uma solução S
3. **faça**
4. Gere uma perturbação S' na solução S
5. **se** $f(S') < menor\_custo$ **então**
6. Adicione ao programa mestre as rotas viáveis presentes em S'.
7. **se** S' é viável e $f(S') < UB$ **então**
8. $UB \leftarrow f(S')$
9. **fim se**
10. **fim se**
11. Obtenha a solução $\bar{S}$ a partir de uma busca local em S'.
12. **se** $\bar{S} < menor\_custo$ **então**
13. Adicione ao programa mestre as rotas viáveis presentes em $\bar{S}$.
14. **se** $\bar{S}$ é viável e $f(\bar{S}) < UB$ **então**
15. $UB \leftarrow f(\bar{S})$
16. **fim se**
17. **fim se**
18. **enquanto** Tempo limite não se esgotou
19. Retorne o limite superior UB e o programa mestre com as colunas adicionadas.

Capítulo 4

Resultados e Discussões

Neste capítulo constam os resultados dos testes realizados para os algoritmos propostos nesta dissertação. O computador utilizado contém um processador *Intel*® *Core* I7 6700 com 3.4GHz e 16 GB de memória RAM. O código foi desenvolvido na linguagem C++ versão 11 e compilado com a opção -O3. Para os problemas de programação linear e inteira dos algoritmos foi utilizado o solver Gurobi, na versão 7.0.2. As rotinas de pré-processamento do solver e as rotinas que envolvem paralelismo foram desabilitadas. Na primeira seção os resultados referentes a geração de colunas com os diferentes algoritmos de *pricing* são exibidos. Na segunda seção constam os resultados dos limites inferiores obtidos pelos algoritmos de *pricing*. Os resultados da heurística de *pricing* são exibidos na terceira seção. Os resultados da metaheurística ILS constam na quarta seção e nas duas últimas seções constam os resultados dos algoritmos de estabilização de box e Neame respectivamente.

4.1 Geração de Colunas

Nesta seção é realizado o teste da geração de colunas utilizando os algoritmos de *pricing* via programação inteira (coluna 'Modelo'), *pricing* via *label* bidirecional (coluna 'Label'), *pricing q-routes* com eliminação de 2-ciclos ('QR2') e *ng-routes* ('NG6'). A tabela 4.1 exibe os resultados obtidos. Para cada algoritmo são reportados o valor da relaxação linear encontrada (Z), o tempo em segundos (T(s)) e o número de iterações (It.). A coluna OPT indica o valor da solução ótima das instâncias já resolvidas. O *pricing* modelo e o *pricing* bidirecional não foram capazes de resolver as instâncias com 12 ou mais times num tempo aceitável. Cada iteração durava de mais de 400 segundos para $n = 12$ e por esse motivo para estes algoritmos somente os resultados até $n = 10$ constam na tabela. No primeiro teste o *pricing NG-routes* foi considerado com o tamanho dos conjuntos $N_i = 6$. O impacto do tamanho dos conjuntos N_i no algoritmo de *pricing* é investigado na tabela 4.2. A heurística de *pricing* não foi considerada neste teste e será avaliada nas próximas

seções. Os limites inferiores obtidos via *Label* são menores que os relatados em [22] pois o procedimento de redução de simetria lá descrito não foi utilizado neste teste. Optou-se por não utilizar a redução de simetria para que houvesse consistência com o valor da relaxação obtida no *pricing* via modelo. Os autores reportaram limites inferiores de 8.160, 22.582,6 e 38.670 para as instâncias nl4, nl6 e nl8, respectivamente. O tempo de execução da geração de colunas não foi reportado naquele artigo, porém o algoritmo de *label* lá implementado é muito superior ao nosso: os autores reportam para as instâncias nl6 e nl8 tempos médios de 0,0004 e 0,001 segundos. Na implementação aqui apresentada os tempos médios para nl6 e nl8 são 0,016 e 0,621 segundos.

Tabela 4.1: Resultados obtidos na geração de colunas.

Instância	OPT	Modelo			Label			QR2			NG6		
		Z	T(s)	It.	Z	T(s)	It.	Z	T(s)	It.	Z	T(s)	It.
nl4	8.276	8.044	17,57	25	8044	6,29	25	8.044	5,45	19	8.044	6,22	19
nl6	23.916	22.557	100,03	60	22557	27,31	52	21.025,90	30,26	67	22.557	26,03	57
nl8	39.721	38.670	598,75	77	38670	442,20	79	33.220	108,64	136	38.406	197,42	92
nl10	59.436	56.659	70.430,08	183	56.659	59.751,03	176	45.834,25	272,58	215	51.475	852,60	203
nl12	-	-	-	-	-	-	-	85.514,43	799,49	328	93.933	1.995,50	240
nl14	-	-	-	-	-	-	-	128.818,40	1993,37	439	141.997,84	5.733,52	403
nl16	-	-	-	-	-	-	-	141.563,26	6460,41	646	167.686,81	12250,30	500
nfl18	-	-	-	-	-	-	-	172442,04	9.759,60	627	188.499,79	19.996,07	616
br24	-	-	-	-	-	-	-	214.546,32	48.617,80	944	259.479,72	120.935,98	758

Os resultados apontam que o *Q-routes* e *NG-routes* conseguem obter limites inferiores de forma mais rápida que as outras estratégias. Estes limites são mais fracos, mas permitiram que a geração de colunas pudesse ser executada para as instâncias maiores. Ao observar a tabela podemos perceber que entre o *Q-routes* e o *NG-routes* com $|N_i| = 6$ há um aumento de entre 7% e 19% no limite inferior, ao passo que o tempo de execução mais que dobra para a maioria das instâncias. Apesar de ser aplicável à instâncias maiores o *NG-routes* ainda precisa ser aprimorado para que se torne eficiente. Na tabela 4.2 constam os resultados da geração de colunas utilizando o *pricing NG-routes* com diferentes tamanhos de conjuntos N_i. Alguns testes para o *NG-routes* com $|N_i| = 8$ e $|N_i| = 10$ demoraram mais de 40 horas sem que a geração de colunas terminasse e estão indicados por '-' na tabela. Como esperado, quanto maior o tamanho do conjunto N_i, maior o tempo para executar a geração de colunas. Podemos notar nas instâncias nl8 e nl10, com $|N_i| = 8$ e $|N_i| = 10$ respectivamente, que o valor da relaxação foi o mesmo que os *pricing* modelo e bidirecional porém em menos tempo.

Tabela 4.2: Resultados da geração de colunas do *NG-routes* para diferentes $|N_i|$.

Instância	NG-2			NG-4			NG-6			NG-8			NG-10		
	Z	T(s)	It	Z	T(s)	It	Z	T(s)	It	Z	T(s)	It	Z	T(s)	It
nl4	8.044	5,75	23	8.044	5,84	19	8.044	6,22	18	8.044	6,11	18	8.044	6,18	18
nl6	21.197,56	52,78	68	21.901,57	30,34	57	22.557	26,03	57	22.557	27,71	57	22.557	29,40	57
nl8	34.601,30	152,59	130	35.512,8	121,53	126	38.406	186,71	92	38.670	120,94	80	38.670	120,18	80
nl10	48.226,65	343,37	222	49.919,7	400,54	207	51.475	684,78	203	56.339,8	6.054,86	200	56.658,7	2518,33	176
nl12	87.884,70	740,76	283	90.980,21	1.164,93	298	93.933	1.773,83	240	98.699,32	12.705,5	232	105.000,4	143.387	201
nl14	131.791	2146,05	415	138.230	2.981,39	408	141.997,84	5.846,21	403	150.802,54	36.642,4	384	-	-	-
nl16	145.233	5504,57	567	164.084	7.346,54	535	167.686,81	12.250,30	500	176.568	55.343,5	510	-	-	-
nfl18	172.662	10.416,9	602	176.742	14.344,9	604	188.499,79	19.996,07	616	-	-	-	-	-	-

4.2 *Q-routes* e *NG-routes* como Limites Inferiores

Nesta seção foi considerado o desempenho dos algoritmos de *pricing* como limites inferiores para o TTP. Para a realização deste teste o algoritmo de *pricing* é executado considerando como custos reduzidos as distâncias originais do problema. Na tabela 4.3 constam limites inferiores obtidos e a média do tempo de 3 execuções no *Q-routes* e no *NG-routes* com variados tamanhos dos conjuntos N_i. Na coluna MNT, com exceção da instância br24, constam os limites inferiores obtidos em [10]. O objetivo naquele trabalho era encontrar limites inferiores e o tempo necessário para encontrar somente os mesmos não foi informado. Em negrito, consta o IB da instância br24, calculado neste trabalho utilizando um solver. Este resultado será discutido na próxima subseção.

Tabela 4.3: Resultados dos teste *Q-routes* e *NG-routes* como limite inferior.

Instância	MNT	qr2		ng2		ng4		ng6		ng8		ng10	
		T(s)	LB	T(s)	LB	T(s)	LB	T(s)	LB	T(s)	LB	T(s)	LB
nl4	8.276	0,002	5.396	0,002	8.044	0,004	8.044	0,004	8.044	0,007	8.044	0,007	8.044
galaxy4	-	0,002	314	0,006	412	0,005	412	0,005	412	0,005	412	0,005	412
nl6	22.594	0,02	14.264	0,04	17.303	0,08	21.184	0,06	22.557	0,06	22.557	0,06	22.557
galaxy6	-	0,02	957	0,04	1.132	0,07	1.275	0,07	1.294	0,07	1.294	0,07	1.294
nl8	38.670	0,03	25.556	0,08	27.368	0,29	29.142	1,61	35.890	0,82	38.670	0,84	38.670
galaxy8	-	0,04	1.788	0,07	1.921	0,33	2.162	1,32	2.229	0,97	2.250	0,95	2.250
super8	-	0,04	30.414	0,11	103.027	0,24	133.318	1,16	175.696	0,83	177.258	1,09	177.258
nl10	56.928	0,14	34.347	0,27	37.972	0,96	39.702	3,64	43.326	32,12	52.855	13,64	56.506
galaxy10	-	0,16	2.951	0,26	3.181	1,05	3.466	3,92	3.682	21,49	4.028	12,80	4.333
super10	-	0,15	43.936	0,29	95.133	0,96	122.073	3,25	217.764	28,56	252.854	17,03	313.408
nl12	107.494	0,42	58.895	0,77	64.316	2,66	68.929	8,78	74.762	67,29	86.140	759,59	100.556
galaxy12	-	0,37	4.579	0,67	4.833	2,59	5.297	10,22	5.600	23,26	5.839	338,44	6.329
super12	-	0,44	73.300	0,85	124.557	2,03	191.103	9,38	274.157	36,51	282.878	618,83	350.275
nl14	182.797	1,01	99.896	2,66	105.924	8,81	110.942	27,5	120.579	200,53	137.718	748,47	145.886
galaxy14	-	0,98	6.592	1,46	6.900	5,01	7.615	17,81	8.059	51,17	8.466	306,41	9.080
super14	-	1,00	94.005	1,89	131.462	4,25	228.332	17,61	263.857	51,92	344.362	210,95	360.788
nl16	249.477	2,05	103.083	4,85	20.141	15,50	135.021	40,39	144.263	221,37	159.418	785,45	166.899
galaxy16	-	1,94	8.800	2,94	9.123	7,91	9.944	28,07	10.465	75,67	10.813	312,23	11.262
nfl18	272.834	4,05	132.959	9,32	137.048	23,04	143.370	64,81	154.586	248,51	171.431	1.482,08	182.327
galaxy18	-	3,97	11.299	5,92	11.863	11,91	12.377	38,25	13.185	107,44	13.714	406,90	14.435
galaxy20	-	7,20	13.758	9,69	14.592	18,05	15.369	53,21	16.159	146,09	16.888	495,66	17.641
galaxy22	-	11,81	17.672	15,78	18.712	29,46	19.433	80,27	20.484	211,72	21.387	676,94	22.273
br24	**389.832**	17,42	127.487	26,09	163.742	49,21	174.448	131,03	185.221	436,18	197.949	2.365,45	212.062
galaxy24	-	18,71	22.256	24,85	23.616	47,24	24.713	106,97	25.402	265,58	26.660	864,37	27.691
galaxy26	-	29,08	28.058	36,83	29.734	67,59	30.864	155,86	31.732	393,78	33.284	1.317,82	34.545
galaxy28	-	42,10	35.062	54,84	37.079	96,36	38.337	221,94	39.415	543,73	41.155	1.781,22	42.769
galaxy30	-	62,65	43.258	72,46	45.532	137,45	47.405	306,86	48.470	722,25	50.466	2.302,33	52.369

Nestes resultados podemos verificar como o tempo e o valor do limite inferior se relacionam a medida que o tamanho dos conjuntos N_i crescem. Como esperado, os limites inferiores obtidos são mais fracos, uma vez que o *q-routes* e *ng-routes* relaxam a restrição de visitar todos os times. A tabela também mostra que executar o *NG-routes* com $N_i = n$ pode ser melhor que executar o algoritmo com N_i menores próximos a n. Por exemplo, nas instâncias com 10 times o tempo necessário para executar o *NG-routes* com $|N_i| = 8$ foi quase o dobro do obtido no teste com $|N_i| = 10$. Na instância nl12 com $|N_i| = 10$ 759,59 segundos. Um teste extra foi feito para o *NG-routes* na instância nl12 com $|N_i| = 12$ e o tempo necessário para encontrar o limite de 107.483 foi 247,38 segundos: menos da metade do tempo do caso anterior, sugerindo que esse comportamento se repete em várias instâncias.

4.2.1 Limite independente vs *Q-routes*/*NG-routes*

Na url `https://allgo.inria.fr/app/vehiclerouting` há um solver online do problema de roteamento de veículos. O limite independente (IB) da instância br24 foi calculado utilizando este solver com a capacidade dos veículos sendo igual a 3, representando a restrição AtMost. Neste teste foram necessários 96,74 segundos para encontrar o IB de 389.832. Na tabela 4.3 podemos verificar que o *Q-routes* e o *NG-routes* com $|N_i| = 4$ conseguem obter um limite inferior aproximadamente na metade do tempo, porém muito mais fraco. A única informação disponível sobre o computador é a quantidade de memória RAM (4 GB), porém como o solver se encontra nos servidores do INRIA, é provável que o poder de processamento seja maior que o da máquina utilizada nos nossos testes e portanto a diferença entre os tempos pode ser ainda maior. Vale notar também que a implementação do CVRP utilizada no INRIA também utiliza o *NG-routes* como algoritmo de *pricing* em um branch and price muito elaborado. Portanto, adaptar ao TTP a implementação utilizada no solver poderia melhorar muito os resultados obtidos.

4.3 Heurística de *Pricing*

Nesta seção é feita uma comparação entre o uso da heurística de *pricing* em relação ao uso do modelo. Cada instância executou a geração de colunas com um tempo limite de 3 horas. A tabela 4.4 exibe os resultados obtidos. A tabela contém a coluna Z que indica o valor da relaxação obtida, o número de iterações executadas e o tempo em segundos da execução. Em negrito os valores que apresentaram o melhor resultado. No testes realizados a diferença nas instâncias com 8 e 10 times, porém a diferença na instância com 12 times foi de 3,5% o que indica que talvez o método seja melhor para instâncias maiores. Todavia mais testes são necessários

para se chegar a uma conclusão.

Tabela 4.4: Comparação entre o *pricing* via modelo e a heurística de *pricing* aliada ao modelo.

Instância	Modelo			Heurística+Modelo		
	Z	It.	T(s)	Z	It.	T(s)
nl8	38.670	77	716,85	38.670	103	**514,69**
galaxy8	2.250	77	342,18	2.250	101	**334,95**
super8	177.258	89	1.176,18	177.258	124	**1.174,53**
nl10	56.771	129	10800	**56.722**	177	10800
galaxy10	4.353,24	141	10800	**4.352,40**	180	10800
super10	322.200,74	62	10800	**320.282,00**	110	10800
nl12	114.025,75	86	10800	**110.015,28**	176	10800

4.4 Metaheurística ILS

Nesta seção constam os resultados obtidos ao executar a metaheurística ILS para se gerar um pool de colunas iniciais. No primeiro teste, a ILS adiciona colunas ao problema mestre por 1000 segundos e posteriormente a geração de colunas é executada com os *pricings* *Q-routes* e *NG-routes*. Este valor foi escolhido para que a metaheurística pudesse ter tempo de gerar boas colunas, permitindo assim uma avaliação sobre como a adição de boas colunas impactam o algoritmo de geração de colunas. As colunas da tabela 4.5 representam respectivamente o tempo total, o tempo destinado à ILS, o tempo gasto somente na geração de colunas, o número de iterações da geração de colunas, o número de colunas adicionadas pela ILS e a porcentagem da variação entre o tempo obtido na geração de colunas com e sem a metaheurística. V. GC = $\frac{(t_{ils}-t)100}{t}$, onde t_{ils} é dado pelo valor da coluna 'T. Total(s)' e t é dado pela coluna 'T(s)' que representa o tempo da geração de colunas simples. Quando V. GC é maior que zero, a ILS+GC foi pior em relação a GC simples.

Como é possível notar, a execução por 1000 segundos não implicou em uma melhora no tempo da geração de colunas, possivelmente pois o número de colunas adicionadas foi muito grande, impactando negativamente nas execuções do problema mestre. Um segundo teste foi realizado na tabela 4.6, onde a execução da metaheurística foi limitada a 10% do tempo obtido na geração de colunas simples. Na última coluna desta tabela é possível notar que essa redução no tempo de execução da ILS diminuiu seu impacto negativo. Nas instâncias nl12 e nl14 com *Q-routes* é possível notar que as colunas adicionadas pela ILS ajudaram na geração de colunas, uma vez que ao adicionar 10% de tempo para a ILS, o tempo total da geração de colunas mais ILS aumentou somente 2,72% e 0,72%, respectivamente. Esta característica se observa também na instância nl10 com *NG-routes*, onde o aumento foi

Tabela 4.5: *Q-routes* e *NG-routes* após a execução de 1000 segundos da metaheurística.

Q-Routes									
Inst.	GC			ILS+GC					
	T(s)	It.	Cols	T. Total(s)	T. H(s)	T. GC(s)	It.	Cols H	V. GC(%)
nl10	272,58	211	2029	1.453,30	1.001,00	452,30	166	5774	433,16
nl12	799,49	323	3317	2.017,03	1.001,11	1.015,92	257	5724	152,29
nl14	1.993,37	404	5258	3.452,11	1.003,13	2.448,98	378	3481	73,18
NG-Routes($\|N\| = 6$)									
Inst.	GC			ILS+GC					
	T(s)	It.	Cols	T. Total(s)	T. H(s)	T. GC(s)	It.	Cols H	V. GC(%)
nl10	852,60	203	1839	2.077,27	1.000,81	1.076,46	139	4809	143,64
nl12	1.995,50	240	3473	3.110,35	1.000,25	2.110,10	184	3648	55,87
nl14	5.733,52	403	5487	7.492,55	1.003,31	6.489,24	346	4511	30,68

de somente 3%. Contudo, para alcançar um ganho real ainda é necessário aprimorar a heurística.

Tabela 4.6: Resultados *Q-routes* e *NG-routes* executando ILS por 10% de tempo obtido na tabela 4.1.

Q-Routes									
Inst.	GC			ILS+GC					
	T(s)	It.	Cols	T. Total(s)	T. H(s)	T. GC(s)	It.	Cols H	Var GC
nl10	272,58	215	2029	356,36	27,2	329,16	214	248	30,74
nl12	799,49	328	3317	821,67	79,9	741,77	297	702	2,77
nl14	1.993,37	439	5258	2007,77	199,33	1808,47	391	1600	0,72
NG-Routes($\|N\| = 6$)									
Inst.	GC			ILS+GC					
	T(s)	It.	Cols	T. Total(s)	T. H(s)	T. GC(s)	It.	Cols H	Var GC
nl10	852,60	203	1839	879,08	85,2	793,88	169	576	3,11
nl12	1.995,50	240	3473	2245,58	199,5	2046,08	199	1309	12,53
nl14	5.733,52	403	5487	6490,02	573,3	5916,72	346	2458	13,19

4.5 Estabilização do Box Step

A tabela 4.7 contém os resultados para a geração de colunas utilizando o *pricing Q-routes* e a estabilização do box step. Os tamanhos de box testados foram de 200,500 e 1000 e a penalidade $\epsilon = 0,1$. A tabela 4.8 contém os mesmos testes com $\epsilon = 1$. Ao testar valores de ϵ maiores que 1 a geração de colunas chega a divergir: Nos testes com $\epsilon = 5$ o método divergiu com o box de tamanho 200. Com $\epsilon = 10$, somente o box de tamanho 1000 não divergiu. Mesmo quando o método não divergia, os resultados eram inferiores aos aqui apresentados. Nas primeiras colunas das tabelas constam os resultados da geração de colunas simples. As próximas colunas contém respectivamente o tempo (T(s)), o número de iterações (It) e a porcentagem de variação no tempo em relação a geração de colunas sem nenhuma estabilização, calculada de forma análoga a seção anterior: V. GC = $\frac{(T(s)_{box} - T(s)_{GC})100}{T(s)_{GC}}$. As tabelas 4.9 e 4.10 contém os mesmos testes utilizando o *pricing NG-routes*. Para a maioria das instâncias não houve melhora significativa, sendo uma redução de 8,15% contra aumentos de 20% em várias instâncias. Nos testes realizados, as diferenças entre os parâmetros não foram significativas

Tabela 4.7: *Pricing Q-routes* e estabilização box step ($\epsilon = 0,1$).

Instância	GC		Box = 200			Box = 500			Box = 1000		
	T(s)	It.	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)
nl10	272,58	215	325,90	233	19,56	326,72	251	19,86	322,89	230	18,46
nl12	799,49	328	784,16	328	-1,92	777,71	318	-2,72	860,59	324	7,64
nl14	1.993,37	439	1.911,76	403	-4,09	1.949,31	428	-2,21	1.854,94	406	-6,94

Tabela 4.8: *Pricing Q-routes* e estabilização box step ($\epsilon = 1$).

Instância	GC		Box = 200			Box = 500			Box = 1000		
	T(s)	It.	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)
nl10	272,58	215	327,15	233	20,02	327,71	233	20,23	322,86	230	18,45
nl12	799,49	328	789,17	328	-1,29	777,71	318	-2,72	787,44	308	-1,51
nl14	1.993,37	439	1.911,76	403	-4,09	2365,76	434	18,68	1946,35	407	-2,36

Tabela 4.9: *Pricing NG-routes* e estabilização box step ($\epsilon = 0,1$).

Instância	GC		Box = 200			Box = 500			Box = 1000		
	T(s)	It.	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)
nl10	852,60	203	862,14	195	1,12	795,17	192	-6,74	850,27	195	-0,27
nl12	1.995,50	240	2.035,50	240	2,00	2.023,60	240	1,41	2.046,51	246	2,56
nl14	5.733,52	403	5.862,01	408	2,24	5.691,34	400	-0,74	5.968,14	422	4,09

Tabela 4.10: *Pricing NG-routes* e estabilização box step ($\epsilon = 1$).

Instância	GC		Box = 200			Box = 500			Box = 1000		
	T(s)	It.	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)
nl10	852,60	203	730,31	177	-14,34	898,62	201	5,40	783,12	189	-8,15
nl12	1.995,50	240	2465,07	286	23,53	2010,32	234	0,74	2284,79	273	14,50
nl14	5.733,52	403	5576,63	388	-2,74	5389,08	373	-6,01	5793	407	1,04

4.6 Estabilização de Neame

Para avaliar a qualidade da estabilização de Neame, foram realizados testes considerando os parâmetros $\alpha \in \{0,2; 0,5; 0,75\}$. Nas tabelas 4.11 e 4.12 constam, os resultados para a geração de colunas com *Q-routes* e *NG-routes*, respectivamente. As colunas indicam o tempo, o número de iterações, e a porcentagem da variação entre o tempo obtido e o tempo da geração de colunas simples. Como é possível notar, a estabilização foi eficiente na maioria dos testes realizados, alcançando até 36% de redução no *Q-routes*. Dados os testes realizados não é possível concluir qual o melhor valor de α a se utilizar, sendo necessários mais testes para verificar se há mesmo uma melhor escolha. Ao comparar com o método do box, é possível notar também que a estabilização de Neame foi mais eficiente, levando a reduções no tempo e número de iterações de forma mais consistente.

Tabela 4.11: Estabilização de Neame aplicada ao *Q-routes*.

Instância	GC		$\alpha = 0,2$			$\alpha = 0,5$			$\alpha = 0,75$		
	T(s)	It.	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)	T(s)	It.	V. GC(%)
nl8	95,67	136	154,61	173	61,61	117,12	136	22,42	112,44	127	17,53
nl10	272,58	215	312,255	238	14,56	260,45	206	-4,45	257,37	200	-5,58
nl12	799,49	328	596,34	295	-25,41	538,99	259	-32,58	609,13	284	-23,81
nl14	1.993,37	439	1260,35	346	-36,77	1483,37	365	-25,58	1573,92	387	-21,04
nl16	6.460,41	646	4245,27	590	-34,29	4304,83	499	-33,37	4978,69	543	-22,94
nfl18	9.759,60	627	5529,89	458	-43,34	6884,15	512	-29,46	8044,51	554	-17,57
br24	48.617,80	944	30158,5	594	-37,97	38593,6	688	-20,62	54324,9	823	11,74

Tabela 4.12: Estabilização de Neame aplicada ao *NG-routes* com $|N| = 6$.

Instância	GC		$\alpha = 0,2$			$\alpha = 0,5$			$\alpha = 0,75$		
	T(s)	It.	Tempo(s)	Iterações	V. GC	Tempo(s)	Iterações	V. GC	Tempo(s)	Iterações	V. GC
nl8	197,42	92	319,44	149	61,81	213,49	104	8,14	187,9	91	-4,82
nl10	852,60	203	833,8	209	-2,21	700,13	176	-17,88	735	181	-13,79
nl12	1.995,50	240	1.876,25	232	-5,98	1.599,03	201	-19,87	1.703,18	221	-14,65
nl14	5.733,52	403	4.383,75	332	-23,54	4.398,19	334	-23,29	4.895,57	364	-14,61
nl16	12250,3	500	8960,82	378	-26,85	9415,58	392	-23,14	10985,11	449	-10,33

Capítulo 5

Conclusão

Este trabalho apresenta um algoritmo de geração de colunas baseado em [22]. Para o algoritmo de *pricing*, cinco estratégias foram implementadas, sendo os itens 3,4 e 5 contribuições originais:

1. Um modelo de programação inteira baseado num grafo discretizado em rodadas, apresentado em [22].

2. Uma adaptação do algoritmo de labeling bidirecional também descrito em [22].

3. Uma heurística de *pricing* gulosa, utilizada em conjunto com o algoritmo do item 1.

4. O algoritmo de *pricing Q-routes*, uma adaptação para o TTP do algoritmo de *pricing* utilizado no CVRP [32]. Neste algoritmo as rotas dos times são substituidas por q-rotas: rotas onde um time $i \neq t$ pode ser visitado mais de uma vez desde que não hajam sequências de times do tipo $i - j - i$, com $j \neq t$.

5. O algoritmo de *pricing NG-routes*, onde um conjunto de vizinhos é atribuído para cada time. Uma rota de um time t pode conter mais de uma visita ao time $i \neq t$ desde que, após a primeira visita ao time i, algum time fora da vizinhança de i seja visitado.

Na esperança de melhorar o desempenho da geração de colunas, duas estratégias de estabilização também foram utilizadas: uma variante do método box step e a estabilização de Neame. Por fim uma metaheurística ILS foi implementada para gerar um pool de colunas iniciais.

O *pricing* baseado no modelo de programação inteira e o *pricing* via label bidirecional não obtiveram resultados satisfatórios para instâncias com mais de 10 times. Quanto ao *pricing* bidirecional, o desempenho ruim se deve principalmente a implementação utilizada, uma vez que os tempos obtidos foram piores que os obtidos no artigo original da implementação. A heurística de *pricing* obteve resultados

um pouco melhores que os encontrados para o modelo, porém a melhoria não foi significativa.

Ao utilizar o *pricing Q-routes* e *NG-routes* foi possível obter limites inferiores de forma rápida para instâncias maiores. Os limites obtidos pelo *Q-routes*, apesar de serem mais fracos que os obtidos pelo *NG-routes*, foram obtidas de forma mais rápida. O *NG-routes* pode ser promissor se a implementação for aprimorada para que a execução seja mais rápida.

O método de estabilização de box não conseguiu reduzir significativamente o tempo da geração de colunas nas instâncias testadas. Por outro lado a estabilização de Neame se provou eficiente no problema, reduzindo o tempo da geração de colunas em até 36%.

5.1 Trabalhos Futuros

Seguindo a abordagem proposta nesta dissertação, algumas sugestões de trabalhos futuros envolvem:

- Quanto ao *Q-routes*, Fukasawa, Longo et al.[32] concluem que a eliminação de 3-ciclos apresenta um benefício ao limite obtido gastando pouco tempo em relação à eliminação de 2-ciclos. Os autores também concluem que a eliminação de 4-ciclos não compensa pois a melhora no limite é pequena em comparação com o tempo gasto. Portanto parece razoável adaptar a eliminação de 3-ciclos ao TTP para avaliar os benefícios em comparação com a eliminação de 2-ciclos implementada nesta dissertação. Na implementação da eliminação de 3-ciclos cada célula da matriz M descrita no capítulo 3 deve armazenar até 6 labels e a proibição é estendida até o predecessor do predecessor de um vértice.

- Quanto ao *NG-routes*, em [33] é descrita uma adaptação ao *NG-routes* denominada DSSR (*Decremental State Space Relaxation*): Nesta técnica, o *NG-routes* é executado uma vez com todos os conjuntos N_i tendo tamanho unitário ($i \in T$). Se um time i for visitado mais de uma vez, como na sequência de times $i - j - ... - k - i$, então i é adicionado aos conjuntos de $j, ..., k$. Uma nova rodada pode ser executada e, caso hajam novos ciclos, um novo incremento nos conjuntos N é realizado. Desta forma somente os times mais envolvidos em ciclos possuem um conjunto N maior reduzindo assim o total de labels mantidas durante a programação dinâmica. Na utilização do DSSR no CVRP, esta técnica permitiu que fossem encontradas rotas sem ciclos de forma rápida para instâncias maiores do problema. Convém, portanto, adaptar o DSSR para o TTP também e avaliar seu impacto.

- A geração de colunas descrita nesta dissertação pode ser utilizada num algoritmo branch and price de forma a avaliar com mais profundidade o desempenho dos algoritmos propostos neste trabalho.

- Na heurística de *pricing*, o método de construção pode ser modificado da seguinte forma: ao invés de se manter fora ou dentro de casa o máximo de rodadas possível, pode ser vantajoso verificar o custo de voltar para casa ou sair de casa antes do máximo de rodadas. Esta idéia é razoável pois durante a geração de colunas os custos obtidos pela solução dual podem variar muito. A implementação de heurísticas de *pricing* para o *Q-routes* e *NG-routes* também é um caminho a seguir.

- Na estabilização de Neame, uma possível melhoria pode ser obtida ao elaborar um esquema de atualização dinâmica do parâmetro α. Uma estratégia de atualização automática de α é descrita em [38]. A atualização é baseada em uma estimativa sobre a qualidade da solução dual π^* obtida. A ideia é usar um valor de α pequeno quando acredita-se que π^* é ruim e aumentar α a medida que a qualidade de π^* melhora.

- Outras heurísticas que gerem soluções viáveis rapidamente podem ser aplicadas na geração de um pool inicial de colunas. A ILS implementada neste trabalho foi testada somente com uma vizinhança. As outras vizinhanças também podem ser avaliadas nesta ILS.

Referências Bibliográficas

[1] ESPORTES, R. S. "Qual clube tem maior faturamento? E a maior dívida? Consultoria BDO analisa dados financeiros." *MG Super Esportes*, maio 2018. Disponível em: <https://www.mg.superesportes.com.br/app/noticias/futebol/futebol-nacional/2018/05/02/noticia_futebol_nacional,470883/consultoria-analisa-dados-financeiros-dos-clubes-brasileiros.shtml>.

[2] LUX, A. "Como funcionam os direitos de transmissão do futebol brasileiro e o impacto disso para o torcedor", *Ludopédio*, fev. 2018. Disponível em: <https://www.ludopedio.com.br/arquibancada/direitos-de-transmissao/>.

[3] KENDALL, G., KNUST, S., RIBEIRO, C. C., et al. "Scheduling in Sports: An annotated Bibliography", *Computers & Operations Research*, v. 37, n. 1, pp. 1–19, jan. 2010.

[4] NEMHAUSER, G. L., TRICK, M. A. "Scheduling a Major College Basketball Conference", *Operations Research*, v. 46, n. 1, pp. 1–8, fev. 1998.

[5] NORONHA, T. F., RIBEIRO, C. C., DURAN, G., et al. "A Branch-and-Cut Algorithm for Scheduling the Highly-Constrained Chilean Soccer Tournament". In: *Practice and Theory of Automated Timetabling VI*, pp. 174–186, Czech Republic, ago. 2006.

[6] KIRKMAN, T. P. "On a problem in combinations". In: *The Cambridge and Dublin Mathematical Journal*, pp. 191–223, 1847.

[7] EASTON, K., NEMHAUSER, G., TRICK, M. A. "The Traveling Tournament Problem Description and Benchmarks". In: *Principles and Practice of Constraint Programming — CP 2001*, pp. 580–584, Cyprus, nov. 2001.

[8] RIBEIRO, C. C., URRUTIA, S. "Heuristics for the Mirrored Traveling Tournament Problem". In: *European Journal of Operational Research*, v. 179, pp. 775–787, jun. 2007.

[9] MELO, R. A., URRUTIA, S., RIBEIRO, C. C. "The traveling tournament problem with predefined venues", *Journal of Scheduling*, v. 12, n. 607, dez. 2009.

[10] URRUTIA, S., RIBEIRO, C., MELO, R. "A New Lower Bound to the Traveling Tournament Problem", *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling, CI-Sched 2007*, pp. 15 – 18, 05 2007. doi: 10.1109/SCIS.2007.367664.

[11] BENOIST, T., LABURTHE, F., ROTTEMBOURG, B. "Lagrange Relaxation and Constraint Programming Collaborative schemes for Traveling Tournament Problems", *Proceedings of CP-AI-OR'01*, 2001.

[12] EASTON, K., NEMHAUSER, G., TRICK, M. "Solving the Travelling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach". In: *Practice and Theory of Automated Timetabling IV*, pp. 100–112, 08 2002. doi: 10.1007/978-3-540-45157-0_6.

[13] ANAGNOSTOPOULOS, A., MICHEL, L., HENTENRYCK, P. V., et al. "A simulated annealing approach to the traveling tournament problem", *Journal of Scheduling*, v. 9, n. 2, pp. 177–193, Apr 2006.

[14] DI GASPERO, L., SCHAERF, A. "A Tabu Search Approach to the Traveling Tournament Problem". In: *Proceedings of the 6th Metaheuristics International Conference (MIC-2005)*, 01 2005.

[15] LIM, A., RODRIGUES, B., ZHANG, X. "A simulated annealing and hill-climbing algorithm for the traveling tournament problem", *European Journal of Operational Research*, v. 174, pp. 1459–1478, 11 2006. doi: 10.1016/j.ejor.2005.02.065.

[16] DI GASPERO, L., SCHAERF, A. "A Composite-Neighborhood Tabu Search Approach to the Traveling Tournament Problem", *Journal of Heuristics*, v. 13, pp. 189–207, 02 2007. doi: 10.1007/s10732-006-9007-x.

[17] LEE, J. H., LEE, Y. H., LEE, Y. H. "Mathematical Modeling and Tabu Search Heuristic for the Traveling Tournament Problem". In: Gavrilova, M., Gervasi, O., Kumar, V., et al. (Eds.), *Computational Science and Its Applications - ICCSA 2006*, pp. 875–884, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[18] URRUTIA, S., RIBEIRO, C. C. "Maximizing breaks and bounding solutions to the mirrored traveling tournament problem", *Discrete Applied Mathe-*

matics, v. 154, n. 13, pp. 1932 – 1938, 2006. ISSN: 0166-218X. doi: https://doi.org/10.1016/j.dam.2006.03.030.

[19] FUJIWARA, N., IMAHORI, S., MATSUI, T., et al. "Constructive Algorithms for the Constant Distance Traveling Tournament Problem". In: *Practice and Theory of Automated Timetabling VI*, v. 3867, pp. 135–146, 2007. doi: 10.1007/978-3-540-77345-0_9.

[20] DE WERRA, D. "Geography, games and graphs", *Discrete Applied Mathematics*, v. 2, n. 4, pp. 327 – 337, 1980. ISSN: 0166-218X. doi: https://doi.org/10.1016/0166-218X(80)90028-1.

[21] CHEUNG, K. "Solving mirrored traveling tournament problem benchmark instances with eight teams", *Discrete Optimization*, v. 5, n. 1, pp. 138 – 143, 2008. ISSN: 1572-5286. doi: https://doi.org/10.1016/j.disopt.2007.11.003.

[22] IRNICH, S. "A new branch-and-price algorithm for the traveling tournament problem", *European Journal of Operational Research*, v. 204, n. 2, pp. 218 – 228, 2010. ISSN: 0377-2217. doi: https://doi.org/10.1016/j.ejor.2009.10.024.

[23] C. UTHUS, D., RIDDLE, P., GUESGEN, H. "DFS* and the Traveling Tournament Problem". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, v. 5547, pp. 279–293, 05 2009. doi: 10.1007/978-3-642-01929-6_21.

[24] MISIR, M., WAUTERS, T., VERBEECK, K., et al. "A new learning hyper-heuristic for the traveling tournament problem", *8th Metaheuristic International Conference (MIC'09)*, 02 2009.

[25] MIYASHIRO, R., MATSUI, T., IMAHORI, S. "An approximation algorithm for the traveling tournament problem", *Annals of Operations Research*, v. 194, n. 1, pp. 317–324, Apr 2012. ISSN: 1572-9338. doi: 10.1007/s10479-010-0742-x.

[26] THIELEN, C., WESTPHAL, S. "Complexity of the traveling tournament problem", *Theor. Comput. Sci.*, v. 412, pp. 345–351, 02 2011. doi: 10.1016/j.tcs.2010.10.001.

[27] C. UTHUS, D., RIDDLE, P., GUESGEN, H. "Solving the traveling tournament problem with iterative-deepening A*", *Journal of Scheduling - SCHEDULING*, v. 15, pp. 1–14, 10 2012. doi: 10.1007/s10951-011-0237-x.

[28] GOERIGK, M., WESTPHAL, S. "A combined local search and integer programming approach to the traveling tournament problem", *Annals of Operations Research*, v. 239, n. 1, pp. 343–354, Apr 2016. ISSN: 1572-9338. doi: 10.1007/s10479-014-1586-6.

[29] WESTPHAL, S., NOPARLIK, K. "A 5.875-approximation for the Traveling Tournament Problem", *Annals of Operations Research*, v. 218, pp. 1–14, 01 2010. doi: 10.1007/s10479-012-1061-1.

[30] GOERIGK, M., KAWARABAYASHI, K.-I., HOSHINO, R., et al. "Solving the traveling tournament problem by packing three-vertex paths", *Proceedings of the National Conference on Artificial Intelligence*, v. 3, pp. 2271–2277, 01 2014.

[31] WOLSEY, L. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998. ISBN: 9780471283669.

[32] FUKASAWA, R., LONGO, H., LYSGAARD, J., et al. "Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem", *Math. Program.*, v. 106, pp. 491–511, 07 2006. doi: 10.1007/s10107-005-0644-x.

[33] POGGI, M., UCHOA, E. "Chapter 3: New Exact Algorithms for the Capacitated Vehicle Routing Problem". In: *Vehicle Routing Problems, Methods, and Applications*, pp. 59–86, 11 2014. ISBN: 978-1-61197-358-7. doi: 10.1137/1.9781611973594.ch3.

[34] RIGHINI, G., SALANI, M. "Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints", *Discrete Optimization*, v. 3, pp. 255–273, 09 2006. doi: 10.1016/j.disopt.2006.05.007.

[35] CHRISTOFIDES, N., MINGOZZI, A., TOTH, P. "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations", *Mathematical Programming*, v. 20, n. 1, pp. 255–282, Dec 1981. ISSN: 1436-4646. doi: 10.1007/BF01589353.

[36] BALDACCI, R., MINGOZZI, A., ROBERTI, R. "New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem", *Operations Research*, v. 59, pp. 1269–1283, 09 2011. doi: 10.1287/opre.1110.0975.

[37] DU MERLE, O., VILLENEUVE, D., DESROSIERS, J., et al. "Stabilized column generation", *Discrete Mathematics*, v. 194, n. 1, pp. 229 – 237, 1999. ISSN: 0012-365X. doi: https://doi.org/10.1016/S0012-365X(98)00213-1.

[38] PESSOA, A., SADYKOV, R., UCHOA, E., et al. "Automation and Combination of Linear-Programming Based Stabilization Techniques in Column Generation", *INFORMS J. on Computing*, v. 30, n. 2, pp. 339–360, maio 2018. ISSN: 1526-5528. doi: 10.1287/ijoc.2017.0784. Disponível em: <https://doi.org/10.1287/ijoc.2017.0784>.