André Gustavo dos Santos

Método de Geração de Colunas e Meta-heurísticas para Alocação de Tripulação

Tese apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para obtenção do título de Doutor em Ciência da Computação

Belo Horizonte

Outubro de 2008

Resumo

Em problemas de alocação de tripulação, a cada membro da tripulação é designada uma jornada de trabalho, composta por uma seqüência de viagens a serem cobertas. O objetivo é selecionar as jornadas de forma a cobrir todas as viagens com custo operacional total mínimo. Embora haja várias restrições na combinação das viagens para a formação de uma jornada de trabalho, a quantidade total de jornadas viáveis possíveis é geralmente muito grande. Para instâncias pequenas é possível enumerar todas as jornadas viáveis, e encontrar a solução ótima resolvendo-se um problema de particionamento ou de cobrimento de conjuntos. Mas esta abordagem exata consome bastante tempo e memória para instâncias baseadas em dados reais. Por outro lado, soluções heurísticas podem se perder no grande espaço de soluções. Já que a solução ótima geralmente contém apenas um número bastante reduzido de jornadas, o método de geração de colunas parece ser a opção ideal. O problema é então decomposto em um problema mestre e um subproblema, responsáveis respectivamente por selecionar o melhor conjunto de jornadas, e gerar novas jornadas. O foco principal deste trabalho é a solução do subproblema, pois embora ambos sejam NP-hard, o problema mestre pode ser rapidamente resolvido pelos atuais pacotes de programação linear inteira, desde que o número de jornadas consideradas não seja grande, o que normalmente acontece quando se usa geração de colunas. Neste trabalho são usadas uma heurística gulosa e as meta-heurísticas Grasp e Genético para acelerar a geração de colunas, assegurando-se a otimalidade da solução combinando essas heurísticas com programação linear inteira. São reportados resultados computacionais para dados da literatura e de instâncias reais, mostrando que esse método híbrido pode resolver de forma exata problemas de alocação de tripulação mais rapidamente que utilizando-se apenas métodos exatos.

Abstract

In a typical crew scheduling problem, for each crew member is assigned a set of trips (a duty) to be performed. The objective is to select the duties such as the total operational cost is minimized, and no trip is left uncovered. Although there are some constraints about how the trips may be combined in a feasible duty, the total number of feasible duties is generally large. For small instances it is possible to enumerate all feasible duties, and find the optimal solution solving a set partitioning or set covering problem. However, this approach may be time and memory consuming for large instances. On the other hand, some heuristics may be lost in the huge feasible space. As the optimal solution contains only few duties, column generation seems to be the best approach. The problem is then decomposed in master and subproblem, whose objectives are respectively select the best set of duties and generate new duties. The main focus of this work is about solving the subproblem, because, although both problems are NP-hard, the master problem can be quickly solved by linear programming packages if the number of duties is not so big, which usually happens in a column generation approach. In this work Greedy, Grasp and Genetic metaheuristics are used to speed up the column generation approach, assuring optimality by combining them with integer linear programming. Computational results for both real instances and instances from the literature are reported, showing that this hybrid method can solve crew scheduling problems to optimality quicker than using only exact methods.

Agradecimentos

Agradeço a Deus pela oportunidade de estudo, por ter me sustentado durante o doutorado, ter me guardado em uma centena de viagens a Belo Horizonte, e me fortalecido em meio a alguns contratempos ocorridos nesse meio tempo. Sem esse cuidado certamente eu não chegaria aqui.

Agradeço muito ao prof. Robson por me aceitar no curso mesmo com tempo parcial, acreditar no trabalho, dar direções em momentos cruciais, quando tudo parecia já bastante explorado em outros trabalhos, e pela paciência em continuar a orientação mesmo que eu estivesse o tempo inteiro a quilômetros de distância.

Agradeço ao DPI – UFV pela liberação de parte da carga horária semanal de trabalho para me dedicar ao doutorado, o que sem dúvida foi um grande incentivo à conclusão do curso. Da mesma forma, agradeço à FIC, instituição que me incentivou e apoiou no início do doutorado, pois mesmo sendo professor e coordenador do curso de Ciência da Computação na instituição, os dias liberados durante a semana para cursar as disciplinas do doutorado foram sem dúvida, essenciais no início do curso.

Agradeço à CAPES pela bolsa de doutorado sanduíche que me foi concedida de janeiro a julho de 2006, e ao professor Alberto Caprara, no DEIS – Università de Bologna, Italia, por ter me recebido em seu departamento, ter dedicado parte do seu tempo como meu orientador no exterior, e ter contribuído de forma substancial para meu aprendizado da técnica de geração de colunas durante o início do desenvolvimento desse trabalho.

Agradeço aos amigos latino-americanos, Albert Muritiba, Andrés Saldaña, Felipe Navarro, Rosa Medina, Pedro Piñeyro, Sebastián Alaggia e Victor Vera Valdes, bolsistas do projeto Alfa em Bolonha durante meu tempo no DEIS, companheiros de sala na universidade e de passeios nos dias livres, que tornaram minha estadia na Itália mais agradável e divertida.

Também agradeço aos italianos do DEIS, com quem aprendi bastante da cultura italiana, e que estavam sempre presentes na luta diária de estudo e trabalho, meus companheiros de almoço e laboratório Valentina Cacchiani, Cláudia D'Ambrosio, Laura Galli, Andrea Tramontani e Michele Monaci, e em especial Enrico Malaguti e Manuel Iori, por estenderem essa amizade fora dos limites da universidade, algo excelente quando se está longe dos amigos e da família.

Agradeço também aos professores Abílio, Cid e Maurício pelas valiosas contribuições na apresentação da proposta de tese, várias delas incorporadas no trabalho, e também pelas suas sugestões, juntamente com o professor Nelson Maculan, para a melhoria da versão final deste trabalho.

E de forma muito especial agradeço à minha família! Das pessoas aqui citadas são as que eu menos conversei a respeito do meu trabalho, mas certamente são as que mais torceram pela sua conclusão. O carinho e a segurança que me proporcionaram trouxeram grande ânimo para chegar aqui.

Sumário

Índice de Figuras

Índice de Tabelas

1. Introdução

Em companhias de transporte de massa, decisões devem ser tomadas a todo instante: planejamento das rotas; alocação, escalonamento, manutenção e renovação da frota de veículos; contratação, treinamento, e alocação da tripulação; replanejamento diante de imprevistos, atrasos, condições climáticas, alteração de demanda; e tantas outras. Tais tomadas de decisão variam em diversos aspectos: desde planejamento em longo prazo, como planejamento de rotas, ou em curto prazo, como escala semanal de trabalho de um membro da tripulação; desde decisões pontuais, como treinar um tripulante para uma nova função, como decisões mais amplas, como gerar escalas mensais de trabalho para todos os funcionários, incluindo dias de folga, férias, e preferências pessoais; desde decisões que podem ser cuidadosamente planejadas durante certo tempo, como aquisição de novos veículos, àquelas que devem ser tomadas de forma urgente, como replanejamento e substituição de veículos e tripulantes, devido a imprevistos, como falha no veículo ou problemas de saúde de um tripulante.

Praticamente todos esses problemas contam com a ajuda da Pesquisa Operacional para otimização da utilização dos recursos, com o objetivo de vencer a concorrência em um mercado tão competitivo. Em alguns setores e localidades, cada empresa detém o monopólio de algumas rotas, devendo apenas planejar a utilização de seus recursos para cobrir suas rotas, o que já não é fácil de conseguir. Porém, na maioria dos casos, mais de uma empresa atende a mesma região, competindo entre si na busca de seus clientes, os usuários dos meios de transporte. Alguns usuários dão preferência a empresas que apresentam regularidade e pontualidade em seus horários de viagem, outros optam por conforto e benefícios concedidos, e ainda outros escolhem as de custos mais baixos. Existem empresas que buscam atender a todos estes tipos de clientes, e oferecem então transporte dividido em categorias: econômica, executiva e primeira classe, no caso de aviões; regular, executivo, leito, e outros, no caso de ônibus. Em todos os casos, um cuidadoso planejamento é feito para se conseguir uma fatia no mercado, e ainda para manter ou aumentar essa fatia.

Dentre todas estas tomadas de decisões, este trabalho trata especificamente do problema de alocação da tripulação. Considera-se que todas as rotas já foram planejadas, bem como todas as viagens que compõem cada rota, seus pontos de repouso e troca de tripulação, e ainda seus horários e freqüências. Considera-se também que os veículos já foram associados a cada viagem, de forma que já se tem conhecimento de quais os funcionários são habilitados para cada viagem, de acordo com sua experiência no tipo de veículo utilizado. De posse de todos os dados, o problema consiste em se determinar a escala de trabalho de cada tripulante, definindo sua jornada de trabalho, ou seja, que tarefas ele deve cumprir em um determinado dia de trabalho. Trabalhos encontrados na literatura tratam o problema de alocação de tripulação tanto em malhas aéreas (Klabjan, Johnson, Nemhauser, Gelman, & Ramaswamy, 2001), (Cohn & Barnhart, 2003) (Ehrgott & Ryan, 2006) quanto em ferroviárias (Caprara, Toth, Vigo, & Fischetti, 1998) (Kwan, Wren, & Kwan, 2000) e em rodoviárias (Fores, Proll, & Wren, 1996) (Lourenço, Paixão, & Portugal, 2001) (Wren & Wren, 1995).

Na literatura este tema é tratado sob dois nomes diferentes, que correspondem a dois horizontes de planejamento: *crew scheduling*, que consiste em se criar um conjunto de jornadas de trabalho para os tripulantes, de tal forma a cobrir todas as tarefas que devem ser realizadas em um determinado dia; e *crew rostering*, que consiste em se distribuir as jornadas de trabalho aos tripulantes ao longo da semana, ou do mês, considerando seus dias de folga e férias, de tal forma haja algum tripulante associado a cada uma das jornadas que devem ser cumpridas em cada dia. Este trabalho trata apenas o problema *crew scheduling*, embora as mesmas idéias, algoritmos e heurísticas, possam ser adaptados ao *crew rostering*.

Existe uma série de regras que definem as normas de regulamentação do trabalho dos tripulantes, diferentes de um meio de transporte para outro, de uma região para outra, e até mesmo de uma empresa para outra, já que além da regulamentação sindical existem normas internas de funcionamento da empresa. De uma forma geral, existe um número máximo de horas de trabalho diário e semanal, um número mínimo de dias de descanso semanal e mensal, tempo mínimo entre o fim de uma jornada de trabalho e o início da jornada seguinte, tempo mínimo de descanso dentro de uma jornada de trabalho, dentre outras regras. Pode-se então definir estratégias e algoritmos gerais para o tratamento dos problemas de *crew scheduling* e *crew rostering*, e se necessário, especializar estas estratégias e algoritmos incluindo dados específicos do problema tratado, para melhorar seu desempenho.

Se todas as possíveis alternativas de jornadas de trabalho são conhecidas, o problema consiste em se selecionar, desse conjunto, uma para cada tripulante, de tal forma que todas as tarefas sejam cumpridas por algum dos tripulantes. Ou seja, o problema se resume a selecionar um conjunto de jornadas de trabalho que cubra todas as tarefas, com custo operacional total mínimo. Esse problema é conhecido como *set covering problem* (SCP), em que há um conjunto de linhas a serem cobertas, e um conjunto de colunas com custos associados, cada uma cobrindo um subconjunto das linhas. O objetivo é selecionar um subconjunto de colunas que cubra todas as linhas, a custo mínimo. SCP é um problema NP-Completo (Garey & Johnson, 1979) extensamente estudado na literatura.

Existem alguns artigos que tratam das propriedades teóricas do SCP, como estudos sobre o politopo do problema, e caracterização e geração de facetas deste politopo (Balas & Ng, 1989) (Cornuéjols & Sassano, 1989). Algoritmos exatos geralmente envolvem melhorias na aplicação do *branch-and-bound* (Balas & Ho, 1980) (Beasley, 1987) (Fisher & Kedia, 1990). Este último mostra a solução de problemas de tamanho até 200 linhas por 2000 colunas, utilizando heurísticas duais. Beasley melhorou o algoritmo anteriormente publicado, utilizando *branch-and-bound* com relaxação lagrangeana, restrições de exclusão e cortes clássicos de Gomory, resolvendo problemas com até 400 linhas por 4000 colunas (Beasley & Jornstein, 1992). Já em (Balas, Ceria, & Cornuéjols, 1996) utilize-se *branch-and-cut* combinado com *lift-and-project*. Algoritmos exatos foram então usados com sucesso em problemas com até centenas de linhas e poucos milhares de colunas. Mas, na prática, problemas bem maiores ocorrem, e por isso várias heurísticas foram propostas.

As heurísticas mais simples envolvem algoritmos gulosos (Balas & Ho, 1980). São algoritmos rápidos, porém com soluções de baixa qualidade. Podem servir como solução inicial para outras heurísticas que necessitam partir de alguma solução viável. Melhores heurísticas

envolvem planos de corte, otimização por subgradiente, e relaxação lagrangeana (Vasko & Wilson, 1984) (Beasley, 1990), apresentando soluções satisfatórias em problemas com até 1000 linhas por 10000 colunas. Também já foram usadas várias meta-heurísticas, como *simulated annealing* (Sen, 1993) e algoritmos genéticos (Beasley & Chu, 1996). Em sua forma multiobjetivo, (Jaszkiewicz, 2004) compara os resultados de dez heurísticas e meta-heurísticas, entre elas algoritmos genéticos, meméticos e *simulated annealing*, e (Carvalho, Mateus, & Santos, 2005) comparam esses resultados com outro algoritmo genético em problemas de alocação de tripulação multiobjetivo.

Na solução do SCP, se alguma viagem é coberta por mais de uma jornada selecionada como solução, parte da tripulação viaja como passageiro, enquanto outra está cobrindo a tarefa. Desta forma economizam-se custos de hospedagem e uma tripulação pode iniciar o trabalho em um local diferente daquele onde havia terminado a última tarefa, mesmo que esteja a milhas de distância. Isso é comum em empresas aéreas, e é chamado de *deadhead*. Porém não é indicado para empresas rodoviárias, porque se a distância for longa, grande parte do horário de trabalho da tripulação seria consumida em deslocamentos. Nestes casos, é mais apropriado resolver o problema de alocação utilizando-se o *set partitioning problem* (SPP). O SPP é semelhante ao SCP, porém cada linha deve ser coberta exatamente por uma coluna. No caso da alocação de tripulações, cada jornada de trabalho é assumida por um tripulante, ou uma equipe única, não havendo a dupla cobertura permitida pelo SCP. Resolver o SPP pode ser mais complicado que o SCP correspondente. Note que toda solução viável do SPP é solução viável do SCP. Porém a solução ótima do SCP pode até mesmo não ser viável no SPP correspondente. Entretanto, as mesmas técnicas de solução geralmente podem ser aplicadas em ambos os problemas, sendo mais difícil manter a viabilidade da solução quando se trabalha com o SPP, degradando assim o desempenho dos algoritmos. Como na maioria das instâncias utilizadas nesse trabalho não são permitidos *deadheads*, o modelo utilizado é o *set partitioning problem*, discutido no próximo capítulo.

Mesmo sendo problemas de difícil solução, SCP e SPP podem ser usados quando todas as jornadas são conhecidas, já que os atuais pacotes de programação linear inteira, como Xpress (Dash Optimization Ltd, 2006) e Cplex (ILOG S.A, 2005) conseguem resolvê-los em tempo razoável, desde que as instâncias não sejam tão grandes. Porém, nem sempre o conjunto completo de alternativas de jornadas de trabalho é conhecido a priori, e deve, portanto, ser gerado. Em aplicações reais, esse conjunto é bastante grande, devido à natureza combinatória do problema, e um tempo considerável seria gasto na sua geração. Além disso, mesmo que gerado completamente, pode ser difícil a aplicação de uma solução direta, pela dificuldade de acomodar todas as jornadas em memória principal. Nesses casos é mais aconselhável o uso da técnica de geração de colunas, que gera jornadas de trabalho enquanto resolve o problema de seleção das jornadas. A principal vantagem é que a solução ótima pode ser obtida sem se gerar explicitamente todo o conjunto de jornadas. Os detalhes de funcionamento desta técnica, e sua aplicação nesse trabalho, são apresentados no capítulo 3, onde se mostra a decomposição do problema em um problema mestre (o próprio SPP mas com um número reduzido de colunas), e um subproblema, cujo objetivo é gerar novas colunas, nesse caso novas jornadas de trabalho viáveis.

Esse subproblema é resolvido várias vezes durante a geração de colunas, para gerar várias jornadas de trabalho, e é também um problema de otimização, podendo ser resolvido por programação linear inteira. Como sua solução exata para instâncias reais se mostrou lenta, são propostas algumas heurísticas para resolvê-lo de forma aproximada, e assim acelerar a geração de novas jornadas. Entretanto, não sendo resolvido de forma exata, não seria possível garantir a otimalidade da solução do problema SPP, pois não se saberia se todas as jornadas interessantes foram geradas. Por isso, foi feito um algoritmo híbrido, que combina as heurísticas com a geração exata de novas jornadas, sendo esta executada quando a heurística não conseguir produzir novas jornadas interessantes. Os métodos heurísticos são descritos no capítulo 4, e são baseados na relaxação linear de um modelo matemático construído para o subproblema, em algoritmos genéticos, e GRASP. Nesse capítulo também é descrita a forma como essas heurísticas são incorporadas no método de geração de colunas para construir um método exato, porém mais eficiente em termos de tempo de execução.

Para usar a técnica de geração de colunas, é resolvida na verdade a relaxação linear do SPP, o que pode produzir uma solução fracionária, em que uma jornada é apenas parcialmente selecionada. A geração de colunas é então combinada com a técnica de *branch-and-bound* produzindo um método *branch-and-price*, descrito no capítulo 5.

Os objetivos do trabalho podem ser resumidos e divididos em quatro itens: (i) utilização de heurísticas e meta-heurísticas para acelerar a solução do problema de alocação de tripulações; (ii) incorporar essas heurísticas em um algoritmo híbrido que garanta a otimalidade da solução encontrada; (iii) produzir um algoritmo geral, que possa ser especializado para tratar vários problemas de alocação de tripulação, apenas acrescentando as características específicas das instâncias consideradas; e (iv) aplicar o algoritmo a instâncias da literatura e instâncias reais.

Resultados completos e uma análise do desempenho dos vários métodos descritos são reportados no capítulo 6, mostrando as vantagens trazidas pelo uso das heurísticas combinadas com os métodos exatos. E finalmente, o capítulo 7 apresenta as conclusões do trabalho, apontando ainda algumas direções de trabalhos futuros para sua extensão.

2. Problema de Alocação de Tripulações

A alocação de tripulações é um dos mais importantes problemas de decisão em companhias aéreas, ferroviárias, e rodoviárias. Uma razão é que os gastos com a tripulação representam um dos fatores preponderantes nas despesas totais das companhias, junto aos gastos com combustíveis (Gopalakrishnan & Johnson, 2005) (Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998) (Cabral, Souza, Maculan, & Pontes, 2000). Assim, qualquer mínima redução percentual representa uma economia substancial para essas companhias. Outra razão é que, embora geralmente fácil de construir um modelo para o problema e interpretá-lo matematicamente como um problema de otimização combinatória, resolver esse modelo de forma eficiente é um desafio.

De uma forma geral, o problema de alocação de tripulações consiste em se criar e designar jornadas de trabalho para cada membro da tripulação, cobrindo todas as viagens que devem ser realizadas. Uma viagem é uma tarefa que algum membro da tripulação deve cobrir, como conduzir uma veículo de uma estação para outra. Cada viagem tem uma duração, durante a qual o tripulante a ela associado não pode ser designado a cumprir nenhuma outra viagem. Uma jornada de trabalho é uma seqüência de viagens, seqüência essa que deve obedecer a uma série de restrições impostas por sindicatos, leis governamentais e acordos trabalhistas entre a companhia e seus funcionários. Restrições típicas de uma jornada de trabalho incluem, entre outras, dependendo do meio de transporte e da companhia em questão, um limite máximo na duração total da jornada, um tempo mínimo de descanso entre as viagens. A cada jornada é associado um custo operacional, e o principal objetivo é encontrar um conjunto de jornadas com um custo total mínimo, de tal forma que se cada jornada for designada a um tripulante, todas as viagens são cobertas por apenas um tripulante, ou equipe. Em certas instâncias tratadas na literatura, essa última restrição é relaxada, de tal forma que a tarefa possa ser coberta por mais de um tripulante. Nesses casos, um deles não trabalha efetivamente, mas viaja como passageiro, e isso de certa forma pode contar em seu tempo de trabalho.

Embora a criação das jornadas dependa muito das instâncias consideradas, o processo de seleção das melhores jornadas é bastante semelhante em todos os meios de transporte e companhias, podendo ser modelado como *set covering problem* (SCP) ou *set partitioning problem* (SPP), dependendo se uma viagem pode ou não ser coberta mais de uma vez. Por isso, muitos dos trabalhos envolvendo o problema de alocação de tripulações tratam de um desses problemas, utilizando as mais variadas técnicas de solução exata, como enumeração (Beasley & Cao, 1996) e geração de colunas (Souza, Moura, & Yunes, 2005) (Desrochers & Soumis, 1989) e também heurísticas (Beasley & Chu, 1996) (Caprara, Fischetti, & Toth, 1999). Isso pode ser comprovado analisando-se os levantamentos de trabalhos feitos por alguns pesquisadores (Kohl & Karisch, 2004) (Gopalakrishnan & Johnson, 2005), em que grande parte dos trabalhos envolve técnicas para resolver o SPP ou SCP, ou algum outro modelo semelhante, e poucos tratam da construção das jornadas, ou seja, na geração dos dados de entrada para o SPP. Em alguns casos supõe-se que todas as jornadas são conhecidas, em outros elas são geradas por enumeração exaustiva, e em outros elas são geradas dentro de um algoritmo de geração de colunas. Este trabalho aborda as duas fases, a seleção e a geração de

jornadas, porém trata de forma especial a fase específica de criação das jornadas de trabalho, dentro de uma abordagem de geração de colunas. Embora seja uma fase muito dependente das instâncias consideradas, a utilização de uma modelagem em grafos com recursos associados às restrições (Desaulniers, Desrosiers, & Solomon, 2005) simplifica a aplicação em vários contextos, e torna a técnica aqui descrita geral o suficiente para praticamente qualquer problema de alocação de tripulações.

Assim, a técnica aqui descrita é geral, e pode ser aplicada praticamente a qualquer instância do problema. Isso porque todos os métodos, exatos e heurísticos, usam uma modelagem do problema em um grafo, e as características e restrições específicas das instâncias tratadas podem ser utilizadas diretamente na construção desse grafo. Apesar disso, a técnica não foi aplicada em diversas instâncias por causa da dificuldade em se obter instâncias reais para o problema. As companhias não fornecem facilmente os dados utilizados na alocação de sua tripulação porque muitas vezes são dados estratégicos que não podem ser divulgados. O mesmo ocorre com as instâncias utilizadas em vários trabalhos já publicados, que são oferecidas em contratos de sigilo e por isso não podem ser testadas por outros pesquisadores. Por esse motivo, os métodos propostos no trabalho são aplicados a três tipos de instâncias, que são detalhados no capítulo seguinte. Um é retirada da biblioteca de instâncias OR-Library (Beasley, OR-Library), em que o único requisito na solução, além da não sobreposição de tempo das tarefas numa mesma jornada de trabalho, é o limite máximo de duração das jornadas, não sendo portanto baseadas em dados reais. Outras instâncias são baseadas em dados reais de uma empresa de transporte coletivo de Belo Horizonte, e foram fornecidos por autores de um trabalho na área (Marinho, 2005). E ainda outras foram criadas a partir de horários de vôos disponíveis no site de uma companhia aérea nacional.

2.1. Problema de Transporte Urbano

Para exemplificar as restrições que tipicamente aparecem na construção das jornadas de trabalho, são detalhadas a seguir algumas das regras das instâncias de transporte urbano usadas nesse trabalho. As instâncias utilizadas trazem dados de uma empresa de transporte público da cidade de Belo Horizonte. Existem outros trabalhos que usam as mesmas instâncias, com técnicas diferentes, como Busca Tabu com várias estratégias para diversificar e intensificar as buscas na vizinhança (Marinho, 2005) (Gonçalves, Santos, & Silva, 2008). O mesmo problema, embora com instâncias diferentes, foi também tratado em (Souza, Moura, & Yunes, 2005) com geração de colunas, utilizando programação por restrições para a geração de novas jornadas. As regras que indicam a viabilidade de uma jornada de trabalho são descritas a seguir, e podem ser encontradas com maiores detalhes nesses trabalhos citados.

Há várias empresas que fazem o serviço de transporte público em Belo Horizonte, cada uma com uma ou mais linhas de ônibus, mas todas seguem as mesmas regras para formação das jornadas de trabalho, estabelecidas na Convenção Coletiva de Trabalho 2002/2003, e fiscalizadas pela BHTrans. É apresentado aqui apenas um resumo das principais regras. A regulamentação da BHTrans permite que a tripulação troque de veículo ou de estações de início e de fim das tarefas, durante sua jornada de trabalho, em casos detalhados a seguir.

Trocas de linha são permitidas apenas entre alguns itinerários, geralmente controlados pela mesma empresa e com finais de rota próximos, mas como estes dados não foram fornecidos, nesse trabalho considera-se que nenhuma troca de linha é permitida. Sendo assim, o problema pode ser resolvido separadamente para cada linha.

Antes de se fazer a alocação de tripulação, as viagens são agrupadas em blocos, que são seqüências de viagens cujos intervalos entre viagens consecutivas sejam de no máximo 5 minutos. Intervalos maiores que 5 minutos caracterizam pontos de troca, onde a tripulação pode ser substituída por outra. Nesse trabalho, denomina-se tarefa a menor unidade de trabalho que não pode ser dividida entre tripulações diferentes. Assim, as tarefas constituem um bloco com uma ou mais viagens, que devem ser conduzidas por uma mesma tripulação.

As jornadas de trabalho podem ser de dois tipos: jornada simples ou de dupla pegada. As jornadas de trabalho do tipo dupla pegada são caracterizadas por um, e apenas um, intervalo entre tarefas com duração igual ou superior a 2 horas. Daí deriva o nome dupla pegada, pois a tripulação realizada o trabalho em dois momentos diferentes. Para cada linha de ônibus são permitidas algumas jornadas do tipo dupla pegada sem custo adicional para a empresa. Jornadas do tipo dupla pegada além desse limite são penalizadas na contagem final dos custos operacionais.

O tempo máximo de trabalho é de 6 horas e 40 minutos, sendo permitidas até 2 horas extras além desse período. Para jornadas simples, são exigidos ainda pelo menos 30 minutos de descanso, elevando o tempo da jornada para 7 horas e 10 minutos. Esse tempo ocioso pode estar distribuído ao longo da jornada, inclusive no fim da mesma, sendo contadas inclusive as folgas dentro de uma tarefa (intervalos entre viagens do bloco), desde que pelo menos um desses momentos seja de pelo menos 15 minutos ininterruptos. Ressalta-se que isso vale apenas para jornadas simples; jornadas do tipo dupla pegada já possuem um intervalo longo, e essas restrições de descanso não são consideradas.

No intervalo de 2 horas de uma dupla pegada é permitido que a tarefa seguinte inicie em uma estação diversa da anterior, pois há tempo suficiente para a tripulação se mover de uma estação para outra. Em todos os demais casos, uma tarefa deve começar no ponto onde foi concluída a tarefa anterior. Trocas de veículos são permitidas em qualquer caso, porém sempre penalizadas.

A Tabela 1 mostra um exemplo contendo dados de parte da entrada. Nesse exemplo nenhuma das tarefas possui folga entre as viagens do bloco, logo todos os intervalos de descanso que existirem nas jornadas são entre uma tarefa e outra. As tarefas são distribuídas ao longo do dia em dois veículos diferentes. Algumas tarefas correspondem a retirar ou levar o ônibus para a garagem (estação inicial ou final igual a 0), mas a maioria é do tipo circular, inicia em um ponto e termina no mesmo ponto.

A Tabela 2 e a Tabela 3 mostram possíveis soluções viáveis para esse exemplo. A primeira, considerando-se que todas as jornadas do tipo dupla pegada têm custo alto para a empresa, e a segunda considerando que a empresa pode utilizar duas jornadas desse tipo sem custo adicional. A segunda solução possui duas jornadas do tipo dupla pegada (assinaladas com *), economizando 32 minutos de trabalho total, e 1:37 de tempo ocioso, com o custo de

se aumentar 28 minutos de hora extra em relação à primeira solução, sem jornadas do tipo dupla pegada. Qual solução é melhor depende dos custos envolvidos no problema, como pagamento de hora extra e jornadas do tipo dupla pegada.

Tabela 1: Dados de parte das tarefas da instância L1170-54

Tarefa	Veículo	Horário Inicial	Horário Final	Estação Inicial	Estação Final	Folga
1	9	345	410	0	4	0
2	10	365	430	0	4	0
3	9	420	480	4	4	0
4	10	435	495	4	4	0
5	9	495	561	4	4	0
6	10	510	578	4	4	0
7	9	570	640	4	4	0
8	10	590	660	4	4	0
9	9	650	710	4	4	0
10	10	665	725	4	4	0
11	9	730	790	4	4	0
12	10	745	805	4	4	0
13	10	830	910	4	4	0
14	10	930	1006	4	4	0
15	10	1020	1080	4	4	0
16	10	1105	1165	4	4	0
17	10	1190	1240	4	4	0
18	10	1250	1315	4	0	0

Tabela 2: uma possível solução sem usar jornadas do tipo dupla pegada

Jornada	Tarefas	Duração	Tempo Ocioso	Hora extra	Troca de veículo
1	1 3 5 7 9 11	7:25	1:04	0:15	0
2	2 4 6 8 10 12	7:20	1:02	0:10	0
3	13 14 15 16 17 18	8:05	1:34	0:55	0
Total		22:50	6:05	1:20	0

Tabela 3: uma possível solução utilizando dupla pegada

Jornada	Tarefas	Duração	Tempo Ocioso	Hora extra	Troca de veículo
1	1 3 5 7 9 11	7:25	1:04	0:15	0
2	2 4 6 * 14 15 16	7:28	0:59	0:48	0
3	8 10 12 13 * 17 18	7:25	1:00	0:45	0
Total		22:18	3:03	1:48	0

De posse de todas as possíveis jornadas o problema é escolher um conjunto de jornadas que cobre as tarefas, como os conjuntos mostrados nas tabelas de exemplo acima. Cada uma das jornadas selecionadas seria designada a um tripulante. Conhecendo-se os custos operacionais de cada jornada, o problema pode ser traduzido como um *set partitioning problem*, que é o assunto da próxima seção.

2.2. Set Partitioning Problem

A geração das jornadas de trabalho depende bastante do problema específico que é tratado, pois a viabilidade de uma jornada de trabalho depende de regras e acordos da própria empresa com os sindicatos envolvidos, e ainda de leis trabalhistas e governamentais que regulam as atividades dos tripulantes, diferentes para os diversos meios de transporte. Entretanto, a seleção de um subconjunto das jornadas disponíveis para cobrir todas as tarefas de um determinado dia, minimizando os custos operacionais, pode ser modelada como um problema de particionamento de conjuntos, o já comentado *set partitioning problem* (SPP), como feito em diversos trabalhos (Mingozzi, Boschetti, Ricciardelli, & Bianco, 1999) (Marsten & Shepardson, 1981). Um modelo de programação inteira para esse problema é o modelo SPP descrito por (1)-(3).

Neste modelo, J é o conjunto de jornadas viáveis e T é o conjunto de tarefas a serem cobertas. Para cada jornada $j \in J$ é associada uma variável de decisão binária x_j cujo valor será 1 se a jornada é selecionada (i.e. associada a um tripulante) ou 0, caso ela não seja selecionada. Cada jornada $j \in J$ possui um custo operacional c_j associado, logo a função objetivo (1) procura minimizar o custo total das jornadas selecionadas. O modelo utiliza ainda uma matriz binária A de dimensões $|T| \times |J|$ com valor $a_{tj} = 1$ se a tarefa $t \in T$ é coberta pela jornada $j \in J$, ou 0 caso contrário. Desta forma, o conjunto de equações (2) assegura que todas as tarefas sejam cobertas, sendo cada tarefa coberta por uma e somente uma das jornadas selecionadas. E por fim, (3) estabelece a integralidade das variáveis de decisão.

SPP:

$$\min \sum_{j \in J} c_j x_j \quad \textbf{(1)}$$

$$\sum_{j \in J} a_{tj} x_j = 1, \forall t \in T \quad \textbf{(2)}$$

$$x_j \in \{0,1\}, \forall j \in J \quad \textbf{(3)}$$

É importante relembrar que para alguns problemas é suficiente utilizar um problema de cobertura de conjuntos (*set covering problem*), onde o sinal de igualdade de (2) é substituído por sinal de maior ou igual, garantindo assim que cada tarefa seja coberta por pelo menos uma das jornadas selecionadas, permitindo múltipla cobertura. Assim, mais de um tripulante pode ser associado à mesma tarefa. Nesse caso um deles cumpre efetivamente a tarefa enquanto o outro apenas viaja como passageiro. Na maioria das instâncias utilizadas

nesse trabalho isso não é permitido, por isso utilizamos apenas o modelo de particionamento de conjuntos.

Outras restrições específicas dos problemas tratados podem ser incorporadas ao modelo SPP. Por exemplo, se um número máximo de tripulantes $MAXJ$ é estabelecido a priori, por exemplo, quando a empresa já possui os tripulantes contratados e por algum motivo não deseja contratar mais nenhum, então este valor impõe também um limite máximo de jornadas a serem selecionadas, já que cada jornada deverá ser cumprida por um tripulante. Isso pode ser controlado acrescentando-se a restrição (4) ao modelo de particionamento de conjuntos (1)-(3):

$$\sum_{j \in J} x_j \leq MAXJ \qquad \textbf{(4)}$$

O problema de particionamento de conjuntos é um problema NP-completo (Garey & Johnson, 1979), e incluir restrições como (4) não reduz a dificuldade do problema. Contudo, quando o conjunto J de jornadas não é muito grande, o problema é fácil e rapidamente resolvido pelos atuais pacotes de programação de programação inteira, como Xpress (Dash Optimization Ltd, 2006) e Cplex (ILOG S.A, 2005). Nesses casos a maior parte do tempo computacional é gasta na geração do conjunto de jornadas viáveis, isto é, no cálculo da matriz de cobertura A usada no modelo (constantes a_{tj}), e não propriamente na sua solução.

Para instâncias de menor tamanho a geração do conjunto de jornadas pode ser feita através de enumeração exaustiva. Como exemplo cita-se as instâncias com 50, 100 e 150 tarefas da OR-Library (Beasley, OR-Library), criadas e utilizadas pela primeira vez por Beasley e Cao (Beasley & Cao, 1996) e posteriormente em vários outros trabalhos, como (Mingozzi, Boschetti, Ricciardelli, & Bianco, 1999) e (Fischetti, Lodi, Martello, & Toth, 2001). Os experimentos aqui realizados mostram que essas tarefas podem ser combinadas em respectivamente 265, 1184 e 2526 jornadas viáveis, sendo possível enumerar todas essas jornadas e ainda resolver o problema de particionamento SPP correspondente quase instantaneamente com os recursos atuais.

Porém, em instâncias reais, o conjunto de possíveis jornadas geralmente é tão grande que o conjunto completo não pode ser gerado em tempo hábil. E mesmo havendo tempo para essa geração, o número de variáveis do problema de particionamento de conjuntos será muito grande, exigindo grande quantidade de memória e um tempo considerável na solução, já que se trata de um problema NP-completo. Uma estratégia de sucesso para contornar esse problema é a geração de colunas, que é apresentada no próximo capítulo.

3. Geração de Colunas

Como já mencionado, uma grande dificuldade na solução do problema de alocação de tripulação é o grande número de possíveis jornadas de trabalho. Mesmo existindo diversas regras que controlam a viabilidade de uma jornada, o conjunto completo de jornadas viáveis é geralmente grande. E quando é possível gerar o conjunto completo, este número enorme de jornadas torna difícil a solução do SPP resultante.

A Tabela 4 mostra o resultado para algumas instâncias dos problemas de alocação de tripulação aqui considerados. Para as instâncias completas dos itinerários 101, 201 e 321, com 40, 49 e 54 tarefas respectivamente, o número de jornadas viáveis ultrapassa milhões.

Tabela 4: Número de jornadas viáveis em um problema de transporte urbano

Itinerário	Número de tarefas	Jornadas viáveis
101	40	1.037.190
201	49	> 4.000.000
321	54	> 4.000.000
1170	54	292.505
2152	43	10.045

Mais importante que o grande número de jornadas viáveis é o fato de que grande parte delas não são jornadas interessantes para o problema: são viáveis, atendem todas as regras, porém possuem custo alto, ou então são semelhantes a algumas outras jornadas mais indicadas para a solução do problema. De fato, o número de jornadas efetivamente utilizadas na solução ótima é mínimo, se comparado ao número total de jornadas possíveis. A solução ótima das primeiras três instâncias, por exemplo, possui apenas 2 ou 3 jornadas. E para a última instância mencionada na tabela, itinerário 2152, um total de 12 jornadas de trabalho é suficiente para se cobrir de forma ótima todas as 43 tarefas.

Mesmo nestas instâncias pequenas é possível notar que havendo a possibilidade de prever que certas jornadas viáveis não fazem da parte da solução ótima, ou que têm pequenas chances de participar desta solução, pode-se trabalhar com um número reduzido de jornadas e encontrar a mesma solução ótima. Ou, pensando de outra forma, prever quais as jornadas têm maiores chances de participar da solução ótima. Para reforçar a necessidade deste tipo de previsão, a Tabela 5 mostra alguns dados de um problema de transporte ferroviário (Caprara, Fischetti, & Toth, 1999).

Tabela 5: Número de jornadas viáveis em um problema de transporte ferroviário

Número de tarefas	Jornadas viáveis
582	55.515
507	63.009
3.586	920.683
4.872	968.672

Para se encontrar a solução ótima não é necessário utilizar o conjunto completo de jornadas viáveis. Qualquer subconjunto que inclua todas as jornadas presentes na solução ótima pode ser utilizado. Desta forma, o ideal é gerar um subconjunto pequeno de jornadas, para não se gastar tempo desnecessário na geração das jornadas, e também tornar fácil a solução do SPP. Existem duas dificuldades principais para se colocar essa idéia em prática: (i) como gerar jornadas boas o suficiente de tal forma a incluir as jornadas da solução ótima, ou seja, como guiar a geração de jornadas para produzi-las; (ii) como garantir que nenhuma jornada não gerada não melhora a solução, ou seja, como garantir que todas as jornadas da solução ótima global estão realmente presentes neste subconjunto, que nenhuma deixou de ser gerada.

Estas duas dificuldades podem ser plenamente resolvidas pelo método chamado geração de colunas. De fato, uma estratégia de sucesso para contornar o grande número de variáveis do problema de particionamento de conjuntos resultante do problema de alocação de tripulações é a geração de colunas (Desaulniers, Desrosiers, & Solomon, 2005) (Gamache, Soumis, & Marquis, 1999) (Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998) (Souza, Moura, & Yunes, 2005). O problema é decomposto em dois problemas:

- Problema mestre: o problema a ser resolvido (nesse caso o problema de particionamento de conjuntos), porém com menos colunas (nesse caso menos jornadas de trabalho)

- Sub-problema: gerador de novas colunas para o problema mestre (nesse caso gerador de novas jornadas de trabalho), colunas estas que quando incorporadas no conjunto de colunas utilizadas pelo problema mestre melhora a solução atual

A interação entre esse dois problemas é mostrada na Figura 1. Essa interação é detalhada formalmente nas próximas seções, onde se comprova que é possível assegurar-se de que as colunas geradas no sub-problema de fato conduzem à solução ótima do problema mestre, e ainda garantir a otimalidade da solução obtida, utilizando-se fundamentos de programação linear e dualidade na relaxação linear do problema mestre. Para isso, não se resolve diretamente o problema mestre, mas sim sua relaxação linear, já que são necessários os preços duais para guiar a geração de jornadas no subproblema. Assim, se a solução linear não for inteira, ainda é necessário utilizar um esquema de enumeração *branch-and-bound* para encontrar a solução ótima, cujos detalhes são dados mais adiante, no capítulo 5.

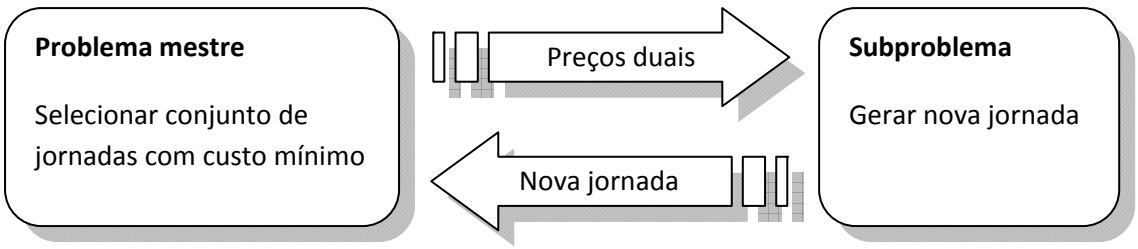

Figura 1: Interação dos problemas mestre e sub-problema no algoritmo de geração de colunas

Como o problema mestre trabalha apenas com um subconjunto de colunas, um pacote de programação linear rapidamente encontra a solução. E como novas colunas são acrescentadas pouco a pouco, uma solução pode ser obtida a partir da solução anterior geralmente com alguns poucos pivoteamentos de colunas no método simplex. Desta forma, a maior parte desse trabalho concentra-se em abordagens para se resolver o subproblema de forma mais eficiente. Algumas meta-heurísticas são utilizadas para acelerar a geração de colunas, e elas são combinadas com métodos exatos para garantir a otimalidade da solução.

3.1. Problemas com muitas restrições

Em problemas com muitas restrições é comum a utilização do método de planos de corte (*cutting planes*), não para criar novas restrições, mas para acrescentar pouco a pouco as restrições já existentes no problema. Para tal, considere um problema de programação linear LP_1 genérico,

LP_1:

$$\min c^T x \quad (5)$$
$$Ax \geq b \quad (6)$$
$$x \geq 0 \quad (7)$$

no qual o número de restrições é muito grande. É possível resolver o problema mesmo sem se utilizar todas as restrições. A idéia consiste em se resolver vários problemas menores, com menos restrições. Para isso resolve-se o problema LP_2,

LP_2:

$$\min c^T x \quad (8)$$
$$\tilde{A}x \geq \tilde{b} \quad (9)$$
$$x \geq 0 \quad (10)$$

onde $\tilde{A} \subseteq A$ e $\tilde{b} \subseteq b$, ou seja, LP_2 é o problema LP_1 com apenas um subconjunto das restrições.

A solução ótima $\tilde{x}^*$ de LP_2 é submetida a um teste para se descobrir restrições do conjunto $Ax \geq b$ (6) de LP_1 não satisfeitas. Todas ou algumas dessas restrições não satisfeitas são incluídas no conjunto $\tilde{A}x \geq \tilde{b}$ (9), modificando os subconjuntos $\tilde{A}$ e $\tilde{b}$ de LP_2. O problema LP_2 modificado é então novamente resolvido, gerando uma nova solução, que satisfaz o atual subconjunto de restrições. Novas restrições são descobertas e adicionadas, e o processo é repetido até que nenhuma restrição violada seja encontrada. Nesse momento a solução ótima de LP_2 é também solução ótima do problema original LP_1, pois atende a todas suas restrições.

A grande vantagem desse método, além de ser simples e direto, é que o tamanho do subconjunto final de restrições (9) de LP_2 geralmente é bem menor que o conjunto original (6), logo a solução desse modelo de programação linear é mais rápida.

Para se resolver um problema por esse método deve-se tomar decisão em três pontos: (i) como determinar o conjunto inicial de restrições; (ii) como descobrir as restrições não

satisfeitas pela solução atual; (iii) dado um conjunto de restrições não satisfeitas decidir quais incluir no modelo.

Desde que (ii) seja resolvido de forma exata, isto é, quanto houver restrições não satisfeitas elas são de fato encontradas, os passos (i) e (iii) podem ser decididos de qualquer forma, afetando apenas o desempenho do algoritmo em termos de tempo de execução, e não a qualidade da solução obtida. Isto porque não importando quantas e quais são as restrições consideradas inicialmente, e nem quantas ou quais das violadas são incluídas no modelo, no momento em que não são mais encontradas restrições violadas, a solução é garantidamente ótima.

Geralmente, o que mais afeta o desempenho do método em termos de tempo de execução também é o algoritmo utilizado no ponto (ii), o algoritmo que testa a solução atual para descobrir restrições violadas. Ele deve resolver o chamado *problema de separação*, separar da região viável do problema uma parte que inclui a atual solução, através da adição de novas restrições, que funcionam como planos de corte. Como se partiu do pressuposto de que o problema original possui um grande número de restrições, verificá-las uma a uma seria um processo bastante trabalhoso, e corresponderia de certa forma, a se resolver o problema original diretamente. O problema de separação pode ser modelado como outro problema de otimização, também resolvido por programação linear (geralmente inteira), talvez não produzindo todas as restrições violadas, mas individualizando a "mais violada", de acordo com algum parâmetro de medida.

3.2. Problemas com muitas variáveis

Na seção anterior foi mostrada uma forma de se lidar com um problema que apresenta um grande número de restrições, utilizando o problema de separação para iterativamente acrescentar restrições a um modelo reduzido do problema original. Idéias semelhantes podem ser usadas para se resolver problemas com número enorme de variáveis. Pensando em termos de dualidade, na verdade as idéias são idênticas.

Seja o mesmo problema anterior,

LP_1:

$$\min c^T x \quad (5)$$
$$Ax \geq b \quad (6)$$
$$x \geq 0 \quad (7)$$

desta vez com um grande número de variáveis, de tal forma que não seja viável acrescentar todas elas diretamente no modelo. Resolve-se primeiramente um problema menor, com um subconjunto de variáveis $\tilde{x} \subseteq x$,

LP_3:

$$\min \tilde{c}^T \tilde{x} \quad (11)$$
$$\tilde{A}\tilde{x} \geq b \quad (12)$$
$$\tilde{x} \geq 0 \quad (13)$$

No caso de muitas restrições, cada solução encontrada deveria ser testada em termos de viabilidade, se realmente atendia todas as restrições, até mesmo aquelas ainda não incluídas no modelo. Neste caso, com muitas variáveis, cada solução encontrada deve ser testada em termos de otimalidade, se realmente é a solução ótima do problema, até mesmo se fossem consideradas aquelas variáveis ainda não incluídas no modelo. O teste de viabilidade é feito através do problema de separação. O teste de otimalidade pode ser feito através do problema de separação no modelo dual correspondente,

LPD_1:

$$\max by \quad (14)$$
$$A^T y \leq c \quad (15)$$
$$y \geq 0 \quad (16)$$

Resolvendo-se o modelo LP_3, encontra-se uma solução $\tilde{x}^*$. Seja $\tilde{y}^*$ a solução dual correspondente no modelo LPD_1. A solução $\tilde{y}^*$ é submetida a um teste, que consiste em se descobrir restrições violadas por $\tilde{y}^*$ no conjunto dual $A^T y \leq c$ (15). Todas, ou algumas das variáveis correspondentes às restrições violadas são acrescentadas ao problema LP_3, modificando $\tilde{x}$, $\tilde{c}$ e $\tilde{A}$. Da mesma forma, esse processo é repetido até não haver mais restrições violadas no problema dual.

O teste $A^T y \leq c$ para uma solução $\tilde{y}^*$, reescrito como $c - A^T y \geq 0$, mostra de forma mais clara o significado do problema de separação no modelo dual. Para cada variável x_j o primeiro termo da inequação (17) é o seu custo reduzido, no problema primal, calculado através dos preços duais y_i^* das restrições.

$$c - \sum_i a_{ij}\, y_i^* \geq 0 \quad (17)$$

Nota-se então que uma restrição não satisfeita no problema dual é equivalente a uma variável de custo reduzido negativo no problema primal. Esta variável, se acrescentada ao modelo, pode gerar uma solução com um menor valor da função objetivo, e por isso se torna interessante seu acréscimo no problema original. Mesmo que essa nova variável não seja incorporada na solução, ela causa uma atualização nos preços duais do problema mestre, permitindo a geração de outras variáveis.

Uma nova variável acrescentada no problema original corresponde a mais uma coluna no modelo de programação linear. O método como um todo é então chamado geração de colunas. O problema original, com menos variáveis, é chamado problema mestre. O problema de separação no modelo dual (ou *pricing*), isto é, o problema de encontrar ou gerar uma nova coluna, é comumente chamado de subproblema.

Existem diferentes abordagens, de acordo com a disponibilidade de colunas no conjunto inicial. Em (Revelle, Marks, & Liebman, 1970) encontra-se um exemplo de problema de localização de facilidades com colunas conhecidas a priori. (Caprara, Fischetti, & Toth, 1999) e (Makri & Klabjan, 2004) trabalham com um conjunto inicial composto por algumas colunas, adicionando novas colunas ao longo do processo de resolução do problema. Outra abordagem consiste em inicialmente utilizar uma base artificial, sem colunas conhecidas a priori, como em (Kohl, 1995).

Vale lembrar que foi considerado o problema de minimização, mas a idéia pode ser usada em problemas de maximização de forma similar, bastando encontrar variáveis de custo reduzido positivo.

Como dito anteriormente, este método pode ser usado em problemas onde o número de variáveis é muito grande, o que tornaria impraticável a criação de um modelo com todas elas. Não é necessário introduzir todas elas no problema, basta criar um algoritmo que, a partir dos preços duais da solução atual, encontre variáveis de custo reduzido negativo, e então acrescentar apenas estas variáveis. É possível ir ainda mais longe: ele pode ser aplicado a problemas onde nem mesmo se conhece todas as variáveis. Basta gerar iterativamente as de custo reduzido negativo, que são interessantes na otimização do problema. As demais não precisam ser introduzidas, e nem sequer explicitamente conhecidas. É um esquema ideal para o trabalho aqui tratado, onde o conjunto de jornadas viáveis é muito grande, devido à natureza combinatória do problema, e também difícil de ser gerado, devido ao grande número de regras e restrições de viabilidade.

Na próxima seção é descrito o problema mestre usado na geração de colunas do problema de alocação de tripulações, e na seção seguinte o subproblema.

3.3. Problema mestre

Aplicado ao problema de alocação de tripulações, o problema mestre é um SPP, *Set Partitioning Problem*. Cada coluna da matriz de cobertura, ou seja, cada variável do problema, é uma jornada viável, que cobre um subconjunto das tarefas. Cada linha da matriz de cobertura, ou seja, cada restrição do problema, é uma tarefa que deve ser coberta. Como existem muitas jornadas viáveis, o problema possui muitas colunas, e por isso é indicado o método de geração de colunas.

Entretanto, o SPP é um problema de programação linear inteira, e não apresenta a propriedade de integralidade da solução, portanto não possui um problema dual associado cuja solução tem garantidamente o mesmo valor, impossibilitando que os custos duais das tarefas sejam corretamente calculados. Para obter os preços duais necessários na solução do subproblema, não se resolve diretamente o SPP, mas sim sua relaxação linear. Nesse caso as variáveis correspondentes a cada coluna deixam de ser binárias e passam a ser não-negativas. Não há necessidade de se limitar explicitamente seu valor em 1, pois as restrições de cobertura, juntamente com restrições de não-negatividade, impõem esse limite. Caso a

solução ótima do problema relaxado não seja inteira, parte-se para uma enumeração *branch-and-bound*, cujos detalhes são descritos no capítulo 5.

No caso das instâncias da OR-Library, além das restrições do SPP, existe um número pré-definido de tripulantes. Por isso, o número de jornadas na solução ótima é previamente estipulado em NJ, exatamente o número de tripulantes. Foi usado o seguinte modelo para o problema mestre, adaptado do modelo SPP. Relembrando, J é o conjunto de jornadas viáveis e T é o conjunto de tarefas a serem cobertas. Para cada jornada $j \in J$, tem-se seu custo operacional c_j e uma variável de decisão binária x_j associada, cujo valor é 1 se a jornada é selecionada e 0 caso contrário. A matriz de cobertura é dada por $a_{tj} = 1$ se a tarefa $t \in T$ é coberta pela jornada $j \in J$, e 0 caso contrário. Como este é o problema mestre num método de geração de colunas, em vez do conjunto completo J o modelo SPP_{OR} usa um subconjunto de jornadas $\tilde{J} \subseteq J$.

SPP_{OR}:

$$\min \sum_{j \in \tilde{J}} c_j x_j \tag{18}$$

$$\sum_{j \in \tilde{J}} a_{tj} x_j = 1, \forall t \in T \tag{19}$$

$$\sum_{j \in \tilde{J}} x_j = NJ \tag{20}$$

$$x_j \geq 0, \forall j \in \tilde{J} \tag{21}$$

No caso das instâncias de transporte urbano de Belo Horizonte, além das restrições do SPP, existe um número limite $MaxDP$ de jornadas do tipo dupla pegada sem custo adicional. É usado o seguinte modelo para o problema mestre, também adaptado do modelo SPP. Nesse modelo, a variável k conta o número de jornadas de dupla pegada além do limite, e a constante b representa o custo adicional de cada uma dessas jornadas. O conjunto $J_{DP} \subseteq J$ contém tais jornadas, sendo $\tilde{J} \subseteq J$ e $\tilde{J}_{DP} \subseteq J_{DP}$ os subconjuntos de jornadas já incluídas no modelo.

SPP_{BH}:

$$\min \sum_{j \in \tilde{J}} c_j x_j + bk \tag{22}$$

$$\sum_{j \in \tilde{J}} a_{tj} x_j = 1, \forall t \in T \tag{23}$$

$$\sum_{j \in \tilde{J}_{DP}} x_j \leq MaxDP + k \tag{24}$$

$$x_j \geq 0, \forall j \in \tilde{J} \tag{25}$$

$$k \geq 0 \tag{26}$$

O problema SPP_{BH} continua sendo NP-hard. Fazendo-se $MaxDP = |J_{DP}|$ a restrição (24) e a variável k se tornam redundantes, e o modelo se torna o clássico SSP. Assim, um algoritmo para o problema com a restrição adicional de duplas pegadas pode ser usado para se resolver o SPP. Como a transformação entre os modelos é polinomial em relação à entrada do SPP, e o SPP é NP-hard, conclui-se que o problema de alocação de tripulação SPP_{BH} também é NP-hard.

3.4. Subproblema

A finalidade do subproblema é gerar uma (ou mais) nova jornada de trabalho, que seja viável e de custo reduzido negativo, quando considerados os preços duais das tarefas do problema mestre anteriormente resolvido. Estas novas jornadas são novas colunas acrescentadas ao problema mestre, que melhoram o valor da solução, ou atualizam os preços duais para permitir a geração de outras novas jornadas pelo subproblema.

Para o problema de alocação de tripulação, que possui diversas restrições na viabilidade de uma jornada, portanto na criação de uma nova coluna, uma abordagem bastante utilizada é a modelagem deste problema em um grafo. A topologia do grafo é facilmente definida pelas características de uma jornada de trabalho: uma seqüência de tarefas ordenadas pelo horário de execução. A construção do grafo, e o problema resultante são detalhados nas próximas seções.

3.4.1. Caminho mínimo com restrição de recursos

A geração de uma nova jornada de trabalho pode ser feita através da modelagem em grafos. Usa-se um grafo dirigido $G = (V, A)$, em que o conjunto V de vértices contém um vértice v_t para cada tarefa $t \in T$ a ser coberta no problema mestre, e dois vértices adicionais, v_0 e v_f; e o conjunto A de arcos contém os arcos (v_0, v_t) e (v_t, v_f), $\forall t \in T$, e os arcos (v_t, v_w) para todo par de tarefas t e w tal que a tarefa w possa ser efetuada logo após a tarefa t em alguma jornada viável. Na definição desses últimos arcos são considerados os horários de execução das tarefas e outras características específicas do problema. Dessa forma, a existência de um arco (v_t, v_w) indica que o horário de execução da tarefa w é posterior ao da t, que não ocorre sobreposição de horários, e que essa sequência atende outros requisitos do problema tratado, como troca de estações, de veículos, e tempo mínimo entre tarefas consecutivas.

O grafo G assim construído é um grafo dirigido acíclico, e qualquer jornada de trabalho viável é um caminho do vértice v_0 ao vértice v_f no grafo. Os vértices v_t nesse caminho representam as tarefas t da jornada construída. Se for associado a cada vértice v_t um custo $-\pi_t^*$, referente ao preço dual da tarefa correspondente no problema mestre, e incorporado de alguma forma o custo c_j de cada jornada como custos nos arcos, o caminho mais curto de v_0 a v_f representa a solução do subproblema, a coluna que está sendo buscada para ser adicionada

ao problema mestre. Se tal caminho mínimo possuir custo negativo, a jornada (coluna) correspondente é adicionada ao problema mestre, que é novamente resolvido, e têm-se novos preços duais para buscar outro caminho mínimo no mesmo grafo, com os custos dos vértices alterados. Quando o caminho mínimo tiver custo zero ou positivo, então a solução atual do problema mestre é a solução ótima do problema.

Os custos dos vértices podem ser passados para os arcos incidentes, e assim um algoritmo para resolver o problema precisa tratar apenas custos dos arcos. Importante ressaltar que, apesar dos arcos, em sua maior parte, apresentarem valores negativos, tem-se a garantia de haver uma solução ótima finita, já que o grafo é dirigido acíclico, portanto não há ciclos de valor negativo.

Para garantir otimalidade na solução do problema mestre, o subproblema deve ser resolvido de forma exata. Se houver alguma coluna com custo reduzido negativo, ela deve ser descoberta, ou seja, deve-se resolver o problema de caminho mínimo por um algoritmo exato. Isso pode ser feito de forma eficiente pelo algoritmo de Floyd, que também trata custos negativos, e tem ordem de complexidade polinomial. Outra forma é a solução de um modelo de programação linear inteira, como o modelo CM seguinte. Nesse modelo, é utilizada a variável de decisão v para os vértices do grafo e y para os arcos. Os custos c_a representam os custos c_j das colunas do problema mestre, distribuídos nos arcos. Detalhes de como essa distribuição de custos pode ser feita são dados nos modelos específicos nas subseções seguintes. A notação $\delta^+(v)$ representa o conjunto de arcos com vértice inicial v, ou seja, o conjunto de arcos de saída, e a notação $\delta^-(v)$ representa o conjunto de arcos com vértice final v, ou seja, o conjunto de arcos de entrada.

CM:

$$\min \sum_{a \in A} c_a y_a + \sum_{t \in T} \pi_t v_t \quad \textbf{(27)}$$

$$\sum_{a \in \delta^+(v_0)} y_a = \sum_{a \in \delta^-(v_f)} y_a = 1 \quad \textbf{(28)}$$

$$\sum_{a \in \delta^+(v_t)} y_a = \sum_{a \in \delta^-(v_t)} y_a = v_t, \forall t \in T \quad \textbf{(29)}$$

$$v_t, y_a \in \{0,1\}, \forall v_t \in V, \forall a \in A \quad \textbf{(30)}$$

O objetivo é encontrar o caminho com menor custo, somado os custos dos vértices e arcos que fazem parte do caminho (27). A restrição (28) garante que deve haver apenas um caminho, iniciado no vértice v_0 e terminado no vértice v_f. O conjunto de restrições (29) faz a ligação entre as variáveis dos vértices e dos arcos, garantindo que para cada vértice no caminho formado sejam escolhidos exatamente dois arcos, um de entrada e um de saída, e ainda que não sejam escolhidos arcos incidentes a vértices que não fazem parte do caminho. E finalmente as restrições (30) garantem que as variáveis sejam binárias.

Entretanto, apesar de toda jornada viável ser um caminho no grafo G, nem todo caminho nesse grafo é uma jornada viável. Algumas restrições do problema de alocação de tripulações não podem ser diretamente incorporadas ao grafo em um problema de caminho

mínimo, como por exemplo, limite de tempo de uma jornada de trabalho, número máximo de tarefas dentro de uma jornada, e regras de descanso durante a jornada. A construção do grafo modela as restrições entre tarefas consecutivas, mas não o conjunto de tarefas de uma jornada. Para se assegurar de encontrar apenas caminhos que representam jornadas viáveis, outros valores são associados aos vértices e arcos além dos custos, na forma de recursos que são consumidos ao longo do caminho, e são estabelecidos limites mínimos e máximos de consumo desses recursos. Caminhos em construção só podem incluir vértices e arcos se sua inclusão atende os limites de consumo de recursos, considerando-se os valores já utilizados ao longo do caminho construído. Quando um vértice é adicionado ao caminho, o consumo de recursos é atualizado através dos valores presentes no vértice e no arco selecionado.

O subproblema se torna então um problema de caminho mínimo com restrição de recursos (Desaulniers, Desrosiers, & Solomon, 2005), que é mais complexo de ser resolvido, não existindo ainda algoritmos polinomiais para sua solução. Mesmo assim, para algumas instâncias, os pacotes de programação linear conseguem resolvê-lo em tempo razoável. Outras abordagens interessantes são programação dinâmica (Gamache, Soumis, & Marquis, 1999) e programação por restrições (Souza, Moura, & Yunes, 2005).

Para exemplificar a modelagem em grafos do subproblema de caminho mínimo com restrição de recursos, considere um problema no qual todas as tarefas têm início e fim no mesmo local, e cujos horários de início e fim, a partir do meio-dia, sejam os expressos em minutos na Tabela 6. Considere ainda que algumas tarefas sejam especiais, nesse caso as tarefas 3 e 5, e que uma jornada de trabalho possa incluir no máximo uma tarefa especial. Esse é um caso comum na prática, por exemplo, com tarefas noturnas, tarefas de longa duração ou de longa distância, dentre outras. Dado que cada jornada pode ter no máximo 300 minutos de trabalho efetivo, descontados os intervalos entre tarefas, o grafo da Figura 2 mostra a modelagem do problema.

Na construção do grafo são considerados dois recursos:

- recurso A: tempo acumulado de trabalho efetivo, em minutos
- recurso B: quantidade de tarefas especiais

Existe um arco ligando dois vértices se eles representam tarefas que podem ser feitas em seqüência. Nesse exemplo, a única restrição é que o horário inicial do vértice final do arco seja posterior ao horário final do vértice inicial do arco, não havendo restrição de troca de estação ou de veículo, por exemplo, como acontece em outras instâncias. Em cada arco o consumo do recurso A é a duração da tarefa do vértice final do arco, e o valor consumido do recurso B é 1 quando a tarefa do vértice final do arco é especial, e 0 nos demais casos.

Tabela 6: Exemplo de instância de um problema de alocação de tripulações

Tarefa	Horário Inicial	Horário Final	Tarefa Especial
1	100	200	
2	150	300	
3	250	350	*
4	320	380	
5	400	500	*
6	510	590	

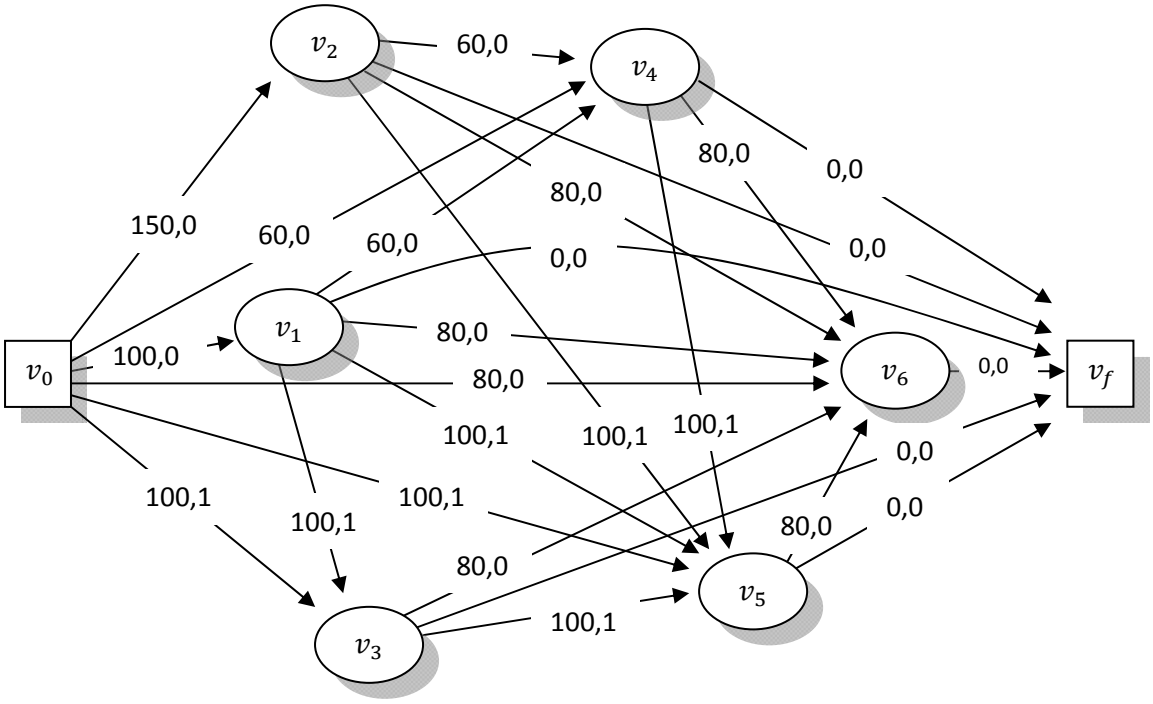

Figura 2: Grafo do subproblema para a instância da Tabela 6

Existem vários caminhos do vértice v_0 ao vértice v_f, mas apenas alguns são viáveis de acordo com os recursos consumidos. Para respeitar as restrições de tarefa especial e tempo máximo de trabalho efetivo, são estabelecidos em cada vértice do grafo um consumo máximo de 300 unidades do recurso A e 1 unidade do recurso B. Estabelecendo-se esse limite em cada vértice, caminhos inviáveis são cortados assim que violam os limites de algum dos recursos, evitando gerar o caminho completo sem necessidade. Desta forma, os seguintes caminhos parciais, por exemplo, não são viáveis:

- (v_0, v_3) (v_3, v_5), porque acumula 2 unidades do recurso B, ultrapassando seu limite
- (v_0, v_2) $(v_2, v_4)(v_4, v_5)$, porque acumula 310 unidades do recurso A, ultrapassando seu limite

Nenhum destes dois caminhos prossegue para incluir outro vértice e nem mesmo para fechar a jornada incluindo o vértice v_f. Por outro lado, existem outros caminhos que não violam os limites de utilização dos recursos em nenhum dos vértices por onde passam, e então representam jornadas viáveis segundo as regras impostas. Eis alguns exemplos:

- (v_0, v_1) $(v_1, v_4)(v_4, v_5)(v_5, v_f)$, consumindo 260 do recurso A e 1 do recurso B
- (v_0, v_2) $(v_2, v_4)(v_4, v_6)(v_6, v_f)$, consumindo 290 do recurso A e 0 do recurso B
- (v_0, v_1) $(v_1, v_5)(v_5, v_6)(v_6, v_f)$, consumindo 280 do recurso A e 1 do recurso B

Esses caminhos representam respectivamente as jornadas de trabalho com tarefas 1-4-5, 2-4-6 e 1-5-6. Quais dessas jornadas são interessantes para se acrescentar no problema mestre depende dos preços duais das tarefas incluídas. Como os vértices representam tarefas, então os preços duais são os custos associados aos vértices (que, como já mencionado, podem ser transferidos aos arcos para facilitar o uso de algoritmos de caminho mais curto). Aos arcos são associados todos os demais custos envolvidos no problema, por exemplo, o custo por tempo ocioso entre tarefas.

Se o tempo de intervalo entre as tarefas também deve ser computado no cálculo do limite de tempo de trabalho da jornada, ele pode ser adicionado ao consumo de recurso de tempo dos arcos. Por exemplo, o arco (v_1, v_4) consumiria 180 desse recurso, sendo 60 pelo trabalho efetivo na tarefa 4 e 120 entre o horário final da tarefa 1 e o início da tarefa 4. Isso pode ser modelado como outro recurso, caso exista um tempo limite de duração total da jornada, e outro limite para o trabalho efetivo. Outras restrições podem ser modeladas por consumo de recursos de forma semelhante.

Nas próximas subseções os modelos para os subproblemas das instâncias consideradas nesse trabalho são apresentados, e no capítulo 4 são descritos os métodos empregados na sua solução.

3.4.2. Modelagem para instâncias da OR-Library

A OR-Library contém 10 instâncias para o problema de alocação de tripulações. Para cada instância é fornecido o número de tarefas, e para cada uma seu horário inicial e final. São fornecidos ainda o tempo máximo de duração de uma jornada, e o custo de transição entre cada par de tarefas que podem aparecer em seqüência. Uma jornada é viável se suas tarefas não se sobrepõem no tempo, e se o tempo total da jornada não ultrapassa o tempo limite. O custo da jornada é a soma dos custos de transição entre as tarefas que a compõem. Uma solução viável é um conjunto de jornadas viáveis que cobrem cada tarefa uma única vez. Cada jornada será cumprida por um tripulante, cuja quantidade é estabelecida a priori.

Para construir o modelo do subproblema, considere $\tilde{\pi}_t$ e $\tilde{\mu}$ os preços duais associados às restrições de combrimento de tarefas (19) e do número de jornadas selecionadas (20) do problema mestre SPP_{OR}, para a solução do problema com um conjunto $\tilde{J}$ de jornadas consideradas.

No grafo do subproblema cada arco $a = (v_t, v_w)$ possui custo c_a igual ao valor de transição c_{tw} entre o par de tarefas (t, w) fornecido na entrada. Os arcos (v_0, v_t) e (v_t, v_f), que conectam os vértices-tarefa aos vértices inicial e final possuem custo c_a = 0. Nas instâncias da OR-Library, cada tarefa $t \in T$ tem um tempo de trabalho w_t. Existe ainda um limite máximo $MaxW$ para o tempo total de trabalho de uma jornada, geralmente de 480 minutos, o que corresponde a 8 horas de trabalho. O valor d_a utilizado no modelo é a duração do arco $a = (v_t, v_w)$, que corresponde ao tempo de trabalho do vértice final somado ao tempo de intervalo entre as tarefas.

SUB_{OR}:

$$\min \sum_{a \in A} c_a y_a - \sum_{t \in T} \tilde{\pi}_t v_t - \tilde{\mu} \quad (31)$$

$$\sum_{a \in \delta^+(v_0)} y_a = \sum_{a \in \delta^-(v_f)} y_a = 1 \quad (32)$$

$$\sum_{a \in \delta^+(v_t)} y_a = \sum_{a \in \delta^-(v_t)} y_a = v_t, \forall t \in T \quad (33)$$

$$\sum_{a \in A} d_a y_a \leq MaxW \quad (34)$$

$$v_t, y_a \in \{0,1\}, \forall v_j \in V, \forall a \in A \quad (35)$$

As restrições (32)-(33) e (35) são as restrições de caminho mínimo no grafo, são as mesmas do modelo CM. A restrição (34) modela a característica específica dessas instâncias, que é o limite máximo de tempo de trabalho numa jornada viável. O modelo SUB_{OR} é, portanto, o modelo CM com uma restrição adicional, que modela o consumo do recurso tempo. Qualquer solução viável desse modelo representa uma jornada viável. Uma solução viável de custo negativo representa uma jornada cuja inclusão no problema mestre pode melhorar o valor de sua solução. Utilizando (31) encontra-se a jornada de custo reduzido mínimo, aquela que nesse momento mais melhora a solução do problema mestre.

Como a modelagem desse problema requer a adição de apenas um recurso consumido no caminho do grafo (o tempo), ele pode ser resolvido com programação dinâmica (Desaulniers, Desrosiers, & Solomon, 2005). Nesse trabalho foi utilizado o software Xpress (Dash Optimization Ltd, 2006) para resolver esse problema de programação linear inteira. Os resultados são mostrados no capítulo 6.

3.4.3. Modelagem para instâncias da BHTrans

O modelo do subproblema para essas instâncias segue a mesma idéia de caminho mínimo com restrição de recursos, porém é bem mais complexo que o anterior, porque são necessários vários recursos: duração da jornada, tempo de descanso, número de intervalos de longa duração, número de intervalos com duração mínima de 15 minutos, e tempo de trabalho efetivo.

Para construir um modelo que contabiliza todos esses detalhes, são utilizados os seguintes dados:

- V: conjunto de vértices do grafo

- A: conjunto de arcos do grafo
- P: conjunto de arcos com intervalo entre tarefas maior que 2 horas (que caracterizam dupla pegada), $P \subseteq A$
- Q: conjunto de arcos com intervalo entre tarefas maior ou igual a 15 minutos, $Q \subseteq A$
- c_a: custo do arco a, que envolve troca de veículo ou de estação
- d_a: duração do arco a, a soma da duração da tarefa correspondente ao vértice final e o intervalo entre as tarefas do arco. Assim, para um arco $a = (v_t, v_w)$, d_a=hora final da tarefa w – hora final da tarefa t
- l_a: tempo efetivo de trabalho no arco a, que é o tempo efetivo de trabalho no vértice final. Assim, para um arco $a = (v_t, v_w)$, l_a=hora final da tarefa w – hora inicial da tarefa w – folga entre viagens do bloco da tarefa w
- μ: preço dual da restrição de dupla pegada (24) no problema mestre SPP_{BH}
- π_t: preço dual da restrição de cobertura da tarefa t (23) no problema mestre SPP_{BH}
- c_h: custo por tempo extra (minutos que extrapolam a duração padrão da jornada)
- c_r: custo por tempo ocioso (minutos de intervalo entre tarefas e folga entre viagens)

Para os arcos $a = (v_t, v_f)$, que finalizam o caminho, d_a = l_a = 0. Para os arcos $a = (v_0, v_t)$, que iniciam o caminho, d_a e l_a são calculados da mesma forma que os arcos entre tarefas, porém considerando-se que o intervalo entre as tarefas é 0.

O modelo SUB_{BH} a seguir procura uma jornada viável com custo mínimo. Para isso utiliza as seguintes variáveis de decisão:

- y_a e v_t, como em SUB_{OR}, variáveis binárias que assumem 1 ou 0 caso o arco a e o vértice t sejam usados ou não no caminho
- p: variável binária que assume o valor 1 se a jornada é dupla pegada, e 0 senão
- h: número de minutos extras na jornada
- q: tempo adicional para completar o limite mínimo de 30 minutos de descanso
- r: tempo ocioso, ou de descanso, na jornada

SUB_{BH}:

$$\min \sum_{a \in A} c_a y_a - \sum_{t \in T} \tilde{\pi}_t v_t - \tilde{\mu} p + c_h h + c_r r \quad \textbf{(36)}$$

$$\sum_{a \in \delta^+(v_0)} y_a = \sum_{a \in \delta^-(v_f)} y_a = 1 \quad \textbf{(37)}$$

$$\sum_{a \in \delta^+(v_t)} y_a = \sum_{a \in \delta^-(v_t)} y_a = v_t, \forall t \in T \quad \textbf{(38)}$$

$$\sum_{a \in P} y_a \leq p \quad \textbf{(39)}$$

$$\sum_{a \in Q} y_a \geq 1 - p \quad \textbf{(40)}$$

$$\sum_{a \in A} (d_a - l_a) y_a + q \geq 30 - 30p \quad (41)$$

$$\sum_{a \in A} d_a y_a + q \leq 430 - 30p + h \quad (42)$$

$$\sum_{a \in A} l_a y_a + r \geq 430 - 30p + h \quad (43)$$

$$h \leq 120 \quad (44)$$

$$v_t, y_a, p \in \{0,1\}, \forall v_t \in V, \forall a \in A \quad (45)$$

$$h, r, q \geq 0 \quad (46)$$

O objetivo é encontrar um caminho que representa uma jornada de custo reduzido mínimo (36). Este caminho deve iniciar no vértice v_0 e terminar no vértice v_f (37). Cada vértice que faz parte do caminho deve ter selecionado exatamente um arco de entrada e um arco de saída, e vértices que não estão no caminho não deve ter arcos de entrada nem de saída selecionados (38). A jornada pode ter no máximo um intervalo longo (superior a duas horas), e se houver é uma jornada do tipo dupla pegada (39). Deve haver pelo menos 15 minutos ininterruptos livres em algum momento da jornada de trabalho, a menos que ela seja uma dupla pegada (40). No caso de dupla pegada, 30 minutos são descontados nas restrições (41)-(43), pois jornada desse tipo não exige 30 minutos de descanso. (41) garante os 30 minutos de descanso em jornadas simples, somado o tempo ocioso entre as tarefas, e os q minutos adicionados ao fim da jornada, se for necessário. (42) restringe a duração máxima da jornada em 430 minutos (6:40 + 30 se for jornada simples), podendo haver hora extra, e (43) calcula os r minutos de tempo ocioso total da jornada. As demais restrições, (44)-(46), estabelecem o conjunto de valores possíveis das diversas variáveis de decisão, limitando o tempo de hora extra em 2 horas (44).

O subproblema continua sendo um problema de caminho mínimo com restrições de recurso, sendo os recursos o tempo máximo da jornada, o tempo mínimo de descanso durante a jornada, a quantidade de intervalos de 15 minutos e de intervalos superiores a 2 horas, e ainda as horas extras. Por causa da quantidade de recursos, resolver o problema através de programação dinâmica por rotulação de vértices (Desaulniers, Desrosiers, & Solomon, 2005) (Gamache, Soumis, & Marquis, 1999) se torna impraticável. Os experimentos mostraram que obter a solução ótima por um pacote de programação linear inteira é mais rápido e simples. Porém, ainda assim leva um tempo razoável, e como o subproblema deve ser resolvido muitas vezes ao longo da geração de colunas, pois é o responsável por gerar cada coluna necessária no problema mestre, essa abordagem pode não resolver o problema para instâncias grandes em tempo satisfatório. Por isso optou-se por resolver o subproblema heuristicamente, enquanto as heurísticas forem capazes de gerar novas colunas de custo reduzido negativo. Três alternativas são propostas nesse trabalho para se resolver o subproblema heuristicamente. Elas são apresentadas no próximo capítulo.

3.4.4. Modelagem para instâncias aéreas

Foram construídas algumas instâncias baseadas em dados reais de empresas aéreas. As restrições de viabilidade de uma jornada são estabelecidas pela portaria interministerial que regulamenta a profissão (Sindicato Nacional dos Aeronautas, 1988). Nem todas as restrições dessa portaria foram utilizadas porque algumas dizem respeito à seqüência de jornadas de trabalho ao longo da semana ou do mês, pertencentes ao problema *crew rostering*, e não ao de *crew scheduling*, que é tratado neste trabalho.

A tripulação nesse caso é o conjunto de tripulantes que exercem função a bordo da aeronave: comandante, co-piloto, mecânico de vôo, navegador, rádio-operador e comissário. As tripulações podem ser mínima, simples, composta ou de revezamento. Para testes dos algoritmos foram utilizadas apenas tripulações simples. Uma jornada de trabalho para estas tripulações deve obedecer ao limite de tempo de vôo total de 9 horas e 30 minutos e ao limite de 5 pousos. Deve haver um intervalo mínimo de 30 minutos entre dois vôos consecutivos da jornada de trabalho. A jornada de trabalho se inicia 30 minutos antes do primeiro vôo, e termina 30 minutos após o último vôo. O tempo total da jornada de trabalho, incluindo o tempo de vôo, não pode exceder 11 horas.

Os custos envolvidos numa jornada de trabalho envolvem, entre outros, o tempo de trabalho e tempo de vôo, a distância total percorrida, e ainda custos de hospedagem quando a jornada termina em cidade diferente daquela onde reside a tripulação. Como esses valores não estavam disponíveis, neste trabalho são considerados como custos operacionais apenas o tempo ocioso da jornada de trabalho, que é o tempo em que o tripulante não está efetivamente trabalhando a bordo ou cumprindo os 30 minutos obrigatórios antes e depois dos vôos, e o tempo inutilizado, que é aquele acrescentado para completar as 11 horas permitidas da jornada de trabalho.

O problema mestre utilizado é o problema SPP sem nenhuma restrição adicional, e o subproblema é mais uma vez modelado sobre um grafo, e corresponde ao problema de caminho mínimo com restrições de recursos, sendo os recursos o tempo total da jornada de trabalho, o tempo total de vôo e o número de pousos. O modelo SUB_A a seguir, composto pelas restrições (47)-(55), utiliza nomenclatura semelhante à do modelo SUB_{BH}:

- V: conjunto de vértices do grafo
- A: conjunto de arcos do grafo
- c_a: custo do arco a
- d_a: duração do arco a, a soma da duração da tarefa correspondente ao vértice final e o intervalo entre as tarefas do arco. Assim, para um arco $a = (v_t, v_w)$, d_a=hora final da tarefa w – hora final da tarefa t
- l_a: tempo efetivo de trabalho no arco a, que é o tempo efetivo de trabalho no vértice final mais os 30 minutos obrigatórios. Assim, para um arco $a = (v_t, v_w)$, l_a=hora final da tarefa w – hora inicial da tarefa w + 30
- $\tilde{\pi}_t$: preço dual da restrição de cobertura da tarefa t (2) no problema mestre SPP

- c_q: custo por tempo inutilizado (minutos adicionados ao fim da jornada para completar o tempo padrão de 11 horas)
- c_r: custo por tempo ocioso (minutos de intervalo entre tarefas)

Para os arcos $a = (v_t, v_f)$, que finalizam o caminho, d_a = l_a = 30. Para os arcos $a = (v_0, v_t)$, que iniciam o caminho, d_a e l_a são a duração da tarefa t mais os 30 minutos.

O modelo SUB_A procura uma jornada viável com custo reduzido mínimo. Para isso utiliza as seguintes variáveis de decisão:

- y_a e v_t, como em SUB_{OR} e SUB_{BH}, variáveis binárias que assumem 1 ou 0 caso o arco a e o vértice t sejam usados ou não no caminho
- q: tempo adicional para completar o tempo padrão da jornada
- r: tempo ocioso, ou de descanso, na jornada

SUB_A:

$$\min \sum_{a \in A} c_a y_a - \sum_{t \in T} \tilde{\pi}_t v_t + c_q q + c_r r \quad \textbf{(47)}$$

$$\sum_{a \in \delta^+(v_0)} y_a = \sum_{a \in \delta^-(v_f)} y_a = 1 \quad \textbf{(48)}$$

$$\sum_{a \in \delta^+(v_t)} y_a = \sum_{a \in \delta^-(v_t)} y_a = v_t, \forall t \in T \quad \textbf{(49)}$$

$$\sum_{t \in T} v_t \leq 5 \quad \textbf{(50)}$$

$$\sum_{a \in A} l_a y_a \leq 570 \quad \textbf{(51)}$$

$$\sum_{a \in A} d_a y_a + q = 660 \quad \textbf{(52)}$$

$$\sum_{a \in A} l_a y_a + r + q = 660 \quad \textbf{(53)}$$

$$r, q \geq 0 \quad \textbf{(54)}$$

$$v_t, y_a \in \{0,1\}, \forall v_t \in V, \forall a \in A \quad \textbf{(55)}$$

O objetivo é encontrar o caminho de custo reduzido mínimo (47) que sai do vértice inicial v_0 e termina no vértice final v_f (48). Para cada vértice nesse caminho devem ser selecionados exatamente um arco de chegada e um de saída (49), e podem haver no máximo 5 vértices intermediários (50), que é o limite máximo de pousos numa jornada. O tempo total de vôo não pode ultrapassar 9 horas e 30 minutos (51), e o tempo total de trabalho é limitado em 11 horas, sendo ainda computados o tempo adicional e o tempo ocioso (52)-(53) da jornada representada pelo caminho selecionado. Esses tempos são não negativos (54) e cada vértice e arco podem fazer parte ou não do caminho (55).

4. Solução do Sub-Problema

A Figura 3 ilustra como o método de geração de colunas é aplicado ao problema de alocação de tripulações, mostrando mais uma vez o papel dos problemas mestre e subproblema. Essa é a forma clássica, em que o subproblema é resolvido sempre de forma exata, gerando em cada iteração a coluna "ideal" para o problema mestre.

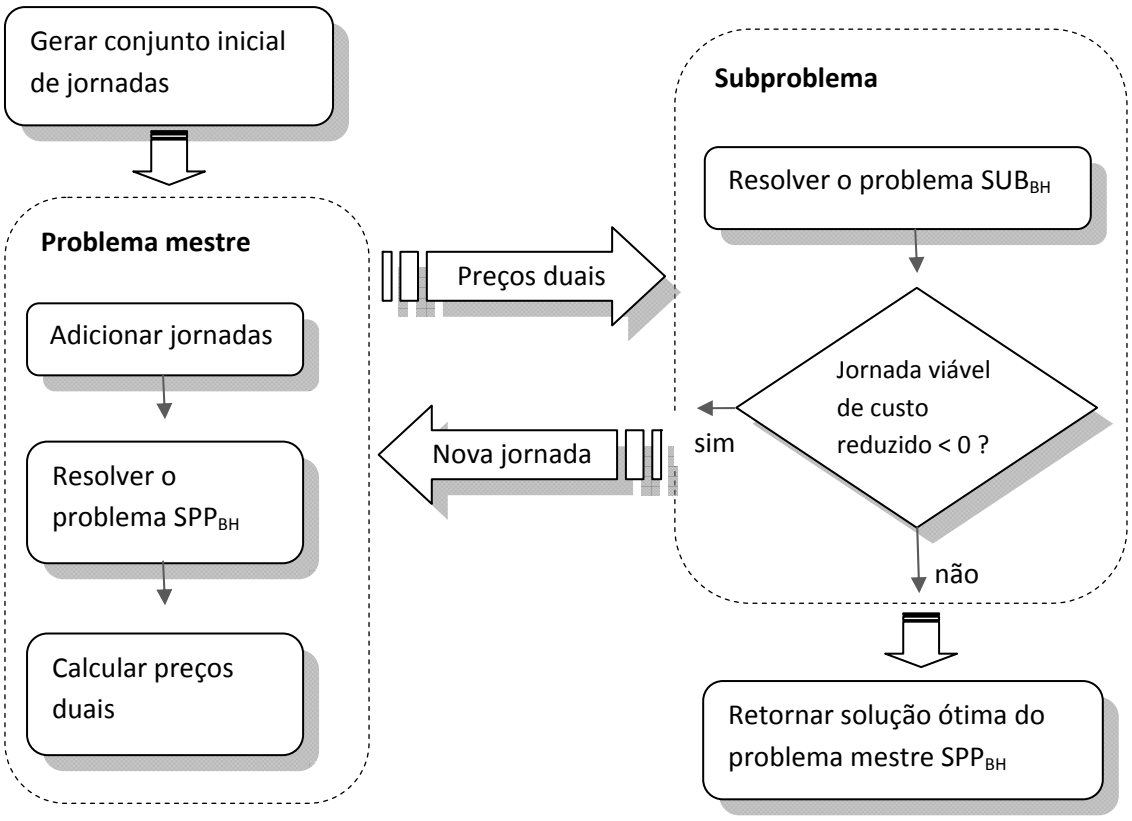

Figura 3: Método de geração de colunas aplicado ao problema de alocação de tripulações

O método de geração de colunas já foi aplicado na solução de problemas de alocação de tripulação por diversos autores (Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998) (Gamache, Soumis, & Marquis, 1999) (Souza, Moura, & Yunes, 2005) (Fores, Proll, & Wren, 1996). O método empregado de solução no subproblema nesses trabalhos é exato, seja por programação linear inteira, ou programação dinâmica ou ainda programação por restrições.

A dificuldade encontrada na aplicação do método é que a solução do subproblema é demorada para as instâncias de transporte urbano da BHTrans. Em nossos experimentos, resolver o problema de forma determinística com programação dinâmica, pelo algoritmo de rotulação de vértices utilizado em (Gamache, Soumis, & Marquis, 1999) é impraticável nesse caso, por causa da quantidade de recursos necessários para modelar o problema. Nesse caso,

a solução do modelo SUB_{BH} pelo Xpress é mais rápida que o método de programação dinâmica citado, mas ainda demorada. É um modelo de programação linear inteira, e em vários casos os pacotes de programação linear encontram dificuldade em encontrar uma solução inteira, ou provar a otimalidade de uma solução inteira já encontrada. Isso foi comprovado empiricamente pelo tempo gasto na otimização e pelo número de nós de *branch-and-bound* utilizados na busca.

Tendo em vista que o objetivo do subproblema é encontrar uma nova jornada, de custo reduzido negativo, pode-se interromper a execução do Xpress tão logo tenha sido encontrada uma jornada desse tipo. A solução ótima do modelo traz a jornada de menor custo reduzido, porém qualquer uma que tenha custo reduzido negativo já é interessante como resposta do subproblema. Ela pode melhorar a solução atual do problema mestre, ou mesmo que não seja incluída nessa solução, sua inclusão no problema altera os custos duais e guia o subproblema na geração de outra jornada. Logo, não é necessário esperar que o Xpress execute toda a árvore de *branch-and-bound* em busca da "melhor" jornada. Entretanto, como mesmo a primeira solução inteira muitas vezes demanda tempo para ser obtida, propõe-se outra estratégia: resolver a relaxação linear do subproblema e construir heuristicamente uma jornada viável a partir de sua solução.

Como é uma solução heurística, uma jornada viável de custo reduzido negativo pode não ser encontrada, mesmo quando existe alguma. Para garantir a otimalidade da solução encontrada pelo problema mestre no final da execução da geração de colunas, toda vez que a heurística falha na obtenção da jornada apropriada, o método exato é executado, ou seja, o modelo SUB_{BH} é resolvido pelo Xpress até a solução ótima. A Figura 4 mostra como as resoluções, heurística e exata, se interagem na solução do subproblema.

Heurísticas já foram utilizadas em diversos trabalhos de alocação de tripulações, incluindo algoritmos genéticos, também usados nesse trabalho (Wren & Wren, 1995) (Souza, Cardoso, Silva, Rodrigues, & Mapa, 2004) (Kwan, Wren, & Kwan, 2000) (Cabral, Souza, Maculan, & Pontes, 2000). Elas geralmente são aplicadas diretamente no problema SCP ou SPP, sem o uso de geração de colunas ou então para gerar uma solução inicial viável. Também já foram usadas para a geração de jornadas de trabalho, mas nesse caso com o objetivo de encontrar uma solução aproximada com boas jornadas geradas heuristicamente, sem garantia de otimalidade da solução encontrada. Não foram encontradas aplicações das heurísticas usadas nesse trabalho em outros trabalhos de alocação de tripulações da literatura, da forma como são incorporadas no método de geração de colunas nesse trabalho.

Na seção seguinte, a heurística baseada na relaxação linear do subproblema é descrita com mais detalhes, e em seguida é proposta a utilização de duas meta-heurísticas, também para resolver o subproblema. Elas tentam gerar jornadas viáveis de custo reduzido negativo baseando-se nos preços duais fornecidos pelo problema mestre. Elas são incorporadas no algoritmo de geração de colunas da mesma forma, como ilustrado na Figura 4.

Apesar de em alguns momentos ser utilizada a nomenclatura SUB_{BH} referindo-se à geração de jornadas para o problema de transporte urbano da BHTrans, isso é feito apenas para facilitar a explicação das heurísticas, pois as mesmas idéias se aplicam na solução dos modelos SUB_{OR} e SUB_{A}.

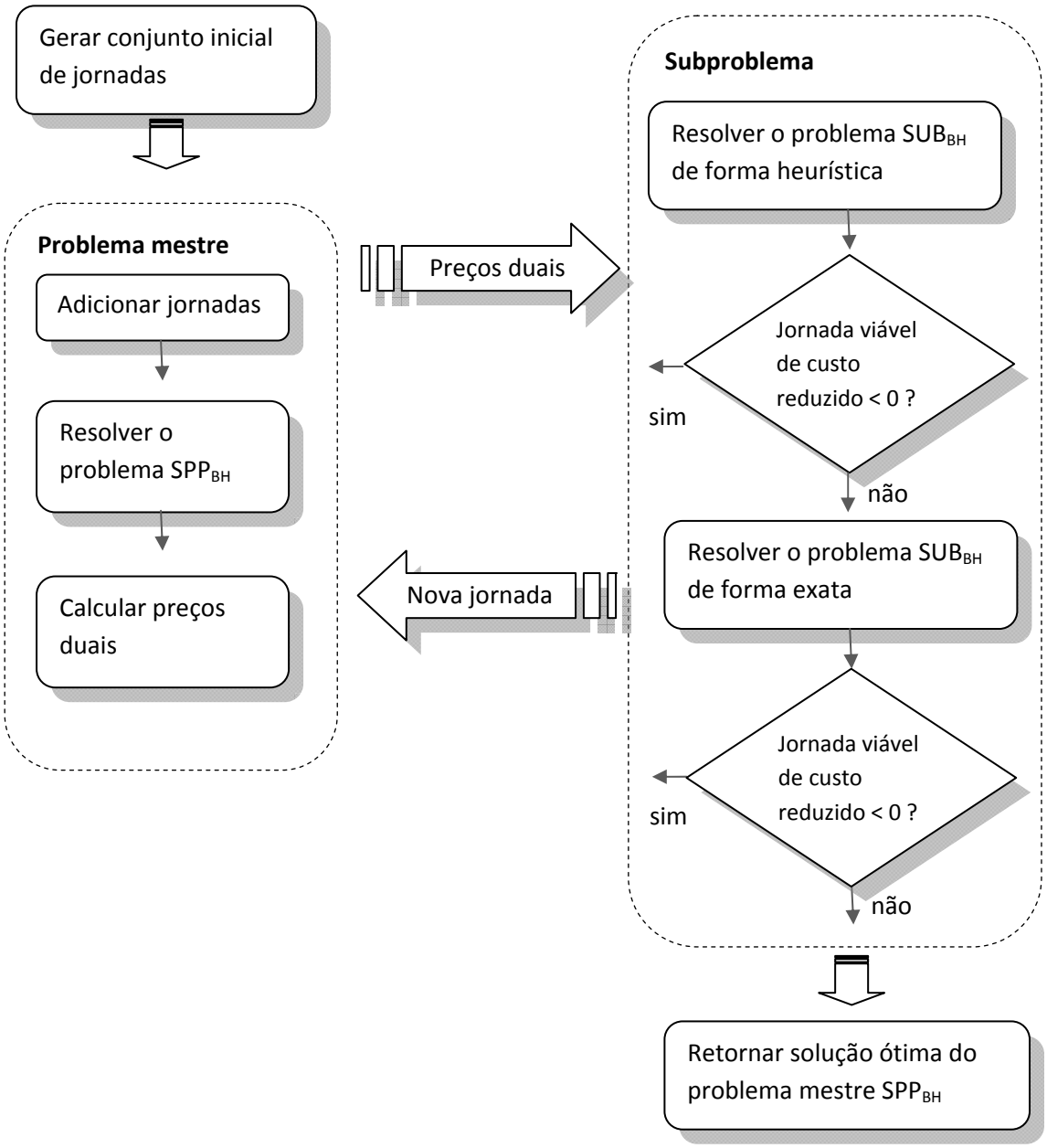

Figura 4: Solução heurística do subproblema incorporada no método de geração de colunas

4.1. Relaxação Linear

A primeira heurística proposta é uma heurística simples, do tipo *greedy*, que tenta criar uma jornada viável a partir da solução da relaxação linear do subproblema. A relaxação linear do modelo do subproblema é submetida a um pacote de programação linear, e se a solução do problema relaxado não for inteira, essa solução fracionária é usada para construir uma jornada (uma solução inteira, mesmo que não ótima).

A solução ótima do subproblema é um caminho do vértice v_0 ao vértice v_f no grafo construído, representando a seqüência de tarefas da jornada. Uma solução fracionária é um conjunto de "caminhos parciais" do vértice v_0 ao vértice v_f, que representa um conjunto de

jornadas, onde algumas tarefas são incluídas parcialmente em uma e outra jornada. A Figura 5 ilustra esse conjunto de caminhos. Os arcos pontilhados não fazem parte da solução do problema relaxado, e os demais foram selecionados, mesmo que parcialmente.

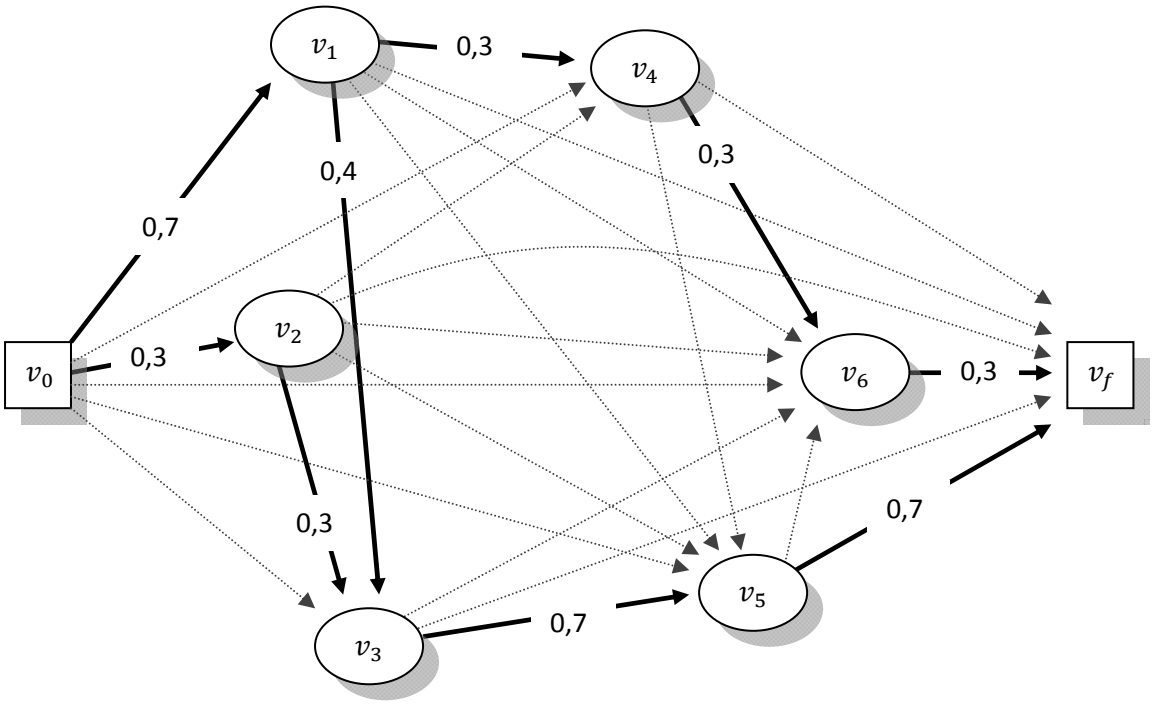

Figura 5: Exemplo de solução fracionária no grafo do subproblema

A heurística é construtiva do tipo *greedy* e é baseada no grafo que originou o modelo do subproblema. Iterativamente é construído um caminho único no grafo, do vértice v_0 ao vértice v_f. Para cada vértice incluído no caminho, escolhe-se como próximo vértice aquele cujo arco de entrada a partir do último vértice incluído tenha o maior valor na solução da relaxação linear. Esse processo continua até atingir o vértice final, descartando os vértices cuja inclusão torne a jornada inviável. O método é mostrado na Figura 6 e explicado em seguida. Desse ponto em diante, o nome RL se refere a essa heurística para solucionar o subproblema.

O passo 5 é o passo *greedy* da heurística, responsável por escolher o próximo vértice do caminho. É escolhido aquele cujo arco tenha o maior valor, entre todos os arcos a partir do vértice escolhido anteriormente. Visto de outra forma, é escolhida a tarefa que aparece, em alguma jornada, em seqüência à tarefa anteriormente escolhida, e cujo valor seja o maior entre os existentes. Se a jornada sendo construída continua viável após a inclusão dessa tarefa, ela é incluída, e continua-se o processo a partir dessa tarefa. Senão, a tarefa não é incluída, e o valor do arco correspondente é zerado, para que outra tarefa seja escolhida na iteração seguinte. Eventualmente nenhuma tarefa poderá ser incluída na jornada, e o vértice final do grafo é escolhido, finalizando o caminho e conseqüentemente a jornada. Os preços duais não são usados diretamente, mas indiretamente, pois eles guiam o valor da solução linear.

Procedure Heurística-RL (SUB_{BH}, G=(V,A))

1. $y \leftarrow$ solução ótima da Relaxação Linear de SUB_{BH}
2. $v \leftarrow v_0$
3. $J \leftarrow \{\}$
4. **while** $v \neq v_f$ **do**
5. $k \leftarrow \{w \in V | \, y_{(v,w)} \geq y_{(v,i)}, \ \forall (v,i) \in A\}$
6. **if** $v_k \neq v_f$ **then**
7. **if** ($J \cup \{k\}$ viável) **then**
3. $J \leftarrow J \cup \{k\}$
4. $v \leftarrow v_k$
5. **else**
6. $y_{(v,k)} \leftarrow 0$
7. **end while**
8. **if** (J viável)
9. Return J
10. **else**
11. Return ∅

end Heurística-RL

Figura 6: Heurística construtiva baseada na solução da relaxação linear do subproblema

A Figura 7 mostra um exemplo de construção de jornada utilizando essa heurística, a partir da solução da relaxação linear mostrada na Figura 5. Seguindo o algoritmo, foram escolhidos os vértices v_0, v_1, v_3, v_5 e v_f, nessa ordem, por causa dos valores dos arcos: (v_0, v_1) – que tem maior valor que (v_0, v_2); depois (v_1, v_3) – que tem maior valor que (v_1, v_4); depois (v_3, v_5) e (v_5, v_f). Dessa forma, a jornada gerada é a seqüência de tarefas 1, 3 e 5.

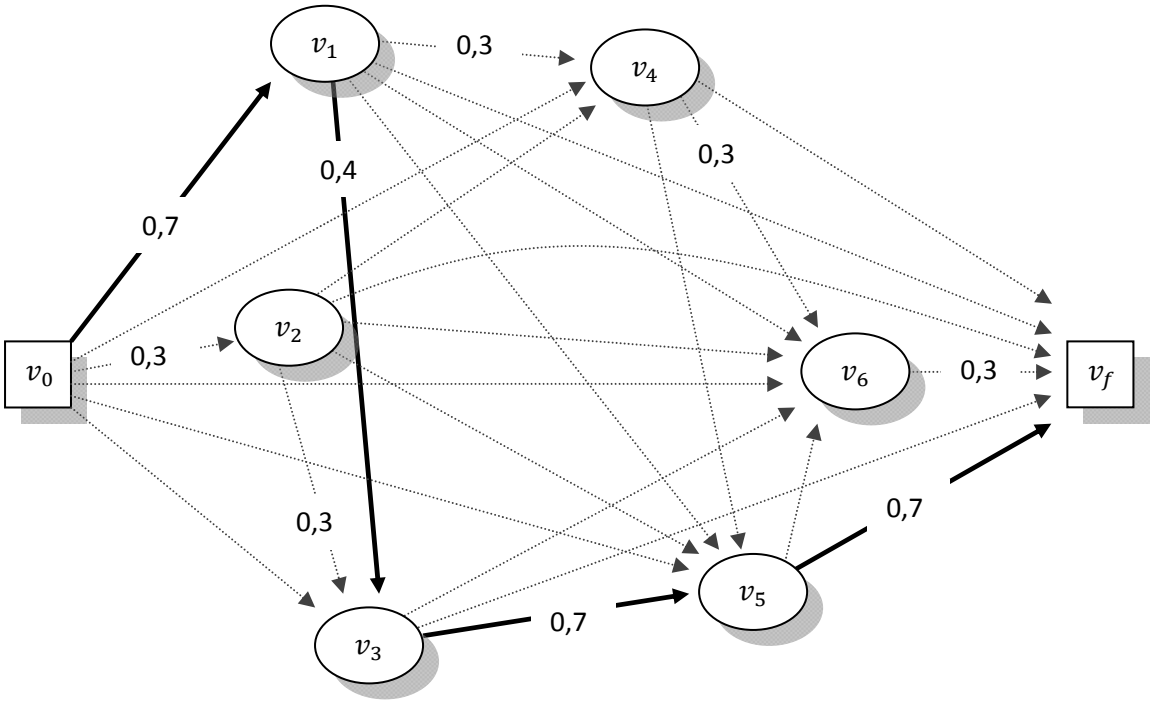

Figura 7: Exemplo de construção de jornada a partir da solução da relaxação linear

Entretanto, mesmo quando finalizada, a jornada pode ainda ser inviável, devido a requisitos que não podem ser verificados durante a construção da jornada. Por exemplo, nas instâncias de transporte urbano da BHTrans deve haver um descanso mínimo de 30 minutos, e pelo menos um intervalo ininterrupto de 15 minutos. Isso só pode ser verificado depois que a jornada é finalizada. Quando isso acontece, a jornada é descartada, e invoca-se novamente o pacote de programação linear inteira para continuar a solução do modelo até obter a solução ótima inteira. Isso também é feito caso a jornada gerada pela heurística não tenha custo reduzido negativo. Outra estratégia seria tentar viabilizar a jornada, mas nesse caso técnicas específicas para cada tipo problema deveriam ser utilizadas. Esse trabalho concentra-se em produzir uma técnica geral que pode ser aplicada a vários tipos de instâncias, e por isso nenhuma técnica de viabilização da jornada foi utilizada.

Resultados experimentais mostram que muitas colunas são adicionadas ao problema mestre, porque algumas das geradas pela heurística não são tão boas jornadas. Como a cada coluna adicionada, o problema mestre é resolvido novamente, o número de vezes que o problema mestre é resolvido é maior quando se usa a heurística. Porém o subproblema é resolvido mais rapidamente quando é utilizada a heurística do que quando somente o método exato é utilizado. Além disso, a solução do problema mestre é computada rapidamente a partir da solução anterior, porque a base ótima anterior pode ser utilizada, e poucos pivoteamentos são necessários. Então, embora o problema mestre e o subproblema sejam resolvidos mais vezes, o processo como um todo é mais rápido. Os resultados computacionais mostrados no capítulo 6 comprovam que o uso da heurística reduziu o tempo total pela metade na maioria das instâncias, com uma redução ainda mais significativa nas instâncias maiores. Diante desses resultados, foram utilizadas também meta-heurísticas para solução do subproblema. Elas são descritas nas duas seções seguintes.

4.2. GRASP – Fase construtiva

A heurística descrita na seção anterior retorna apenas uma jornada cada vez que é invocada. Mas a partir da solução da relaxação linear do subproblema é possível gerar mais caminhos, alguns com custos reduzidos similares. Em vez de se usar uma heurística gulosa determinística, podem ser incorporadas algumas escolhas aleatórias para encontrar rapidamente mais de um caminho no grafo, o que significa produzir mais de uma jornada com os mesmos preços duais fornecidos pelo problema mestre. Assim, a heurística seria adaptada para gerar mais de uma jornada por iteração do método de geração de colunas.

A meta-heurística GRASP considera esses caminhos alternativos no grafo. GRASP é um acrônimo de *Greedy Randomized Adaptive Search Procedure*, introduzido por (Feo & Resende, 1995). GRASP tem duas fases distintas: a primeira é uma heurística construtiva para gerar uma solução; a segunda fase é uma busca local para melhorar a solução gerada. Nesse trabalho não é utilizada a segunda fase, pois no contexto desse trabalho é mais importante gerar rapidamente uma nova jornada que gastar mais tempo tentando melhorá-la. Se for necessário, uma jornada melhor será gerada em futuras iterações da geração de colunas, com a meta-heurística guiada por diferentes preços duais.

Na fase construtiva uma solução é construída passo a passo, incorporando novos elementos em cada iteração. Há uma lista restrita de candidatos que contém os elementos candidatos de maior potencial a serem incluídos na solução. É dessa lista que o próximo elemento da solução é aleatoriamente selecionado. A lista é construída em cada iteração, contendo de 1 a $\alpha\%$ de todos os elementos candidatos viáveis. Se $\alpha = 0$, a lista contém sempre um elemento, sendo, portanto uma heurística gulosa determinística, como a descrita na seção anterior. Se $\alpha = 100\%$, a geração será completamente aleatória. Do contrário, com $0 < \alpha < 100\%$, será uma heurística construtiva gulosa aleatorizada.

Determinado um bom valor do parâmetro α, em sua forma clássica a heurística é executava várias vezes, construindo várias soluções, das quais a melhor é retornada como a solução encontrada pela heurística. Já que a heurística produz várias soluções ao longo de sua execução, nesse caso várias jornadas, optou-se por retornar não uma, mas um conjunto de jornadas de custo reduzido negativo. Assim, a cada vez que o problema mestre envia os preços duais ao subproblema, este retorna várias novas jornadas, que são então incluídas no problema mestre. Dessa forma reduz-se o número de vezes que o problema mestre é resolvido, pois várias colunas são acrescentadas de uma vez, com a desvantagem de aumentar o tamanho do problema mestre de forma mais rápida, às vezes com jornadas não tão úteis na sua solução. Os resultados experimentais comprovam que a vantagem de resolver o mestre e subproblema menos vezes supera a desvantagem de se introduzir algumas colunas sem utilidade. Analisando os *log*s gerados durante a execução foi verificado que isso acontece em grande parte porque dessa forma antecipa-se a inclusão de algumas colunas que futuramente seriam necessárias no problema mestre.

Como é uma heurística, o GRASP pode não encontrar jornadas de custo reduzido negativo, mesmo havendo alguma. Nos casos em que nenhuma é encontrada, o método exato de programação inteira é usado para gerar uma nova jornada, ou provar a otimalidade da solução atual, caso ele também não encontre tal jornada.

Outra melhoria proposta é executar a meta-heurística diretamente no grafo do problema, sem usar os resultados da relaxação linear do subproblema. Ao se resolver o modelo do subproblema, os valores das variáveis de decisão indicam os caminhos de menor custo. Uma parte desses custos é fixa, e a outra é baseada nos preços duais fornecidos pelo problema mestre. Por isso o subproblema gera soluções diferentes em cada iteração da geração de colunas. Já que os preços duais influenciam a solução da relaxação linear, portanto na escolha do caminho, então foi resolvido usar diretamente esses valores. A lista de candidatos é construída contendo os vértices que podem estar após os já acrescentados na solução, ou seja, a lista contém as tarefas que podem seguir as tarefas atuais numa jornada de trabalho viável. As tarefas da lista de candidatos são ordenadas pelo valor decrescente do preço dual, e as primeiras $\alpha\%$ da lista compõem a lista restrita de candidatos. A jornada é gerada "caminhando-se" no grafo, como feito na heurística RL, mas o próximo vértice é escolhido aleatoriamente de uma lista restrita de candidatos de acordo com seu preço dual, e não pelo valor da solução da relaxação linear. Dessa forma evita-se resolver o modelo de programação linear do subproblema, porém os preços duais são ainda usados para guiar a solução do subproblema. A adaptação da meta-heurística GRASP ao problema como descrito acima é mostrada em forma de algoritmo na Figura 8.

Procedure GRASP (G=(V,A), π)

1. $S \leftarrow \{\}$
2. **while not** (critério parada) **do**
3. $v \leftarrow v_0$
4. $J \leftarrow \{\}$
5. **while** $v \neq v_f$ **do**
6. $LC \leftarrow \{\}$
7. **for all** $w \in \delta^+(v) | J \cup \{w\}$ viável **do**
8. $LC \leftarrow LC \cup \{w\}$
9. Ordenar LC pelo preço dual π
10. $t = \max\{1, \alpha|LC|\}$
11. $LRC \leftarrow$ primeiros t elementos de LC
12. $k \leftarrow$ um dos elementos de LRC, escolhido aleatoriamente
13. **if** $v_k \neq v_f$ **then**
14. $J \leftarrow J \cup \{k\}$
15. $v \leftarrow v_k$
16. **end while**
17. **if** (J viável **and** CustoReduzido(J)<0)
18. $S \leftarrow S \cup \{J\}$
19. **end while**
20. Return S

end GRASP

Figura 8: Meta-heurística GRASP para solução do subproblema

O mesmo método, com os mesmos parâmetros, é utilizado da mesma forma em todas as instâncias e para todos os tipos de problema de tripulação aqui testados: o tamanho da lista restrita de candidatos é mantido fixo em 40% do total de candidatos; o critério de parada é a geração de 6 novas jornadas viáveis de custo reduzido negativo, ou 4 falhas na geração de uma nova jornada viável e de custo reduzido negativo. Esses valores foram obtidos empiricamente. A composição da lista restrita de candidatos é feita observando-se os valores dos candidatos, mas a escolha do próximo elemento é feito de maneira aleatória uniforme entre os presentes na lista restrita de candidatos.

O critério de parada utilizado impede a geração de um número grande de novas jornadas a cada iteração, evitando aumentar desnecessariamente o tamanho do problema mestre, e também que a heurística fique por longo tempo tentando encontrar novas jornadas. Caso ela não consiga gerar mais alguma jornada em 4 tentativas, ela retorna o conjunto de jornadas já gerado, mesmo que não tenha gerado 6 jornadas. Caso não tenha gerado nenhuma, o método exato de programação inteira é utilizado para gerar a próxima jornada, ou determinar que não existe mais nenhuma ainda não gerada.

No capítulo 6 são mostrados os resultados obtidos com o uso dessa meta-heurística, parte deles publicados em (Santos & Mateus, 2007). Na próxima seção é apresentada a proposta de uso de outra meta-heurística, um algoritmo genético.

4.3. Algoritmo Genético

Os resultados obtidos com a meta-heurística GRASP encorajaram o uso de meta-heurísticas para gerar mais de uma jornada a cada solução do subproblema. Como algoritmos genéticos trabalham com um conjunto de soluções potenciais, é natural esperar que seja uma excelente opção para resolver o subproblema, na tentativa de inserir várias novas jornadas no problema mestre. De fato os resultados experimentais demonstram que se encaixam bem nesse algoritmo de geração de colunas.

A idéia geral sobre Algoritmos Genéticos pode ser encontrada em muitos livros e artigos. (Michalewicz, 1996) traz tanto um bom referencial teórico sobre os algoritmos genéticos quanto várias aplicações em problemas reais, mostrando como especializar os algoritmos genéticos para alguns deles. Os algoritmos genéticos são amplamente utilizados atualmente para uma vasta gama de problemas de otimização combinatória, incluindo problemas de alocação de tripulações (Kwan, Wren, & Kwan, 2000). Diferentemente do uso tradicionalmente encontrado na literatura, aqui eles são usados apenas para tratar o subproblema, especificamente para acelerar a geração de novas jornadas. Da forma como são incorporados no método de geração de colunas, há uma garantia teórica da obtenção da solução ótima do problema de alocação de tripulações, pois um método exato é usado na solução do subproblema quando o algoritmo genético falha em seu objetivo, gerando as jornadas restantes.

Os algoritmos genéticos imitam a idéia de seleção natural e evolução que se acredita ocorrerem na natureza. Num algoritmo genético existe uma população de indivíduos, cada um representando uma solução em potencial. As características dos indivíduos são codificadas na de forma semelhante aos genes de um cromossomo. Assim, cada cromossomo representa um indivíduo da população, no caso uma possível solução do problema em questão. Como cada solução possui um valor, os cromossomos também podem ser avaliados para produzir um valor, para que sejam comparados uns com os outros qualitativamente. Esses valores representam a adaptação (*fitness*) do indivíduo codificado nesse cromossomo, e indivíduos de melhor adaptação têm mais chance de participar das etapas de reprodução e mutação para gerar novos indivíduos com características semelhantes. A população de indivíduos evolui através de sucessivas gerações (iterações do método), uma geração sendo produzida por recombinação das características dos indivíduos da geração anterior. Os indivíduos de melhor adaptação são selecionados e combinados entre si por operadores de cruzamento. Geralmente dois indivíduos são selecionados como “pais”, e produzem um ou dois “filhos” combinando-se os genes de seus cromossomos. Os indivíduos podem passar ainda por um operador de mutação, que modifica alguns genes de seus cromossomos, introduzindo novas características na população. É um método probabilístico em que, idealmente, uma população inicial produz sucessivas populações, cada vez mais próximas da solução ótima. Isso acontece porque os melhores indivíduos são selecionados para se reproduzirem e gerar novos indivíduos semelhantes para a população seguinte.

Na implementação feita nesse trabalho, um cromossomo codifica uma jornada em um vetor de inteiros, cada gene sendo uma tarefa da jornada. A adaptação (*fitness*) de um

cromossomo é calculada utilizando os custos operacionais normalmente envolvidos na jornada e também os preços duais fornecidos pela solução do problema mestre. Dessa forma a solução é guiada pelo problema mestre, na tentativa de convergir a população de soluções do algoritmo genético em um bom conjunto de jornadas ainda não adicionadas ao problema mestre, como é de esperar em um método de geração de colunas.

Durante todo o algoritmo, as tarefas da jornada codificada em um cromossomo nunca possuem sobreposição de tempo, mas jornadas inviáveis devido a outras restrições são permitidas, como tempo total de trabalho, tempo mínimo de descanso, e outras, dependendo das instâncias que estão sendo consideradas. Tais soluções são mantidas para preservar diversidade na população, mas apenas jornadas viáveis são incluídas em uma população externa de elite. Essa população elite será a solução do algoritmo, retornada ao final de sua execução, conforme mostra a Figura 9.

Para facilitar a aplicação em vários problemas de alocação de tripulação, a população inicial é gerada de forma trivial: são geradas jornadas artificiais contendo apenas uma tarefa. Assim não é necessário desenvolver nenhum método de viabilização das jornadas. Outra forma seria gerar jornadas iniciais utilizando alguma heurística construtiva como as já apresentadas, mas corre-se o risco de criar uma população que abranja apenas uma região do espaço de soluções. Para que todas as tarefas tenham chance de se perpetuar durante o algoritmo, em diferentes combinações, é interessante que todas existam na população inicial, e que apareçam mais de uma vez. Por isso são geradas inicialmente três jornadas para cada tarefa, e assim o tamanho da população inicial é o triplo do número de tarefas. Essas jornadas de uma tarefa são rapidamente recombinadas pelos operadores de seleção, *crossover* e mutação para formar outras jornadas com mais tarefas.

Procedure Genético (G=(V,A), π)

1. $PElite \leftarrow \{\}$
2. $P \leftarrow$ criapopulação(G)
3. **while not** (critério parada) **do**
4. avaliapopulação(P,G, π)
5. atualizaelite($PElite$, P)
6. $P' \leftarrow$ seleção(P)
7. $P'' \leftarrow$ crossover(P')
8. $P''' \leftarrow$ mutação(P'')
9. $P \leftarrow P'''$
10. **end while**
11. Return $PElite$

end Genético

Figura 9: Algoritmo Genético para solução do subproblema

A fase de seleção utiliza o método de torneio por roleta. É construída uma roleta contendo todos os cromossomos, em qualquer ordem. A roleta nesse caso é uma faixa de valores no intervalo $[0,1)$. Para cada cromossomo é dado um pedaço na roleta de acordo com

sua função de adaptação. Supondo uma população de tamanho n, a faixa de valores da roleta $[0,1)$ é dividida em n partes, $[\alpha_0, \alpha_1)$, $[\alpha_1, \alpha_2)$, $\cdots$, $[\alpha_{n-1}, \alpha_n)$, com $\alpha_0 = 0$ e $\alpha_n = 1$. Os cromossomos da população atual são escolhidos aleatoriamente da roleta para criar uma nova população. Supondo uma seleção de m cromossomos para compor a nova população, essa seleção é feita escolhendo-se aleatoriamente valores $s_1, s_2, \cdots, s_m \in [0,1)$. O *i*-ésimo valor $s_i \in [0,1)$ escolhido indica que o *i*-ésimo cromossomo selecionado para a nova população é o cromossomo j da população atual, $1 \leq j \leq n$, tal que $s_i \in [\alpha_{j-1}, \alpha_j)$. Note que quanto maior a parte da roleta dada a um cromossomo j, isto é, quanto maior o valor $\alpha_j - \alpha_{j-1}$, maior a probabilidade do cromossomo j ser selecionado. Por isso cromossomos com menor valor de adaptação recebem proporcionalmente maior pedaço na roleta, e assim têm maior probabilidade de ser selecionado para a próxima fase. Desta forma, cromossomos que representam jornadas com melhor valor na função objetivo, especialmente aqueles com custo reduzido negativo, são perpetuados na população.

Os cromossomos da nova população, escolhidos na fase de seleção, são recombinados via operador de *crossover* e sofrem pequenas modificações via operador de mutação. Na operação de *crossover*, dois cromossomos A e B da população são escolhidos aleatoriamente. Sejam $A_1, A_2, \cdots, A_a$ as tarefas da jornada codificada no cromossomo A, e $B_1, B_2, \cdots, B_b$ as tarefas da jornada codificada no cromossomo B. Uma tarefa $A_i, 1 < i \leq a$, é aleatoriamente escolhida entre as tarefas do cromossomo A, e em seguida é selecionada do cromossomo B a primeira tarefa B_j cujo horário cujo horário inicial seja posterior ao horário final da tarefa A_i escolhida. As tarefas são trocadas de cromossomo, e os cromossomos finais resultantes são $A_1, \cdots, A_i, B_j, \cdots, B_b$ e $B_1, \cdots, B_{j-1}, A_{i+1}, \cdots, A_a$. Pelo método usado na escolha das tarefas nota-se que as tarefas do primeiro cromossomo gerado não se sobrepõem no tempo. Não há garantia que isso não ocorra no segundo cromossomo. Se o horário final da tarefa B_{j-1} é anterior ao horário inicial da tarefa A_{i+1} então não há sobreposição de tempo. Caso contrário a tarefa A_{i+1} é retirada do cromossomo. Nesse caso o processo é repetido para as tarefas seguintes a A_{i+1} até não haver sobreposição de tempo. Como mencionado anteriormente não são verificadas outras restrições de viabilidade da jornada codificada nos cromossomos resultantes. Tais violações são penalizadas na adaptação do cromossomo, de sorte que cromossomos que representam jornadas inviáveis terão menor probabilidade de se perpetuar nas gerações seguintes, mas a diversidade é mantida e boas características podem ainda ser passadas para outros cromossomos da população. Outras vantagens em se manter os inviáveis em vez de se criar um procedimento de viabilização é manter a heurística simples (e rápida) e geral (facilmente adaptável às diferentes instâncias de problemas de alocação de tripulações).

Para evitar convergência prematura e aumentar a diversidade da população, um operador de mutação é aplicado com certa probabilidade. Ele substitui aleatoriamente uma tarefa de um cromossomo por alguma outra que não se sobreponha no tempo com as outras do mesmo cromossomo.

Nota-se que o algoritmo genético implementado é relativamente simples, mas o objetivo principal aqui é gerar rapidamente um conjunto de jornadas diversas, com o direcionamento dado pelos preços duais. Não há necessidade de funções e operadores complexos para evitar mínimos locais de forma mais eficiente, ou gerar sempre as melhores jornadas, porque somente as jornadas de custo reduzido negativo são adicionadas ao

problema mestre. E se uma boa jornada não for descoberta em uma das execuções do algoritmo genético, haverá novas iterações da geração de colunas, ou seja, ele será chamado novamente depois que o problema mestre for resolvido. Além disso, quando não são encontradas boas jornadas, o método exato de solução do subproblema é invocado. Então, mesmo que uma jornada que deveria estar na solução ótima nunca seja descoberta pelo algoritmo genético, em algum momento o método exato de programação linear inteira gera esta jornada. Baseado nos experimentos feitos, o mesmo pode ser dito do critério de parada: é melhor manter um número de gerações baixo, para gerar rapidamente novas jornadas, que esperar uma boa convergência do algoritmo, pois isso gasta mais tempo em cada iteração da geração de colunas, e às vezes uma das jornadas geradas já nas primeiras gerações pode atualizar de forma eficiente os preções duais do problema mestre guiando o subproblema a outro conjunto de jornadas interessantes.

Pela necessidade de geração de um conjunto de jornadas de custo reduzido negativo, de preferência diversificado, há uma população externa que mantém os melhores cromossomos encontrados ao longo da execução do algoritmo. Assim, mesmo que em algum momento os cromossomos do algoritmo genético se concentrem em alguma região do espaço de soluções, se melhores soluções foram encontradas ao longo da execução, elas ainda estarão na população externa. O tamanho dessa população é mantido fixo e baixo, para evitar inundar o problema mestre com várias novas jornadas de uma única vez, pois isso poderia aumentar desnecessariamente seu tamanho.

No capítulo 6 são reportados os resultados obtidos com os algoritmos genéticos, alguns deles publicados em (Santos & Mateus, 2007), e são feitas comparações com as demais heurísticas aqui propostas.

5. Branch-and-Price

O método de geração de colunas resolve a relaxação linear do problema SPP. Quando a solução encontrada não for inteira, parte-se para uma enumeração da árvore de *branch-and-bound*. Se a geração de colunas (*princing*) continua a ser usada em cada nó da árvore, o método é chamado de *branch-and-price*. Nesse trabalho usou-se a técnica de *branch-and-price*, pois novas colunas continuam sendo geradas nos demais nós da árvore, tanto por métodos exatos quanto heurísticos.

A forma mais simples de se fazer um *branch* de uma solução fracionária é o *branch* de variável, explicado na próxima seção. No problema de alocação de tripulações esse tipo de *branch* pode não produzir bons resultados, então uma alternativa de *branch* é discutida na seção seguinte.

5.1. Branch de variável

A primeira tentativa foi fazer *branch* de variável. Entre as variáveis fracionárias da solução linear do problema mestre, ou seja, as variáveis de decisão x_j tais que $0 < x_j < 1$, uma é escolhida para ser fixada em 1 num dos ramos da árvore de enumeração *branch-and-bound* e em 0 no outro ramo. Isto corresponde a escolher uma jornada ou proibir sua escolha. O processo é idêntico para todas as instâncias, seja para os modelos SUB_{OR}, SUB_{BH} ou SUB_{A}. Não são consideradas as demais variáveis de decisão, pois elas não assumem valores fracionários quando essas não assumem.

Desta forma, em um dos ramos da árvore acrescenta-se a restrição (56) no problema mestre, e as restrições (57) no subproblema, em que $C(j)$ é o conjunto das tarefas cobertas pela jornada j. Isto faz que com que a jornada correspondente esteja presente, integralmente, na solução do problema mestre, e indica ao subproblema que nenhuma das tarefas já cobertas por essa jornada deve fazer parte das novas jornadas geradas. Como a jornada já fará parte da solução, as novas jornadas devem conter apenas as demais tarefas não cobertas por essa jornada.

$$x_j = 1 \tag{56}$$

$$v_t = 0, \forall t \in C(j) \tag{57}$$

No outro ramo a seleção dessa jornada é proibida, adicionando-se a restrição (58) ao problema mestre. Como essa jornada fora escolhida parcialmente para compor a solução, é uma boa jornada. E se ela for proibida no problema mestre, o subproblema gerará novamente esta jornada. Para impedir que o subproblema gere uma jornada idêntica, são acrescentadas as duas restrições (59) e (60). Se a nova jornada não contém todas as tarefas da jornada j, a restrição (59) deixa a variável f livre para assumir o valor 1, e a restrição (60) fica sem efeito. Porém, se a nova jornada contém todas as tarefas da jornada j, então f = 0 pela restrição (59), e a restrição (60) obriga que pelo menos alguma outra tarefa esteja na jornada. Incluindo essas

duas restrições é garantido que a nova jornada não será igual à jornada proibida neste nó da árvore, pois ou a nova jornada não contém todas as tarefas da jornada proibida, ou contém todas e mais alguma.

$$x_j = 0 \tag{58}$$

$$\sum_{t \in C(j)} v_t + f \leq |C(j)| \tag{59}$$

$$\sum_{t \notin C(j)} v_t + f \geq 1 \tag{60}$$

Esse tipo de *branching* dificulta a utilização das heurísticas na solução do subproblema. Do lado onde a jornada é selecionada, em que se faz $x_j = 1$no problema mestre, as heurísticas funcionariam caso fossem retirados do grafo os vértices das tarefas cobertas pela jornada. Porém, do lado onde a jornada é proibida, é trabalhoso impedir que a heurística gere a mesma jornada. Elas teriam que ser modificadas para incluir esta restrição adicional. Por isso, com esse tipo de *branching*, as heurísticas são usadas apenas no nó raiz da árvore. Nos demais, apenas o método exato de programação linear inteira é utilizado.

Outro tipo de *branching* é apresentado na próxima seção. Ele tem a grande vantagem de permitir o uso das heurísticas em todos os nós da árvore de *branch-and-bound* sem nenhuma modificação em suas implementações. Isso é possível porque as modificações necessárias para impedir a geração de jornadas idênticas podem ser feitas diretamente no grafo do subproblema.

5.2. Branch follow-on

Outro tipo de *branching*, proposto em (Ryan & Foster, 1981) com o nome de *follow-on*, e utilizado em outros trabalhos (Barnhart, Johnson, Nemhauser, Savelsbergh, & Vance, 1998), é particularmente útil nesse trabalho. Eles demonstraram que se a solução da relaxação linear do problema mestre SPP for fracionária, então existe pelo menos um par de tarefas t_a e t_b onde vale a propriedade (61), sendo $AB \subseteq T$ o conjunto de todas as jornadas em que as tarefas t_a e t_b aparecem em seqüência.

$$0 < \sum_{j \in AB} x_j < 1 \tag{61}$$

Numa solução viável, inteira, essas tarefas devem ser cobertas exatamente uma vez. Então, ou as tarefas t_a e t_b aparecem em seqüência em alguma jornada da solução, ou elas não aparecem em seqüência em nenhuma jornada da solução. O *branching* consiste em se fixar essa soma em 1 em um dos ramos da árvore e em 0 no outro ramo.

Escolhido um par de tarefas t_a e t_b para o qual (61) é verdadeira, identifica-se o conjunto $AB \subseteq T$ de jornadas nas quais elas aparecem em seqüência. De um lado da árvore de

enumeração *branch-and-bound* acrescenta-se a restrição (62) no problema mestre, e nenhuma modificação é necessária no subproblema.

$$\sum_{j \in AB} x_j = 1 \quad \textbf{(62)}$$

Do outro lado da árvore acrescenta-se a restrição (63) no problema mestre, e retira-se o arco (v_a, v_b) do grafo do subproblema, o que corresponde a se fazer $y_{(a,b)} = 0$ no modelo do subproblema. Desta forma impede-se a geração de uma nova jornada com essas tarefas em seqüência.

$$\sum_{j \in AB} x_j = 0 \quad \textbf{(63)}$$

Como a única modificação necessária no subproblema pode ser feita diretamente no grafo, todas as heurísticas e meta-heurísticas continuam válidas, pois todas geram jornadas representadas por caminho no grafo. Desta forma, com esse tipo de *branching*, elas podem ser utilizadas em todos os nós da árvore de *branch-and-bound* sem nenhuma modificação em seu funcionamento.

6. Análise dos Resultados

Os algoritmos foram implementados na linguagem C, utilizando chamada de funções da Xpress-MP (Dash Optimization Ltd, 2006) para resolver os modelos de programação linear e de programação linear inteira. Os resultados foram obtidos pela execução dos métodos em um computador com processador Intel Core 2 Duo 1.80 GHz, com 2Gb de memória RAM, com sistema operacional Windows Vista Home Premium.

Nas próximas duas seções são apresentados alguns resultados obtidos empregando-se as várias formas propostas de solução do subproblema para instâncias retiradas da literatura, e os métodos são comparados. Em seguida são detalhados os resultados de expansão da árvore de *branch-and-bound* para as poucas instâncias onde a solução obtida no nó raiz não é inteira. E por fim são descritos os resultados obtidos com as instâncias aéreas, geradas para teste dos métodos aqui propostos, além de comentários sobre outras aplicações.

6.1. Instâncias da OR-Library

A OR-Library possui algumas instâncias para o problema de alocação de tripulações. A Tabela 7 apresenta algumas características dessas instâncias. Para cada instância são apresentados o número de tarefas e a quantidade de jornadas viáveis, calculada por enumeração exaustiva. A contagem foi feita contando-se, por *backtracking*, os caminhos do vértice inicial até o vértice final no grafo do subproblema, cortando-se a busca tão logo um caminho represente uma jornada inviável.

Tabela 7: Características das instâncias da OR-Library

Instância	Tarefas	Jornadas
Csp50	50	265
Csp100	100	1184
Csp150	150	2526
Csp200	200	5495
Csp250	250	11565
Csp300	300	25770
Csp350	350	27624

As instâncias fornecem o horário inicial e o horário final de cada tarefa, e também todos os custos envolvidos no problema, nesse caso os custos de transição entre cada par de tarefas i e j em que j pode seguir i em alguma jornada. O custo das jornadas é a soma dos custos de transição entre as tarefas da jornada. O número de jornadas na solução ótima, NJ, que corresponde ao tamanho da tripulação, é um valor estabelecido a priori, mas não é um dado do problema. Seguindo a idéia de (Beasley & Cao, 1996), cada instância é resolvida para vários tamanhos de tripulação. Embora tenham sido aplicados em vários tamanhos diferentes,

os experimentos aqui relatados referem-se apenas ao tamanho mínimo para o qual existe uma solução viável, e mais um ou dois tamanhos acima desse.

A Tabela 8 mostra os resultados obtidos com o algoritmo de geração de colunas, resolvendo o subproblema: somente com programação linear inteira, ou seja, solução do subproblema pelo modelo SUB_{OR} (colunas PLI); com a heurística sobre a relaxação linear (colunas RL); e com a meta-heurística Grasp (colunas GR). A base inicial do problema mestre consiste em jornadas artificiais de custo alto, uma jornada para cada tarefa. Em seguida novas jornadas são geradas com a solução do subproblema pelos métodos citados. Para cada método de solução são apresentados o tempo de execução em minutos (coluna CPU) e a quantidade de jornadas inseridas no problema mestre durante a execução. Todos esses dados referem-se à resolução da relaxação linear do problema mestre, cujo valor é mostrado na coluna Z_{RL}. O mesmo resultado é encontrado por todos os métodos, pois mesmo quando resolvido heuristicamente, com os métodos RL e GR, na falha de geração de nova coluna, o método PLI é utilizado, garantindo que o problema mestre relaxado seja resolvido de forma exata. A coluna Z lista o valor da solução ótima inteira para os poucos casos onde a solução da relaxação linear não é inteira, sendo o problema resolvido através de *branch-and-price*. Os dados referentes à expansão da árvore de *branch-and-bound* são relatados mais à frente.

Tabela 8: Resultados para as instâncias da OR-Library

Instância	NJ	Z_{RL}	Z	Jornadas geradas			CPU (minutos)		
				PLI	RL	GR	PLI	RL	GR
Csp50	29	2399	2399	73	94	150	0,06	0,03	0,02
	28	2706		80	93	136	0,09	0,04	0,03
	27	3139		81	95	146	0,09	0,05	0,06
Csp100	44	4812		204	242	359	1,0	0,6	0,5
	43	5130		195	253	343	1,2	0,8	0,6
	42	5458		200	234	322	1,7	0,9	1,1
Csp150	72	5550,5	5551	306	362	597	3,1	1,4	1,1
	69	6249,5	6275	321	348	593	5,4	3,1	2,2
	67	7164	7164	309	376	507	9,1	7,3	6,8
Csp200	88	7979		436	507	761	9,3	6,8	6,2
	87	8244		445	541	734	18,5	13,8	11,2
	86	8597,25	8634	459	522	727	15,3	11,5	9,9
Csp250	112	7707		622	660	1122	13,1	6,8	6,6
	111	7863		694	745	1225	22,0	9,3	8,6
Csp300	131	9378		888	880	1448	55,9	22,9	24,9
	130	9580		805	850	1344	40,5	29,9	26,8

A Tabela 9 mostra o ganho no tempo de execução proporcionado pelas heurísticas em relação ao método exato. Mostra também o aumento percentual no número de colunas do problema mestre. Grosso modo, pode ser dito que o uso da heurística RL reduziu em média 40% o tempo de execução, quando comparado com o uso somente do modelo exato SUB_{OR}. Em alguns casos a redução não foi tão significativa, como para a instância Csp150 com $NJ = 67$, em que o tempo do algoritmo com a heurística RL foi de 7,3 minutos contra 9,1 do método PLI, uma redução de apenas 20%.

Em contrapartida, o uso da heurística acrescenta mais colunas no problema mestre, 15% em média. Como a cada nova coluna, o problema mestre é resolvido novamente, constata-se que o número de iterações do algoritmo de geração de colunas é maior quando se usa o método RL que quando usado o método PLI. Um resultado já esperado, visto que o método PLI sempre acrescenta, teoricamente, a melhor coluna no momento. Apesar de o problema mestre ser resolvidos mais vezes, o subproblema é resolvido mais rapidamente, e assim o tempo de execução da geração de colunas é menor, comprovando o ganho com a solução heurística do subproblema.

Tabela 9: Percentual de redução no tempo de execução e aumento do número de colunas

Instância	NJ	%Jornadas		%CPU	
		RL	GR	RL	GR
Csp50	29	29	105	47	65
	28	16	70	51	71
	27	17	80	50	33
Csp100	44	19	76	42	50
	43	30	76	36	48
	42	17	61	46	35
Csp150	72	18	95	54	66
	69	8	85	42	58
	67	22	64	20	25
Csp200	88	16	75	27	33
	87	22	65	26	39
	86	14	58	25	35
Csp250	112	6	80	48	50
	111	7	77	58	61
Csp300	131	-1	63	59	55
	130	6	67	26	34
MÉDIA		+15%	-40%	+74%	-48%

Os resultados comprovam ainda que o uso da meta-heurística Grasp auxilia a solução do problema. Mesmo acrescentando mais colunas que todos os outros, a solução é encontrada mais rapidamente, em cerca de metade do tempo. Isso se deve a dois motivos principais: primeiramente, a heurística é baseada no grafo do subproblema, e o modelo SUB_{OR}, cuja solução é demorada, não é utilizado, nem mesmo sua relaxação linear (a não ser, é claro, quando o Grasp não encontra nova jornada, e então o modelo SUB_{OR} é resolvido de forma exata); em segundo lugar, cada vez que o subproblema problema é resolvido, não uma, mas várias colunas são acrescentadas no problema mestre. Logo, o número de iterações da geração de colunas pode diminuir. Isso é comprovado em resultados mais detalhados apresentados adiante para as instâncias da BHTrans. Assim, apesar do problema mestre final ser maior que quando usado os outros métodos, pois possui um número bem maior de jornadas, ele é resolvido menos vezes, e resolvido facilmente pelo Xpress com base na solução anterior, pois apenas algumas colunas são acrescentadas em cada iteração.

Para reforçar essa conclusão, pode-se verificar que em 13 dos 16 casos reportados, o tempo de execução foi menor que o método RL, que já era menor que o PLI. As únicas exceções são Csp50 com 27 tripulantes, Csp100 com 42 tripulantes, e Csp300 com 131

tripulantes. Mesmo para esses casos, o tempo de execução poderia ser menor se fossem utilizados outros parâmetros para o método, mas aqui são utilizados os mesmos parâmetros para todas as instâncias e tipos de problemas.

Outro resultado importante é a rápida aproximação da solução ótima dos métodos heurísticos. O gráfico da Figura 10 mostra o resultado para a instância Csp150 com 67 tripulantes, um dos casos em que as heurísticas menos melhoraram o tempo de execução. Nota-se que mesmo com tempo de execução não muito diferentes, o valor da solução melhora bastante já no início da geração de colunas, aproximando-se mais rapidamente do valor da solução ótima que com a utilização do método exato PLI. É um fato importante, pois se houver necessidade de interrupção da geração de colunas antes do final, por questão de urgência na obtenção de uma solução viável, a utilização das heurísticas fornece uma solução de valor bem mais próximo da solução final. Para as outras instâncias esse resultado é ainda mais acentuado, já que a redução no tempo de execução é ainda maior. De fato, os *log*s produzidos durante os experimentos comprovam que a solução ótima, em vários casos, é obtida várias iterações antes de ser comprovada sua otimalidade.

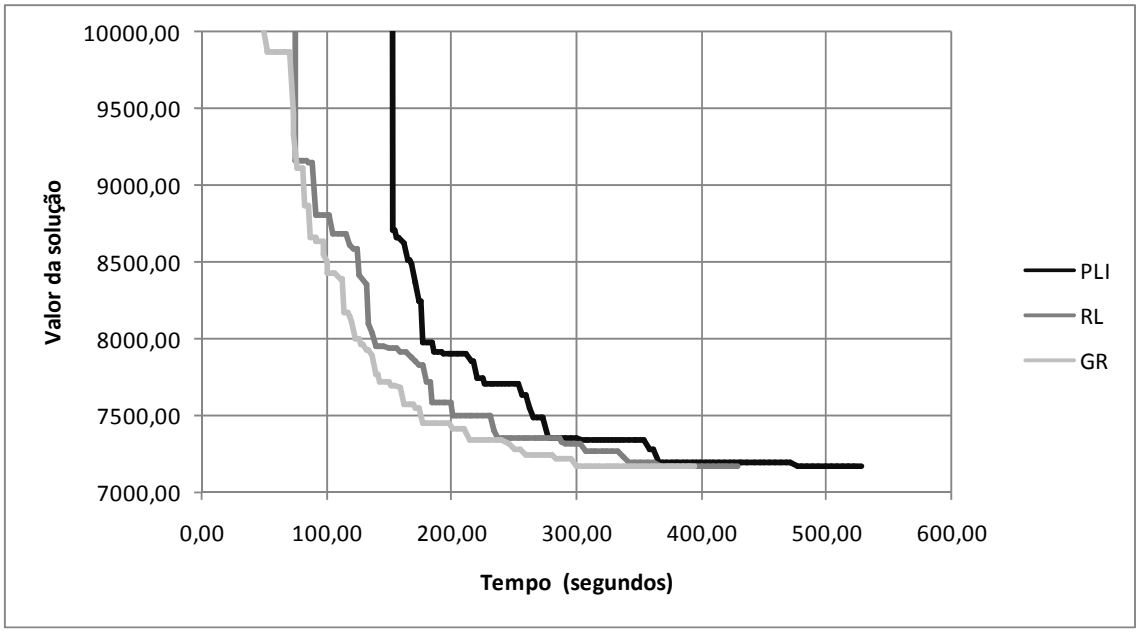

Figura 10: Comparação do valor da solução com o tempo de execução (Csp150 com 67 tripulantes)

6.2. Instâncias da BHTrans

O conjunto de dados contém 733 tarefas, divididas em 11 diferentes linhas de ônibus. Para cada tarefa, são fornecidos os seguintes dados: número do veículo utilizado; horário inicial e final da tarefa, em minutos; ponto inicial e final da tarefa; tempo de repouso entre viagens da tarefa; número da linha de início e término da tarefa.

O número da linha de início e término é o mesmo para cada tarefa, embora possa mudar de uma tarefa para outra. Nas instâncias utilizadas trocas de linha são permitidas

apenas entre linhas pertencentes a um mesmo grupo. Como o grupo a que pertence cada linha não faz parte da entrada, decidiu-se não permitir trocas de linhas. A Tabela 10 resume as características desta entrada, dividindo-a entre as linhas operadas pela empresa.

Tabela 10 Resumo dos dados das instâncias da BHTrans

Nome da Instância	Linha (itinerário)	Número de Tarefas	Número de Veículos
L101-40	101	40	3
L201-49	201	49	2
L321-54	321	54	3
L1170-54	1170	54	5
L2104-94	2104	94	17
L2152-43	2152	43	6
L4150-63	4150	63	9
L5201-75	5201	75	12
L8207-101	8207	101	18
L8208-78	8208	78	12
L9206-82	9206	82	14

Na Tabela 11 são mostrados alguns resultados para estas instâncias da BHTrans. Os resultados são o número de jornadas geradas e o tempo gasto pela geração de colunas utilizando cada um dos métodos propostos para solução do subproblema: solução exata por programação linear inteira (PLI), heurística *greedy* sobre a relaxação linear (RL), e meta-heurísticas GRASP (GR) e algoritmo genético (AG). Essas instâncias possuem menos tarefas que as da OR-Library, mas a duração das tarefas é menor, logo um número maior de tarefas pode fazer parte de uma jornada, elevando bastante a quantidade de jornadas viáveis diferentes. Os dados da coluna Jornadas viáveis foram gerados por enumeração exaustiva, feita por *backtracking* no grafo do subproblema. A contagem consiste então em se contar os caminhos do vértice inicial ao vértice final que representam jornadas viáveis. É interessante usar o grafo, pois nele as tarefas já estão ordenadas e sem sobreposição de horário. Para acelerar a contagem, um caminho é interrompido tão logo a jornada representada se torne inviável.

Em todas as instâncias são utilizados os seguintes custos na geração das jornadas de trabalho, contabilizados no subproblema: custo por minuto excedente no tempo da jornada, c_h = 0,2; custo por minuto de tempo ocioso, c_r = 0,15; finalmente, o custo c_a de uma aresta no grafo do subproblema, que corresponde à transição entre duas tarefas consecutivas, é dado por 19 se há troca de veículo, mais 5, se há cada troca de estação, ou então 0, se não houver nenhuma troca. No problema mestre, todos esses custos são somados para compor o custo c_j da jornada de trabalho j, havendo ainda um custo de penalização b = 45 por dupla pegada além da quantidade permitida. Os primeiros resultados consideram que todas as jornadas do tipo dupla pegada são penalizadas.

Com exceção dos custos c_h e c_r, todos os demais são os custos empregados em (Marinho, 2005) e (Souza, Cardoso, Silva, Rodrigues, & Mapa, 2004). Os custos de hora extra e de tempo ocioso são diferentes porque nesses trabalhos tais custos não eram lineares.

Como o algoritmo genético permite soluções inviáveis em sua população de potenciais soluções, são definidas algumas penalidades para essas soluções: 1000 por troca de estação proibida, ou por mais de um intervalo de 2 horas de duração entre jornadas, e 30 por minuto além das 2 horas extras permitidas. Além desses valores, são utilizados os seguintes parâmetros para todas as instâncias; o critério de parada é o número de iterações, fixado em 40; a probabilidade de um cromossomo participar do *crossover* é de 90% e de sofrer mutação 10%; a população elite tem um tamanho máximo de 10 cromossomos, logo no máximo 10 novas jornadas são retornadas a cada solução do subproblema.

Os problemas são resolvidos de forma exata, então todas as soluções encontradas possuem o mesmo valor, e por isso nem são mostrados na tabela. Esses resultados são apenas da relaxação linear do problema mestre. Resultados mais detalhados, para os casos em que a solução linear não é inteira, são apresentados na próxima seção.

Tabela 11: Resultados para as instâncias da BHTrans

Instância	Jornadas viáveis	Jornadas geradas				CPU (minutos)			
		PLI	RL	AG	GR	PLI	RL	AG	GR
BH-L101-40	1.037.190	255	298	480	487	1,0	0,7	0,8	0,7
BH-L201-49	>4.000.000	445	535	983	862	3,0	3,0	4,6	3,1
BH-L321-54	>4.000.000	947	856	1.229	1.476	8,5	7,0	9,6	9,6
BH-L1170-54	292.505	503	565	683	853	2,9	2,1	2,4	2,1
BH-L2104-94	11.828	332	413	771	747	14,9	8,2	9,0	6,2
BH-L2152-43	10.045	194	244	385	546	0,8	0,5	0,3	0,6
BH-L4150-63	9.243	156	176	422	413	5,3	3,9	1,7	1,4
BH-L5201-75	9.788	199	264	630	611	9,9	6,6	2,5	2,6
BH-L8207-101	19.996	?	380	881	864	>30,0	12,8	5,0	6,1
BH-L8208-78	18.160	320	378	731	791	8,1	6,4	2,1	3,1
BH-L9206-82	16.621	283	349	700	885	16,3	7,9	2,8	3,9

Aqui as vantagens da geração de colunas são mais evidentes. Não é reportado na tabela, mas para instâncias com "pequeno" número de jornadas viáveis, como L2104-94, L2152-43, L4150-63 e L5201-75, enumerar todas as jornadas e resolver o modelo de partição de conjuntos com todas elas leva menos tempo que o algoritmo de geração de colunas. Mas, para as demais instâncias, a enumeração exaustiva é superada pelo método de geração de colunas. Para a instância L101-40, por exemplo, foram gastas mais de 3 horas apenas para contar todas as jornadas viáveis, sem nem mesmo inseri-las no modelo SUB_{BH}. Para os modelos L201-49 e L321-54, o programa de geração exaustiva foi interrompido após 5 horas, e já contava mais de 4 milhões de jornadas. Entretanto, usando-se o método de geração de colunas, o problema pode ser resolvido (pelo menos a relaxação linear) em apenas alguns minutos, sendo geradas geralmente menos de 1000 jornadas.

Uma observação importante é que o desempenho do método de geração de colunas não é muito influenciado pela quantidade de jornadas viáveis existentes, mas pela quantidade de tarefas a serem cobertas. Por exemplo, o método gasta menos tempo para resolver as instâncias com milhões de jornadas viáveis do que para resolver as instâncias L9206-82, L2104-94 e L8207-101, que possuem mais tarefas, mesmo que essas possam ser combinadas em

apenas milhares de jornadas. De fato, essas são as instâncias com mais tarefas, e as únicas onde a geração de colunas com o método exato na solução do subproblema gasta mais de 10 minutos para encontrar a solução linear.

Os resultados podem ser vistos de forma gráfica na Figura 11. Note que as meta-heurísticas apresentam uma grande vantagem, principalmente nas instâncias mais difíceis.

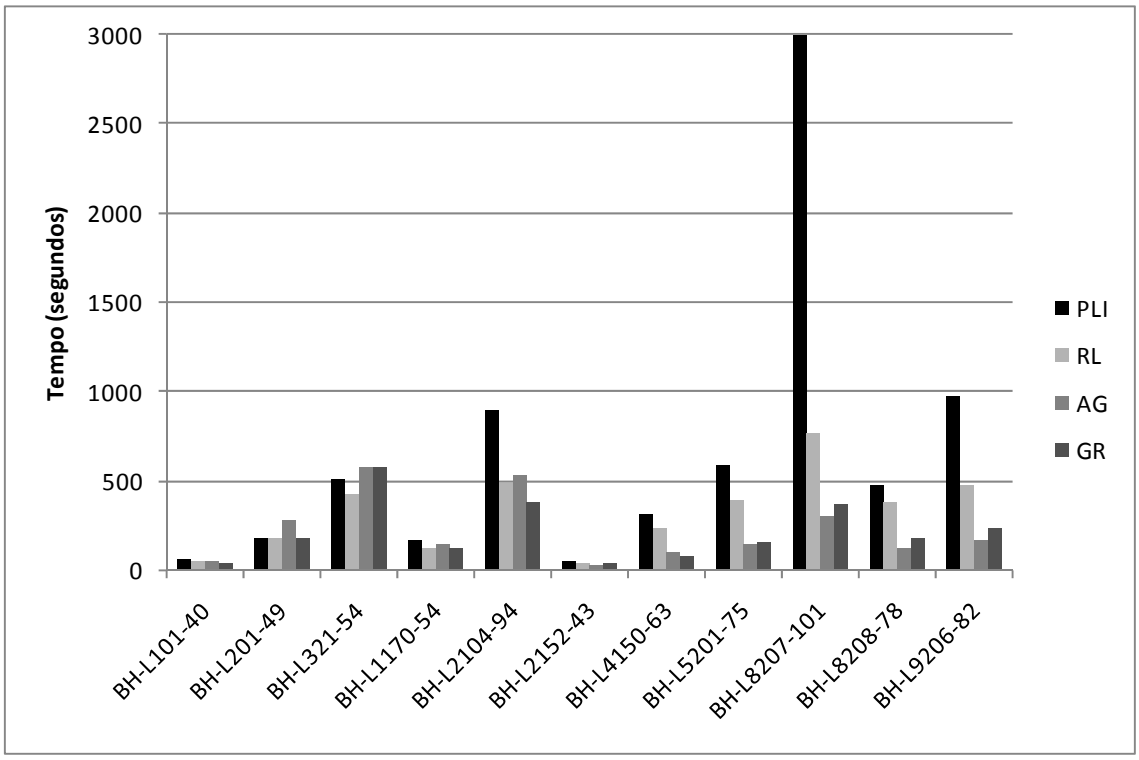

Figura 11: Gráfico comparativo do tempo de execução dos vários métodos para as instâncias da BHTrans

A Tabela 12 mostra esses resultados em termos percentuais, indicando a porcentagem de aumento no número de jornadas acrescentadas pelos métodos RL, AG e GR em relação ao método PLI, e a redução percentual no tempo de execução de cada um desses métodos em relação ao PLI. Pode-se dizer que o uso do método RL reduz em média 34% do tempo de execução, em relação ao método PLI, chegando a ser metade do tempo em instâncias grandes, como L2104-94 e L9206-82. Essa redução no tempo é acompanhada de um pequeno aumento (12% em média) no número de jornadas geradas. Exceto para as instâncias com grande quantidade de jornadas viáveis, L201-40 e L321-54, o tempo de execução utilizando as meta-heurísticas é ainda menor, chegando a ser metade do método RL para várias das instâncias. E a redução é maior para as instâncias com maior número de tarefas, que são mais difíceis de serem resolvidas, mostrando que à medida que as instâncias aumentam de tamanho, a aplicação das meta-heurísticas torna-se ainda mais interessante. Por exemplo, para a instância L9206-82, empregando-se o método PLI o algoritmo termina em mais de 16 minutos, contra 7,9 com o método RL, e em apenas 2,8 e 2,9 com os métodos AG e GR respectivamente, uma redução de cerca de 80%. Para a instância L8207-101 o método PLI não consegue gerar as jornadas para encontrar a solução em menos de 30 minutos, enquanto o algoritmo genético e o Grasp fazem isso em apenas 5 ou 6 minutos.

Tabela 12: Resultado comparativo das heurísticas em relação ao método PLI

Instância	PLI		RL		AG		GR	
	Jorn.	CPU	%Jorn	%CPU	%Jorn	%CPU	%Jorn	%CPU
BH-L101-40	255	1	17	30	88	20	91	30
BH-L201-49	445	3	20	0	121	-53	94	-3
BH-L321-54	947	8,5	-10	18	30	-13	56	-13
BH-L1170-54	503	2,9	12	28	36	17	70	28
BH-L2104-94	332	14,9	24	45	132	40	125	58
BH-L2152-43	194	0,8	26	37	98	62	181	25
BH-L4150-63	156	5,3	13	26	171	68	165	74
BH-L5201-75	199	9,9	33	33	217	75	207	74
BH-L8207-101*	*?*	*> 30,0*	*380*	*12,8*	*132*	*61*	*127*	*52*
BH-L8208-78	320	8,1	18	21	128	74	147	62
BH-L9206-82	283	16,3	23	52	147	83	213	76
MÉDIA PONDERADA			+12%	-34%	+93%	-49%	+111%	-53%

**Nessa instância específica, as colunas AG e GR são comparadas com as colunas RL*

Os dados da Tabela 13 mostram o número de iterações da geração de colunas para cada um dos métodos usados na solução do subproblema. Esse número corresponde à quantidade de vezes que o problema mestre e o subproblema são resolvidos. Esses dados comprovam que, apesar de acrescentar cerca do dobro de jornadas no problema mestre (segundo resultados da Tabela 12), o uso das meta-heurísticas pode de fato diminuir o número de vezes que o problema mestre é resolvido. Assim, apesar do problema mestre ter mais colunas, e conseqüentemente poder se tornar mais complexo de ser resolvido, ele é resolvido menos vezes, e conseqüentemente também o subproblema é resolvido menos vezes, o que se traduz em redução no tempo de execução. Ao utilizar o algoritmo genético, por exemplo, a geração de colunas teve menos iterações em 7 das 10 instâncias testadas (excluindo a instância L8207-101, para a qual não se têm o número de iterações do método PLI).

Tabela 13: Comparação do número de iterações da geração de colunas para as instâncias da BHTrans

Instância	Número de iterações			
	PLI	RL	AG	GR
BH-L101-40	255	298	191	253
BH-L201-49	445	535	517	526
BH-L321-54	947	856	890	1158
BH-L1170-54	503	565	384	493
BH-L2104-94	332	413	322	317
BH-L2152-43	194	244	133	247
BH-L4150-63	156	176	158	178
BH-L5201-75	199	264	218	237
BH-L8207-101	?	380	295	364
BH-L8208-78	320	378	242	348
BH-L9206-82	283	349	226	345

É interessante notar também o que ocorre nessas iterações, particularmente quando os métodos heurísticos conseguem gerar jornadas para o problema mestre, e quando o método exato precisa ser utilizado. Esses dados são mostrados graficamente para duas das instâncias nas figuras seguintes. Para facilitar a visualização, o número de iterações é normalizado, e as jornadas geradas são agrupados de 5 em 5% das iterações.

Na Figura 12 são apresentados os resultados da heurística RL. Nesse caso o número de jornadas geradas em relação ao número de iterações é constante, pois apenas uma nova jornada é gerada em cada iteração. As pequenas variações são causadas meramente por arredondamentos na normalização. O comportamento é semelhante para as duas instâncias: na primeira metade das iterações a grande maioria das jornadas é gerada pela heurística RL; mas a partir daí esse número começa a decrescer até que nas últimas iterações ela praticamente não consegue mais gerar novas jornadas, recorrendo-se ao método PLI em quase todas as iterações. Vale dizer que nos últimos 25% das iterações o valor da solução já estava a menos de 1% do valor da solução final, portanto dificilmente uma heurística gera boa jornada.

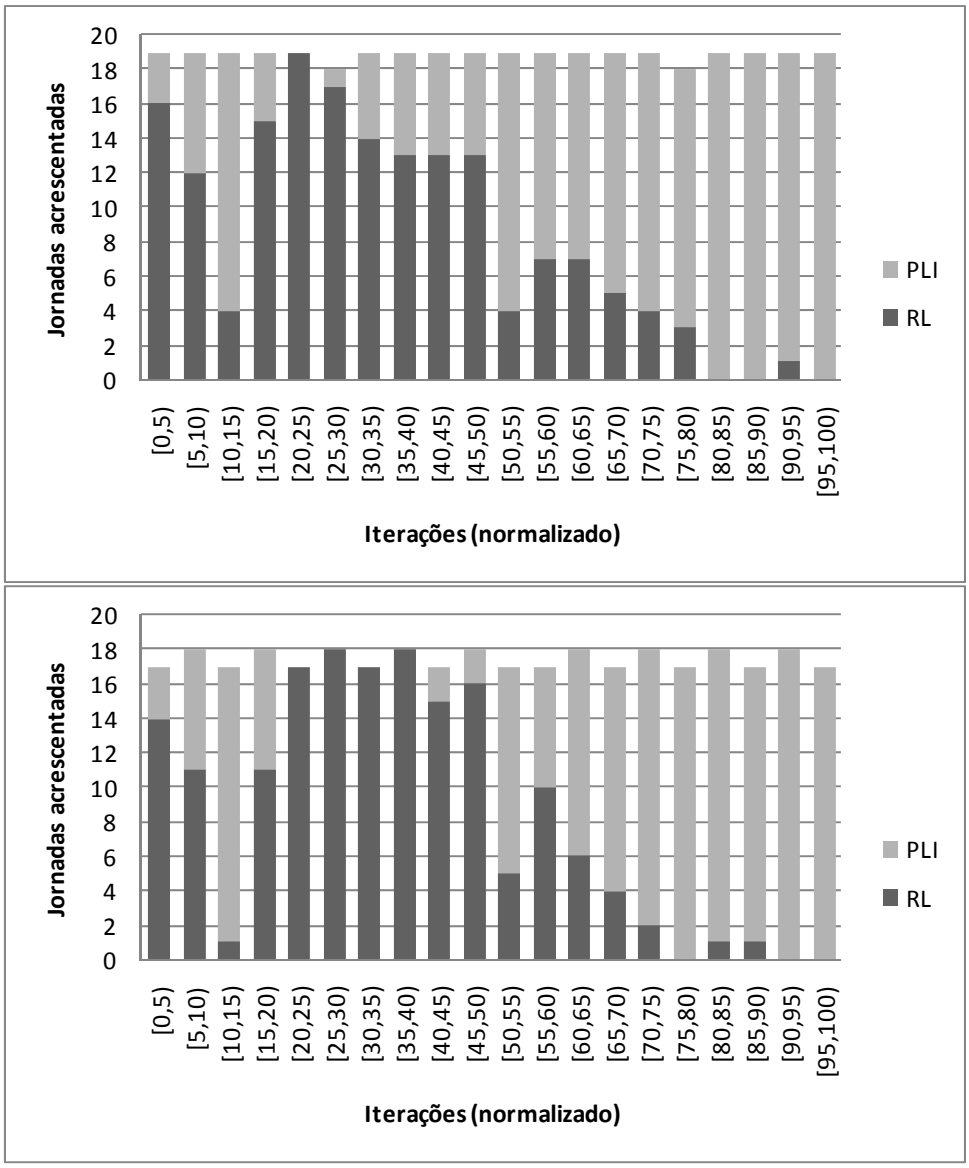

Figura 12: Geração de jornadas pela heurística RL e método PLI nas instâncias L8208-78 (a) e L9206-82 (b)

A Figura 13 apresenta os resultados para a meta-heurística GRASP. Como esse método pode gerar mais de uma jornada cada vez que resolve o subproblema, o número de jornadas geradas por iteração varia durante a execução da geração de colunas, sendo bem maior nas iterações iniciais, quando a meta-heurística é capaz de gerar muitas novas jornadas. Para a instância L9206 a geração de jornadas pelo GRASP é um pouco mais intensa e continua por mais tempo, mas para ambas as instâncias é possível perceber que na primeira metade das iterações, em pouquíssimas vezes é necessário se recorrer ao método PLI. O GRASP continua gerando jornadas por mais tempo que a heurística RL apresentada na figura anterior, e nas últimas 4 colunas do gráfico (últimos 20% de iterações), quando ele consegue gerar poucas jornadas, o valor da solução já havia sido atingido (na instância L8207-78) ou foi modificado apenas mais uma vez (instância L9206-82). Como a base da solução do SPP é bastante degenerada, várias jornadas de custo reduzido negativo são acrescentadas nas últimas iterações sem modificar o valor da solução. Tais jornadas são necessárias para garantir a terminação da geração de colunas, mas dificilmente seriam descobertas por métodos heurísticos, pois possuem custo reduzido negativo, mas bem próximos de zero.

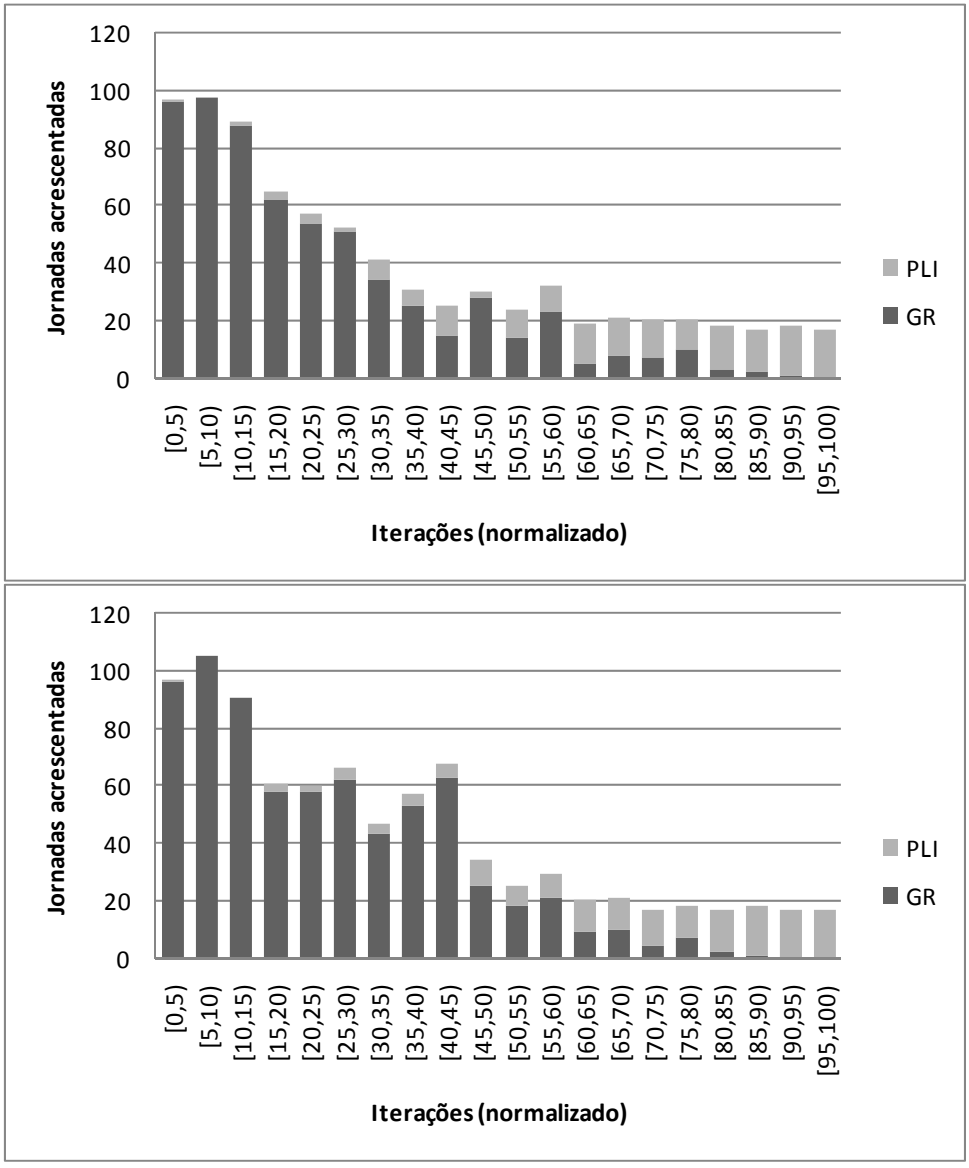

Figura 13: Geração de jornadas por GRASP e método PLI nas instâncias L8208-78 (a) e L9206-82 (b)

A Figura 14 mostra os resultados do algoritmo genético para as mesmas instâncias. Nota-se que o comportamento é semelhante ao do GRASP: na primeira metade das iterações a grande maioria das jornadas é gerada pelo algoritmo genético, e nas últimas iterações torna-se necessário recorrer mais vezes ao método PLI. Entretanto, como a escala dos gráficos é a mesma, é possível perceber que o método PLI é usado menos vezes nas últimas iterações, levando a crer que o algoritmo genético conseguiu prever melhor as jornadas necessárias para encontrar a solução e garantir a terminação da geração de colunas. Isso influencia o tempo de execução do algoritmo, pois contribui diretamente para a diminuição do número de iterações. Essa conclusão confere com os dados já apresentados (Tabela 11), pois para essas instâncias a solução foi encontrada mais rapidamente com o algoritmo genético, e menos jornadas foram acrescentadas no problema mestre.

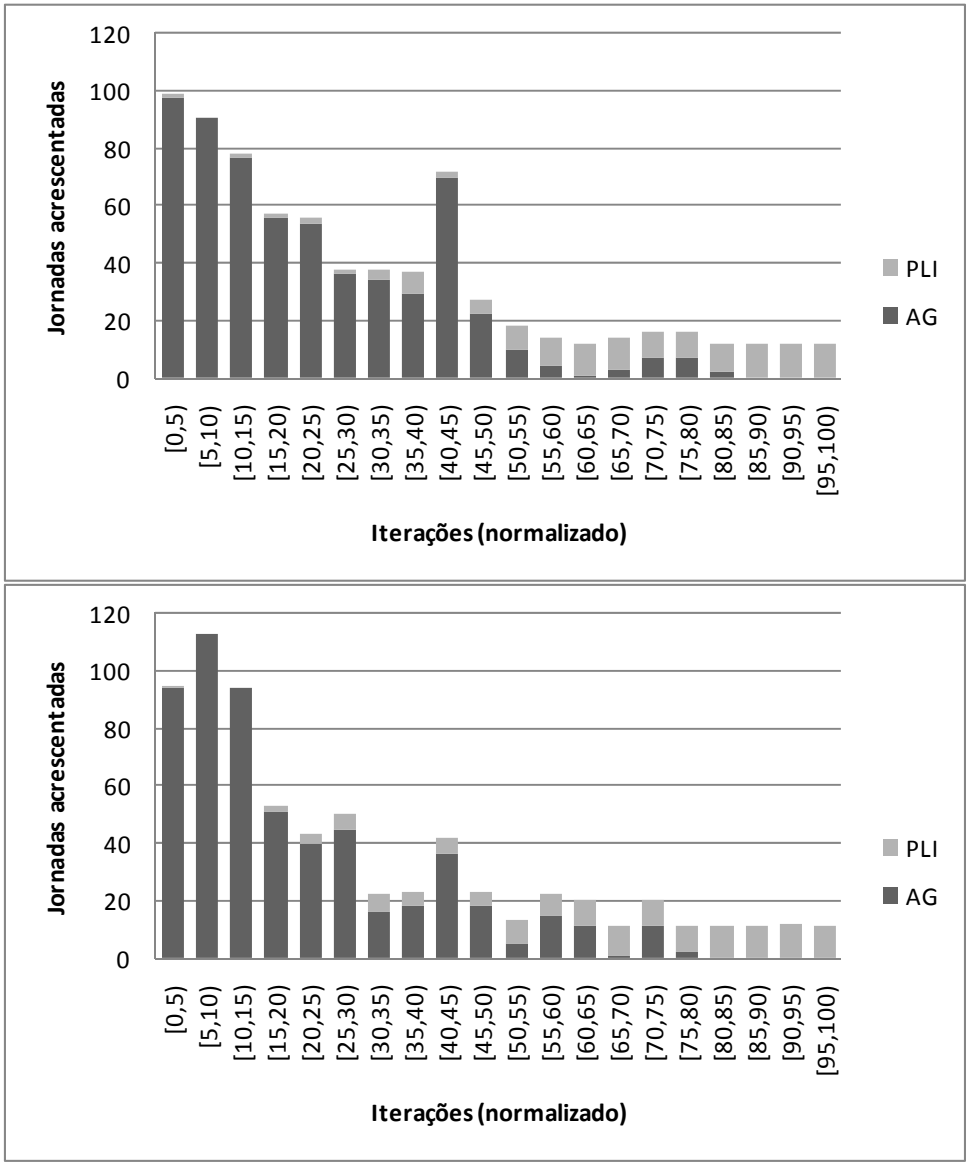

Figura 14: Geração de jornadas por algoritmo genético e método PLI nas instâncias L8208-78 (a) e L9206-82 (b)

As heurísticas não apenas atingem a solução mais rapidamente, mas também melhoram o valor da solução bem no início da geração de colunas, como mostram os gráficos da Figura 15 e Figura 16. Nesses gráficos são mostrados os valores da solução do problema mestre em cada iteração do algoritmo de geração de colunas, em função do tempo de execução. Os valores da solução são mostrados em escala logarítmica devido à diferença de ordem de grandeza entre as primeiras iterações, com jornadas artificiais, e as últimas iterações, próximas da solução ótima. O primeiro mostra a evolução do valor da solução nos primeiros 30 segundos de execução, para a instância L321, um dos poucos casos em que as meta-heurísticas não ajudam a acelerar a geração de colunas. Nessa instância, elas gastaram 9,6 minutos, enquanto o uso do método exato PLI gastou 8,5 minutos. Entretanto, mesmo que o método não tenha terminado em menos tempo, as heurísticas rapidamente melhoram o valor da solução. Todos os métodos partem de um conjunto inicial de jornadas artificiais, contendo uma jornada para cada tarefa, com um custo elevado. Já nos primeiros segundos o método GR melhorou bastante o valor da solução, enquanto o método RL gasta cerca de 5 segundos para atingir o mesmo nível. Por sua vez, o método PLI demora mais de 30 segundos para atingir valor semelhante da solução. O método GA tem comportamento semelhante ao GR, e não é mostrado no gráfico por questão de clareza. Esse é um resultado importante, pois mostra que, em não havendo tempo para executar o algoritmo até atingir a solução, a interrupção antecipada do algoritmo produz uma solução muito melhor quando é usada uma das meta-heurísticas na solução do subproblema (solução linear, pois no SPP pode não haver solução viável com apenas um subconjunto de colunas).

Os resultados detalhados da convergência da solução para a instância L9206-82 são mostrados graficamente na Figura 16. Esse representa o caso mais comum, onde as heurísticas não apenas geram jornadas que melhoram bastante o valor da solução já no início da geração de colunas como também ajudam a atingir a solução mais rapidamente. Enquanto com sua utilização a geração de colunas já encontrou a solução, a geração de jornadas pelo método PLI nem mesmo conseguiu gerar jornadas para cobrir todas as tarefas satisfatoriamente, apresentando um alto valor de solução, o que indica que as jornadas iniciais artificiais ainda estão presentes na solução do problema mestre.

Analisando-se os *log*s produzidos durante os experimentos, constata-se que nas últimas iterações nem GR, nem GA, nem RL conseguem gerar novas jornadas, logo, o método PLI é usado para gerar as últimas jornadas, a fim de atingir a solução ou provar sua otimalidade. Mas, nesse momento, o algoritmo já está bastante próximo da solução, então converge rapidamente e em menos tempo que se o método PLI fosse usado desde o início. Além disso, os preços duais já estão bem específicos, de tal forma que o método PLI rapidamente gera uma nova jornada, o que não acontece nas etapas iniciais da geração de colunas.

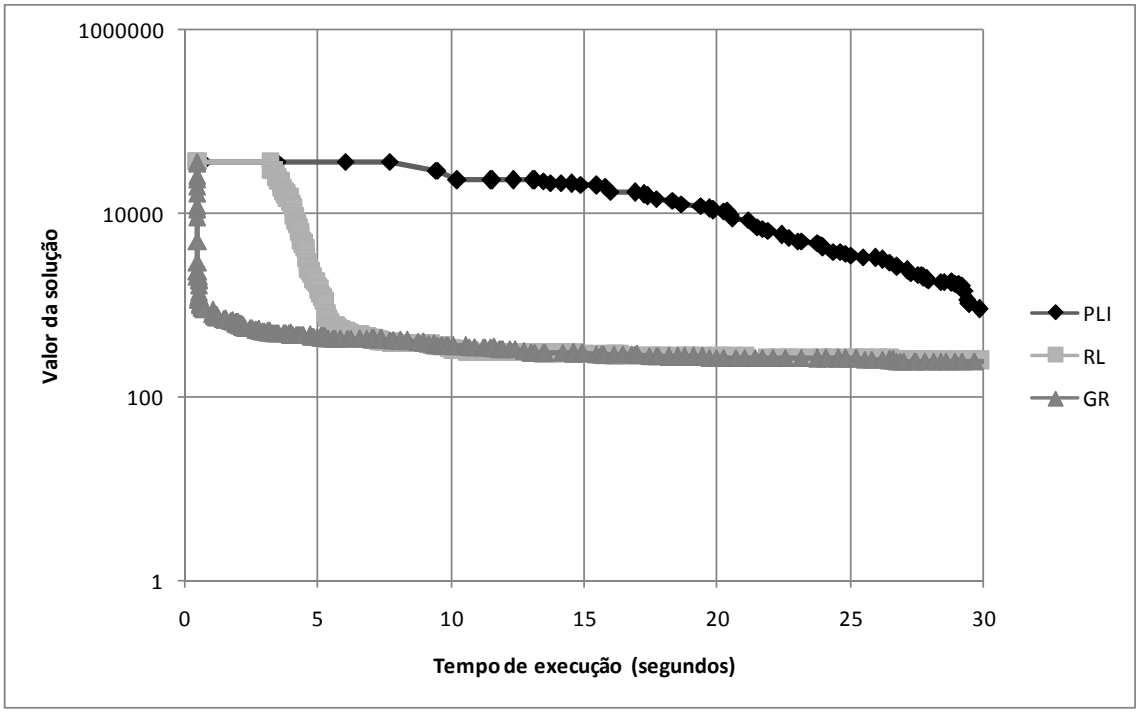

Figura 15: Convergência dos métodos PLI, RL e GR para a instância L321-54

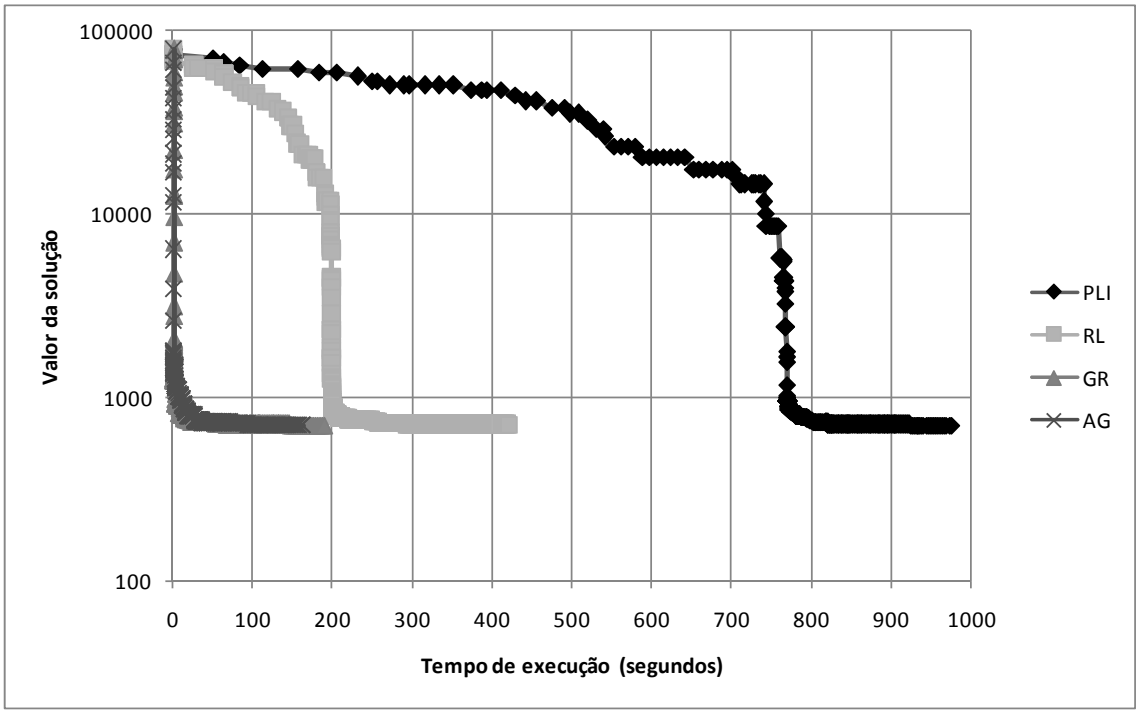

Figura 16: Convergência dos métodos PLI, RL e GR para a instância L9206-82

Para comprovar que o tempo gasto pelo método exato PLI realmente diminui nas últimas iterações, o gráfico da Figura 17 mostra o número de iterações do algoritmo de geração de colunas em relação ao tempo de execução, para algumas instâncias. Todos os tempos são normalizados em 100 para facilitar a plotagem em um único gráfico. A cada iteração o subproblema é resolvido uma vez, portanto o método PLI é usado uma vez, e uma

coluna é gerada. Como a inclinação da curva é menos acentuada nas primeiras iterações, estas são mais espaçadas no tempo, logo as iterações são mais lentas no início que no final. E como a solução do problema mestre é geralmente bem rápida se comparada com a solução do subproblema, e, além disso, quanto mais no início da execução menos colunas existem no problema mestre, pode-se dizer que a inclinação do gráfico reflete quase unicamente a execução do subproblema. Já que no final do algoritmo o método PLI resolve o subproblema de forma relativamente rápida, não há dificuldade em utilizá-lo quando as heurísticas falham na obtenção de novas jornadas, pois isso acontece justamente na geração das últimas jornadas.

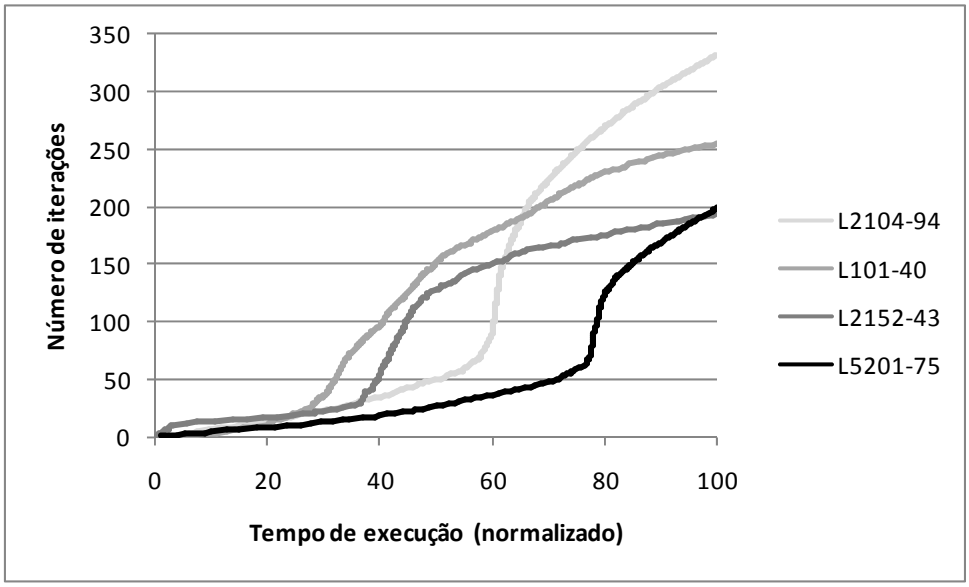

Figura 17: Número de iterações da geração de colunas em relação ao tempo de execução, para o método PLI

6.3. Branch-and-price

A Tabela 14 mostra resultados da aplicação do *branch-and-bound* para algumas instâncias da BHTrans cuja solução da relaxação linear não é inteira. O método $B\&B_{var}$ corresponde ao *branch-and-bound* com *branch* de variável, enquanto $B\&B_{f\text{-}on}$ é o *branch-and-bound* usando *follow-on*. A geração de colunas continua em todos os nós da árvore, logo é um método *branch-and-price*. Com *branch* de variável, apenas o método PLI é empregado nos nós da árvore, pois as heurísticas teriam que ser modificadas para impedir a geração de jornadas repetidas. Já com *branch follow-on*, as modificações são feitas diretamente no grafo, e qualquer heurística pode ser usada, sem modificação. Para cada tipo de *branch* a tabela mostra o número total de jornadas geradas, o tempo gasto até provar a otimalidade da solução (em minutos) e o número de nós gerados durante o processo, utilizando o método PLI para resolver o subproblema. Um tempo limite de 100 minutos foi utilizado, por isso o número de nós e de jornadas não são apresentados para algumas instâncias, e são marcados com '?'.

Em $B\&B_{var}$ a variável escolhida é a variável fracionária com valor mais próximo de 0,5. Primeiramente a jornada correspondente é fixada em 1, e depois fixada em 0. Em $B\&B_{f\text{-}on}$

também é escolhido o *follow-on* mais próximo de 0,5, que primeiramente é fixado em 1, e depois em 0. Os resultados mostram uma drástica redução no número de nós ao se usar essa estratégia, por exemplo, 13 nós contra 61 na instância L2104-94, 51 contra 329 na instância L4150-63, e 17 contra 199 na instância L8207-101. Isso se traduz conseqüentemente em redução do tempo de execução. Para duas das instâncias, L321-54 e L5201-75, a solução ótima não foi obtida em 100 minutos de execução utilizando *branch* de variável, mas foi obtida em respectivamente 46 e 20 minutos com a estratégia *follow-on*.

Tabela 14: Comportamento das estratégias de branch no branch-and-price

Instância	Z_{RL}	Z	$B\&B_{var}$			$B\&B_{f\text{-}on}$		
			Jorn.	CPU	Nós	Jorn.	CPU	nós
BH-L321-54	200,45	219,35	?	> 100	?	3686	46,1	45
BH-L2104-94	1084,87	1089,25	567	26,3	61	429	19,6	13
BH-L2152-43	187,20	202,30	437	2,8	23	479	3,1	25
BH-L4150-63	628,14	642,45	642	23,7	329	355	11,4	51
BH-L5201-75	787,16	793,50	?	> 100	?	407	20,4	33
BH-L8207-101	1193,68	1202,05	633	64,4	199	406	40,9	17
BH-L8208-78	548,91	555,95	446	11,2	19	388	11,4	5

Como a estratégia de *branch* do tipo *follow-on* permite a utilização das heurísticas em todos os nós da árvore de *branch-and-price*, elas foram utilizadas também para a obtenção da solução ótima inteira. A Tabela 15 resume os principais resultados. Embora o tempo de execução para obtenção da solução ótima pareça relativamente alto, todas as instâncias puderam ser resolvidas utilizando-se as heurísticas na solução do subproblema. Muitas dessas instâncias não são resolvidas diretamente pelos pacotes de programação linear. Para várias delas, a solução linear pode ser obtida gerando-se o conjunto completo de jornadas, porém quando há necessidade de execução de um *branch-and-bound* para obtenção da solução inteira, o número de variáveis é tão grande que as execuções são abortadas por falta de memória disponível. Mas com a geração de colunas, e especialmente com as heurísticas, todas elas podem ser resolvidas em tempo satisfatório.

Tabela 15: Comportamento das heurísticas durante o *branch-and-price*

Instância	PLI			RL			GR		
	Jorn.	CPU	nós	Jorn.	CPU	nós	Jorn.	CPU	nós
BH-L321-54	3686	46,1	45	3787	47,3	51	4006	43,8	41
BH-L2104-94	429	19,6	13	602	19,6	19	939	17,6	33
BH-L2152-43	479	3,1	25	581	3,2	29	862	3,4	33
BH-L4150-63	355	11,4	51	370	10,3	63	630	10,7	115
BH-L5201-75	407	20,4	33	516	14,9	35	800	10,1	33
BH-L8207-101	406	40,9	17	479	20,3	15	1064	21,2	93
BH-L8208-78	388	11,4	5	456	8,4	5	881	5,3	5

De uma forma geral, a utilização da heurística RL causa um pequeno aumento na quantidade de nós gerados, porém o tempo para garantir a solução ótima é ligeiramente menor. Já o uso da heurística Grasp aumenta bastante o número de nós gerados. Por exemplo, para a instância L2104-94 são utilizados 33 nós *branch-and-bound* contra 13 utilizando o

método PLI, porém com uma ligeira redução no tempo total de execução. O mesmo ocorre na instância L4150-63. Por outro lado, apesar de aumentar o número de nós, reduz o tempo de execução mais que a heurística RL.

Esses resultados levam a crer que o uso das heurísticas traz uma grande redução no tempo do primeiro nó, conforme resultados discutidos anteriormente, porém nos demais nós a redução não é tão significativa. Isso pode ser justificado pelo fato de já existirem mais jornadas no problema mestre, e assim os preços duais já estão devidamente ajustados permitindo o método PLI rapidamente encontrar as novas jornadas que são necessárias para terminar o processo de obtenção da solução ótima inteira. Mesmo assim, para uma das instâncias, L8207-101, a vantagem ainda continua clara. Em cerca de 20 minutos a solução ótima inteira é obtida utilizando as heurísticas para geração das jornadas, enquanto utilizando o método exato PLI nem mesmo a solução da relaxação linear foi obtida nesse tempo (conforme resultados da Tabela 11).

Nos resultados mostrados nas tabelas anteriores o número de jornadas e o tempo de execução correspondiam ao total do algoritmo, incluindo a obtenção da solução linear no primeiro nó. A Tabela 16 mostra os dados descontando-se os do primeiro nó, ou seja, mostra o número de novas jornadas geradas e o tempo gasto durante a busca da solução ótima inteira, sendo já conhecida a solução linear do primeiro nó (a diferença entre os dados da Tabela 15 e da Tabela 11). E a Tabela 17 mostra os mesmos dados, mas em termos percentuais, além do *gap* entre a solução da relaxação linear e a solução ótima inteira.

Tabela 16: Quantidade de novas jornadas e tempo gasto na obtenção da solução inteira, a partir da solução linear

Instância	PLI		RL		GR	
	Jorn.	CPU	Jorn.	CPU	Jorn.	CPU
BH-L2104-94	97	4,7	189	11,4	192	11,4
BH-L2152-43	285	2,3	337	2,7	316	2,8
BH-L4150-63	199	6,1	194	6,4	217	9,3
BH-L5201-75	208	10,5	252	8,3	189	7,5
BH-L8207-101	-	-	99	7,5	200	15,1
BH-L8208-78	68	3,3	78	2	90	2,2

Tabela 17: Percentual de novas jornadas e tempo gasto na obtenção da solução inteira, relativo à solução linear

Instância	*%Gap*	PLI		RL		GR	
		%Jorn.	%CPU	%Jorn.	%CPU	%Jorn.	%CPU
BH-L2104-94	0,4	23	24	31	58	20	65
BH-L2152-43	9,6	59	74	58	84	37	82
BH-L4150-63	2,2	56	54	52	62	34	87
BH-L5201-75	0,8	51	51	49	56	24	74
BH-L8207-101	0,7	-	-	21	37	19	71
BH-L8208-78	1,3	18	29	17	24	10	42

Esses dados comprovam o que foi dito anteriormente. Durante a expansão da árvore de *branch-and-bound* as heurísticas continuam levando vantagem, porém a vantagem em termos de tempo de execução já não é tão grande quanto era na obtenção da solução linear.

Os tempos gastos são similares, e por isso a porcentagem de tempo de execução nessa etapa é elevada para as heurísticas, já que elas são rápidas na solução linear. Porém, isso não significa que elas não levem vantagem nessa fase. Ela só não é tão perceptível quanto era obtenção da solução da relaxação linear.

6.4. Instâncias aéreas

A Tabela 18 resume os dados das 12 instâncias aéreas geradas. Elas contêm todos os vôos operados por uma companhia aérea brasileira entre os aeroportos descritos. Todos os dados foram obtidos no site web da companhia aérea. Assim, a quantidade de vôos e suas freqüências e horários são reais. Nem todos os vôos possuem freqüência diária, alguns não ocorrem aos sábados ou domingos. Como a quantidade e o horários dos vôos varia ligeiramente nos fins de semana, foram montadas 3 instâncias para cada conjunto de aeroportos: vôos operados durante os dias de semana (segunda-feira a sexta-feira); vôos operados aos sábados; e vôos operados aos domingos.

Tabela 18: Descrição e características das instâncias aéreas utilizadas

Instância	Aeroportos	Descrição	Dias	Vôos
Nacional1	4	Vôos entre os aeroportos de Confins (MG), Guarulhos e Congonhas (SP), e Galeão (RJ) CNF-GRU-CGH-GIG	seg-sex	70
			sab	53
			dom	53
Nacional2	5	Todos os vôos acima mais os vôos entre esses aeroportos e o aeroporto de Brasília (DF) CNF-GRU-CGH-GIG- BSB	seg-sex	141
			sab	100
			dom	93
Nacional3	6	Todos os vôos acima mais os vôos entre esses aeroportos e o aeroporto de Salvador (BA) CNF-GRU-CGH-GIG- BSB-SSA	seg-sex	185
			sab	141
			dom	134
Nacional4	7	Todos os vôos acima mais os vôos entre esses aeroportos e o aeroporto de Recife (PE) CNF-GRU-CGH-GIG- BSB-SSA-REC	seg-sex	223
			sab	178
			dom	171

Os primeiros testes consideram custos unitários para tempo ocioso e tempo adicional ao fim da jornada, $c_r = c_q = 1$, e nenhum custo de transição entre tarefas, $c_a = 0$. Na prática o custo c_r e a variável de decisão r podem ser retirados do modelo SUB_A, desde que o custo c_a de um arco $a = (v_i, v_j)$ seja o tempo ocioso entre as tarefas i e j, e assim, o custo por tempo ocioso em uma jornada é corretamente contabilizado. A vantagem é que dessa forma o custo não é descoberto apenas no final da geração da jornada, mas durante a sua construção, o que é altamente proveitoso para as heurísticas RL e GRASP, que trabalham de forma construtiva.

A Tabela 19 resume os resultados para as instâncias aéreas da Tabela 18. A coluna Z indica o valor da solução, e as demais colunas indicam, como nas tabelas de resultados anteriores, o número de jornadas geradas por cada método e o tempo gasto por eles na solução da relaxação linear do problema.

A maioria das soluções encontradas não possui resultados inteiros, mas todas apresentam o mesmo valor da solução ótima inteira. Os resultados da expansão da árvore de *branch-and-bound* são apresentados e discutidos mais à frente.

Tabela 19: Resultados para as instâncias aéreas

Instância		Z	Jornadas Geradas			CPU (minutos)		
			PLI	RL	GR	PLI	RL	GR
Nacional1	seg-sex	5143	113	183	255	0,75	0,10	0,41
	sab	5476	79	86	142	0,36	0,02	0,11
	dom	5460	77	101	137	0,25	0,02	0,05
Nacional2	seg-sex	13367	227	258	342	7,27	0,37	0,65
	sab	12208	170	154	221	2,39	0,22	0,16
	dom	10608	145	173	214	1,48	0,10	0,24
Nacional3	seg-sex	15204	290	320	483	18,95	0,80	3,37
	sab	14559	216	215	283	4,98	0,37	0,53
	dom	13577	211	215	317	4,48	0,29	1,21
Nacional4	seg-sex	16617	349	414	518	29,70	1,59	7,20
	sab	14892	278	309	436	9,65	0,57	1,46
	dom	15170	242	323	379	8,27	0,58	2,16

Nessas instâncias é nítida a melhoria de desempenho com a utilização das heurísticas, principalmente a heurística RL. O número de jornadas geradas não aumentou muito, mas a redução no tempo de execução foi bastante significativa. Enquanto usar apenas o método PLI para gerar as jornadas pode levar o método de geração de colunas a gastar cerca de 10, 20 e 30 minutos para resolver os problemas com os vôos de dia de semana, usando a heurística RL este tempo é menor que 1 minuto para todas as instâncias, exceto a maior delas, para qual são gastos 1,6 minutos. Embora seja ligeiramente melhor que a heurística RL em apenas 2 casos, o método GRASP se mostrou bastante eficiente, reduzindo bastante o tempo de execução, se comparado ao método exato.

Com os métodos RL e GR sendo usados em todos os nós da árvore, essa melhoria de desempenho continua semelhante durante a enumeração da árvore de *branch-and-bound*. A Tabela 20 mostra os resultados: número de nós de *branch-and-bound* explorados; quantidade de novas jornadas geradas durante a exploração dos nós da árvore (não contadas as jornadas geradas no 1º nó); e tempo gasto na expansão da árvore (mais uma vez, descontado o tempo gasto para resolver a relaxação linear, no 1º nó). Note que o tempo de solução continua sendo bem menor quando as heurísticas são utilizadas. Por exemplo, para a instância Nacional3 nos dias de semana já eram gastos quase 20 minutos no 1º nó com o método PLI e são gastos mais 39 na expansão da árvore. Já com o método GR gasta-se cerca de 5 minutos no total, 3 no primeiro nó e 2 na expansão da árvore. Para a instância Nacional4 essa diferença de desempenho é bastante acentuada, em mais de 100 minutos não foi terminada a expansão da árvore de *branch-and-bound* com o método PLI, mas em menos de 10 minutos os outros métodos já produziram colunas suficientes para a obtenção e comprovação da solução ótima.

Como mencionado, o valor da solução ótima já é atingido no 1º nó, mas para a maioria das instâncias essa solução é ainda fracionária. Após alguns nós de *branch-and-bound*, e tarefas consecutivas sendo fixadas pela estratégia de *follow-on*, uma solução inteira de mesmo valor é encontrada. Para 4 das instâncias, a solução obtida usando-se o método RL já é inteira no 1º nó. Elas podem ser identificadas na Tabela 20 observando-se que usaram apenas 1 nó de *branch-and-bound*. São especificamente as 3 instâncias com 4 aeroportos (Nacional1), e a instância com os vôos de Brasília aos domingos (Nacional2). O mesmo ocorre para 2 dessas instâncias com o método GR. Para esses métodos, uma solução inteira pode ser mais facilmente atingida porque eles geralmente acrescentam várias colunas a mais que o método PLI, então jornadas que seriam necessárias para compor a solução inteira podem já ter sido geradas durante a solução da relaxação linear. Isso explica porque geralmente menos nós são explorados, e também porque o número de novas jornadas geradas nessa exploração é menor.

Tabela 20: Resultados da expansão da árvore de branch-and-bound para as instâncias aéreas

Instância		Nós explorados			Novas Jornadas			CPU (min)		
		PLI	RL	GR	PLI	RL	GR	PLI	RL	GR
Nacional1	seg-sex	11	1	5	31	-	12	0,21	-	0,26
	sab	5	1	1	25	-	-	0,06	-	-
	dom	11	1	1	35	-	-	0,13	-	-
Nacional2	seg-sex	14	10	12	87	67	104	4,44	1,08	0,85
	sab	8	3	4	15	21	16	0,46	0,03	0,06
	dom	9	1	3	49	-	32	0,48	-	0,10
Nacional3	seg-sex	21	8	9	122	72	66	39,35	1,25	1,74
	sab	12	4	6	59	36	38	1,86	0,07	0,33
	dom	7	9	11	24	44	55	0,43	0,33	0,94
Nacional4	seg-sex	*>15*	14	14	*>100*	95	141	*>100,00*	6,34	9,29
	sab	19	8	15	85	48	124	7,89	0,93	3,73
	dom	16	8	8	84	50	57	5,00	0,72	0,84

Não são mostrados os resultados com Algoritmo Genético, porque, apesar de seu código genérico ser facilmente especializado para esse tipo de instância com penalidades para jornadas inviáveis, como foi para as instâncias da OR-Library e da BHTrans, nesse caso eles não trazem tão bons resultados, porque o número de vôos de uma jornada de trabalho é limitado e pequeno, sendo importante então um procedimento de viabilização de uma jornada, como retirada ou substituição de tarefas.

Além disso, ao contrário das instâncias da OR-Library e da BHTrans, aqui o GRASP não trouxe melhores resultados que a heurística RL. Trouxeram uma melhoria significativa em relação ao método exato PLI, porém a heurística RL apresentou melhores resultados para quase todas as instâncias. Analisando a geração de colunas utilizando o método GR, foi verificado que esse método gera várias colunas durante o algoritmo, porém quase todas bem no início do algoritmo. Logo depois, ele praticamente falha em todas as tentativas, sendo o método PLI responsável por gerar quase todas as colunas mesmo ainda antes da metade das iterações realizadas pela geração de colunas. Por outro lado, o método RL gera bastante

colunas, sendo as das primeiras iterações mais demoradas para serem geradas, e de valores relativamente ruins.

Como o método de geração de colunas aqui apresentado pode usar qualquer uma dessas heurísticas em cada iteração, ele pode também utilizar mais de uma numa mesma execução, em conjunto ou alternadamente. Já que para essas instâncias o GRASP se sai melhor apenas nas primeiras iterações, e a heurística RL não é tão boa nas primeiras iterações, mas gera muitas boas jornadas depois de certo número de iterações, um método híbrido GR+RL foi aplicado para a solução dessas instâncias: inicialmente apenas o método GR é utilizado; quando ele falha na geração de novas jornadas em alguma iteração do método de geração de colunas, ao invés de se usar diretamente o método PLI para resolver o subproblema nessa iteração, utiliza-se o método RL; se ele também falhar, o método PLI é utilizado. Depois que o método GR falha várias vezes, ele nem é mais utilizado, passa-se a usar sempre o método RL como primeira alternativa, e se ele não gerar nova coluna, o método PLI é encarregado de gerar alguma (ou provar que a solução ótima já foi encontrada).

Note que esse método híbrido GR+RL une as vantagens das duas heurísticas, e procura descartar as desvantagens de cada uma. O método GR é utilizado quando ele é mais eficiente, e o método RL quando ele começa a ser mais eficiente. A Tabela 21 compara os resultados desse método híbrido em relação aos resultados já apresentados na Tabela 19 com cada uma das heurísticas trabalhando isoladamente. Em todos os testes, inicialmente o método GR é utilizado, e sempre que falha o método RL é acionado, e quando esse também falha, o método PLI gera de forma exata a coluna de menor custo. Isso é feita em cada iteração, ou seja, mesmo que uma das heurísticas não funcione bem em alguma iteração do método de geração de colunas, ela é acionada novamente na iteração seguinte, agora com os novos preços duais calculados pelo problema mestre, portanto elas têm chance de mais uma vez tentar gerar alguma jornada.

Tabela 21: Comparação dos resultados do método híbrido GR+RL com os métodos RL e GR isoladamente

Instância		Jornadas Geradas			CPU (minutos)		
		RL	GR	GR+RL	RL	GR	GR+RL
Nacional1	seg-sex	183	255	261	0,10	0,41	0,08
	sab	86	142	159	0,02	0,11	0,02
	dom	101	137	137	0,02	0,05	0,01
Nacional2	seg-sex	258	342	341	0,37	0,65	0,29
	sab	154	221	220	0,22	0,16	0,08
	dom	173	214	239	0,10	0,24	0,09
Nacional3	seg-sex	320	483	466	0,80	3,37	0,78
	sab	215	283	295	0,37	0,53	0,16
	dom	215	317	321	0,29	1,21	0,21
Nacional4	seg-sex	414	518	533	1,59	7,20	0,53
	sab	309	436	426	0,57	1,46	0,54
	dom	323	379	373	0,58	2,16	0,33

Apesar de muitas vezes gerar mais jornadas de trabalho que os métodos RL e GR, esse método híbrido GR+RL é melhor em termos de tempo de execução. A geração de colunas utilizando esse método resolveu todas as instâncias de forma mais rápida que quando apenas um dos métodos era usado, seja RL ou GR. A Figura 18 mostra graficamente essa melhoria de desempenho obtida por esse método híbrido.

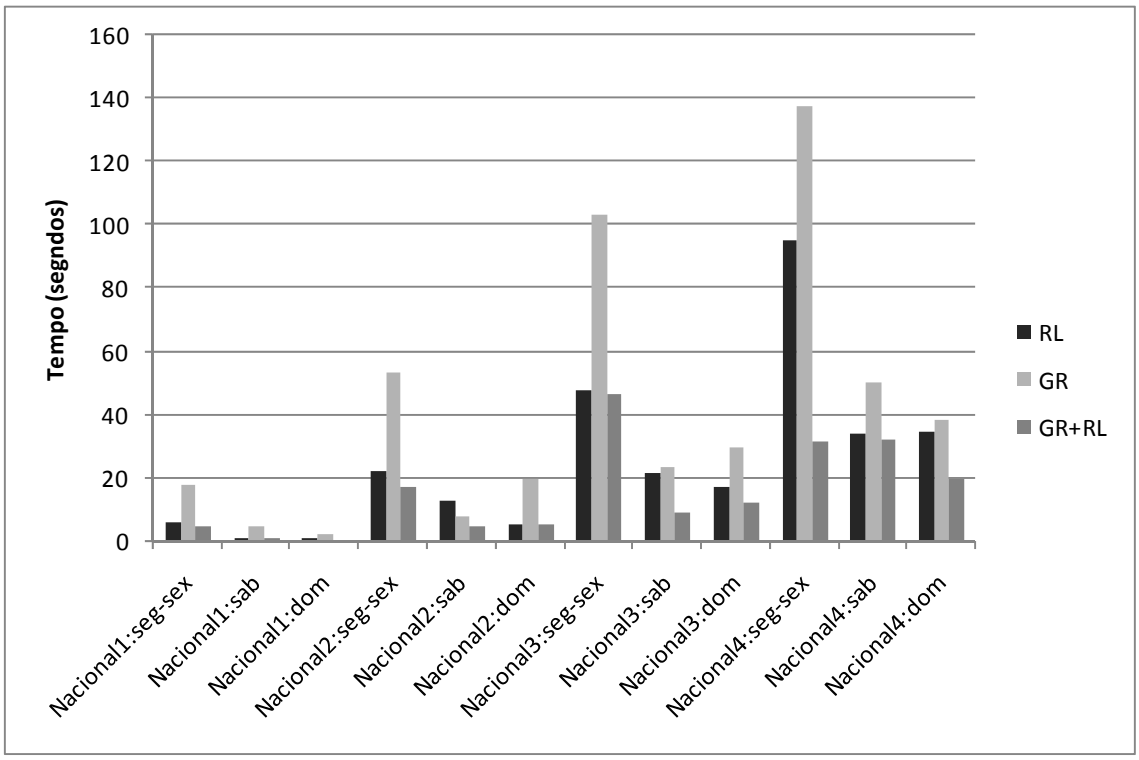

Figura 18: Tempo de execução do método híbrido GR+LR comparado aos métodos RL e GR usados isoladamente

Como o método GR gera cada vez menos jornadas de custo reduzido negativo com o passar das iterações (como já comprovado pelos gráficos da Figura 13 para instâncias da BHTrans), então, na implementação feita o método híbrido GR+RL passa a invocar diretamente o método RL depois de 20 falhas do método GR. Esse limite de tolerância foi obtido após alguns experimentos e todos os resultados apresentados utilizaram esse valor. Vale ressaltar que nem é necessário estabelecer tal limite, o método GR poderia ser utilizado como tentativa de gerar nova coluna até o fim da geração de colunas. Mas como a probabilidade de gerar novas colunas diminui à medida que o algoritmo avança então uma boa estratégia foi nem mesmo perder tempo tentando gerar nova jornada com esse método nas iterações finais.

O problema foi resolvido também com custos diferenciados para tempo ocioso e tempo adicional ao fim da jornada, no caso $c_r = 0{,}5$ e $c_q = 1$. Assim, o templo ocioso entre os vôos tem custo menor que o tempo adicional ocioso no fim da jornada, o que tende a produzir jornadas com os vôos mais distantes um do outro dentro da mesma jornada. Isso é interessante, uma vez que a tripulação cumpre sua tarefa de forma mais folgada, e eventuais atrasos em algum vôo não comprometem a continuação de seu trabalho nos vôos seguintes, pois geralmente há tempo ocioso suficiente até o início do próximo vôo, evitando a reprogramação das tarefas restantes do dia.

A Tabela 22 mostra os resultados para esses custos antes da expansão da árvore de *branch-and-bound*. Apenas uma dessas instâncias não teve solução inteira no primeiro nó, e os resultados são mostrados na Tabela 23. Nota-se que, como já esperado, com esses custos o problema se torna mais difícil de ser resolvido. As instâncias menores, como Nacional1, e Nacional2 aos sábados e domingos, continuam sendo resolvidos em sua maioria em menos de 2 minutos, por qualquer dos métodos de solução do subproblema, sendo o PLI o mais demorado. Nas demais instâncias o tempo é geralmente superior a 10 minutos para todos os métodos, mas as heurísticas continuam ajudando a solução, diminuindo o tempo de execução. Por exemplo, PLI demora quase meia hora para a instância Nacional2 nos dias de semana, e o método RL leva 18 minutos, uma redução de quase 40% do tempo de execução. Para Nacional4 aos sábados e domingos, tanto RL quanto GR+RL reduzem cerca de 30% do tempo de execução. Para a maior das instâncias, usando-se o método PLI a solução não foi obtida em 100 minutos de execução, enquanto que usando-se os demais métodos a solução pôde ser obtida dentro desse tempo limite.

Tabela 22: Resultados com custos diferenciados, c_r = 0,5 e c_q = 1,0

Instância		Z	Jornadas Geradas				CPU (minutos)			
			PLI	RL	GR	GR+RL	PLI	RL	GR	GR+RL
Nacional1	seg-sex	3432,5	249	279	416	406	2,6	1,3	2,0	1,0
	sab	3708,5	129	163	228	246	0,4	0,3	0,1	0,3
	dom	3781,5	127	167	247	251	0,6	0,4	0,3	0,3
Nacional2	seg-sex	7693,5	517	582	768	746	27,0	18,1	20,7	20,0
	sab	7093	297	325	493	492	4,1	2,2	3,2	2,5
	dom	6600,25	240	289	416	419	2,4	1,5	1,9	1,4
Nacional3	seg-sex	8855	669	695	1030	884	73,7	54,9	62,8	56,6
	sab	8353	456	501	633	598	14,0	8,9	9,6	9,7
	dom	8073,5	407	469	594	536	10,8	7,0	8,3	6,3
Nacional4	seg-sex	9677,5	*> 700*	839	1128	1042	*> 100*	97,8	98,2	96,1
	sab	8651	562	600	875	820	34,8	23,3	29,2	23,7
	dom	8927,5	486	501	742	687	22,8	15,6	19,1	16,7

Mais uma vez, a combinação das heurísticas no método híbrido GR+RL trouxe melhores resultados em quase todas as instâncias, quando comparada ao método GR. Com exceção da instância Nacional3 aos sábados, o tempo de execução utilizando esse método é menor que se utilizado o método GR. Porém em várias instâncias não houve redução em relação ao método RL, o que indica que o método GR não obteve bons resultados para esse tipo de problema, nem trabalhando isoladamente nem em conjunto com a heurística RL. Um dos motivos é que durante a construção da jornada no método GR, as tarefas são escolhidas considerando-se o aumento de custo que elas representam se adicionadas à jornada sendo construída. A dificuldade para o GRASP, nesse caso, é que uma tarefa com início mais tarde representa um aumento maior de custo em relação a uma tarefa com início mais próximo ao da última tarefa adicionada à jornada, já que implica em maior tempo ocioso. Porém, com custo menor para tempo ocioso entre vôos em relação ao custo do tempo adicional no final da

jornada, a inclusão dessa tarefa com início mais tarde, embora aumente em maior proporção o custo da jornada sendo construída, isso se dá apenas localmente, pois ao final, pode implicar em menor tempo adicional ao fim da jornada, produzindo uma jornada com menor custo.

Esse inconveniente de custos para a heurística Grasp pode ser atenuado considerando-se de forma diferente os custos na construção da jornada, por exemplo, levando-se em conta apenas o custo reduzido da tarefa a ser incluída na jornada, ou dando prioridade às tarefas mais distantes. Essas modificações não foram feitas apenas porque se desejava fazer a comparação dos diferentes métodos em vários tipos de instâncias considerando-se as mesmas características.

Como mencionado, com esses valores de custos em apenas uma das instâncias a relaxação linear do problema mestre teve uma solução fracionária. A Tabela 23 mostra os resultados completos da árvore de *branch-and-bound* para essa instância, Nacional2 com vôos operados aos domingos. Os dados foram obtidos utilizando-se a estratégia *follow-on*. Em todas as demais a solução foi inteira, então não foi necessário expandir a árvore de *branch-and-bound*, e assim os dados já comentados da Tabela 22 representam os valores para obtenção da solução ótima.

Tabela 23: Resultados da árvore de branch-and-bound para a instância Nacional2 aos domingos

	PLI	RL	GR	GR+RL
Jornadas nó raiz	240	289	416	419
Tempo nó raiz (min)	2,4	1,5	1,9	1,4
Z_{RL}	6600,25			
Nós de B&B	13	9	23	13
Total de Jornadas	320	349	509	480
Tempo total (min)	4,1	2,7	4,1	2,7
Z	6617,5			

Nota-se que as heurísticas RL e GR+RL são mais rápidas tanto no nó raiz quanto nos demais nós, reduzindo em mais de 30% o tempo de execução. Além disso, elas acrescentam menos jornadas durante a expansão da árvore, cerca de 60 novas jornadas, contra 80 do método PLI. Mais uma vez, nota-se que pelo acréscimo de mais jornadas durante a obtenção da solução linear, algumas que seriam geradas pelo PLI apenas para obtenção da solução ótima inteira já foram geradas pelas heurísticas, diminuindo a necessidade de novas jornadas, reduzindo o tempo e às vezes o número de nós. Nesse caso os resultados não foram tão favoráveis ao método GR, em parte pela forma como os custos são calculados, como falado anteriormente.

Todas as heurísticas foram utilizadas da mesma forma, com os mesmos parâmetros, para todas as instâncias aqui apresentadas. Se necessário elas podem ser calibradas para atender detalhes específicos de cada instância.

6.5. Outras aplicações

A geração de colunas com o método GRASP também foi aplicada ao problema de alocação de tripulação aérea, com outras instâncias criadas a partir de dados pesquisados no site de uma companhia aérea, particularmente as rotas entre algumas cidades da região sudeste (Bento, Santos, & Mateus, 2008). As restrições de viabilidade de uma jornada são as detalhadas na regulamentação profissional do sindicato dos aeronautas (Regulamentação Profissional, 1984). Nesse trabalho foi considerada também a possibilidade de *deadhead*, ou seja, uma tripulação viajar como passageiro, quando a próxima tarefa a ser cumprida se inicia em um aeroporto diferente do final da última tarefa. Alem do tempo ocioso, foi considerada na função objetivo quilometragem voada em cada jornada de trabalho. A utilização do GRASP como forma de solução do subproblema também reduziu o tempo de execução, principalmente para instâncias maiores. Por exemplo, em uma instância com 3 aeroportos e 226 tarefas, a solução foi obtida em 76 segundos contra mais de 500 sem a utilização da heurística. Além disso, para uma instância com 4 aeroportos e 538 tarefas, a solução da relaxação linear do problema foi obtida com 1 hora de execução, contra 2 horas sem o uso da heurística. Nesse caso, a solução ótima inteira não pôde ser obtida em 2 horas e 30 minutos de execução, porém uma solução inteira viável obtida com as jornadas geradas pelo Grasp tem melhor valor que quando são usadas apenas as jornadas geradas pelo método exato de programação inteira.

Para as instâncias mais difíceis da BHTrans, também foi feita uma tentativa de gerar soluções iniciais mais elaboradas, contendo jornadas interessantes e não apenas jornadas artificiais com apenas uma tarefa. Tais soluções foram geradas com um método de Pesquisa Tabu que obtém boas soluções aproximadas em curto tempo de execução (Gonçalves, Santos, & Silva, 2008). Porém, a solução ótima usando a geração de colunas, mesmo utilizando essa boa base inicial, é obtida em tempo de execução semelhante aos reportados na seção 6.2, com soluções inicias triviais. De fato, mesmo criando uma base inicial contendo as jornadas que fazem parte da solução ótima, o método de geração de colunas continua gerando várias novas jornadas, para obter uma solução linear de melhor valor. Isso se dá mesmo nos casos onde a solução da relaxação linear é a própria solução inteira, por causa da grande degeneração da base, que ocorre em problemas de alocação de tripulação. Várias colunas são geradas sem modificar o valor da solução. Por isso, os resultados apresentados anteriormente referem-se sempre à geração de colunas a partir de uma base trivial.

Além disso, as instâncias da BHTrans também foram resolvidas considerando-se uma ou mais de uma dupla pegada sem custo adicional. Entretanto, as soluções de todas as instâncias já continham dupla pegada, mesmo quando todas eram penalizadas. Portanto, a não cobrança de uma ou algumas duplas pegadas não altera as jornadas da solução ótima, pelo menos para essas instâncias e com os custos operacionais considerados.

7. Conclusões e Trabalhos Futuros

O trabalho descreve um algoritmo de geração de colunas para a solução de problemas de alocação de tripulação com uma etapa adicional de solução do subproblema por métodos heurísticos. Apesar do uso de heurísticas, o método como um todo permanece exato, pois as heurísticas são embutidas no processo de geração de colunas apenas para acelerar a resolução do subproblema, que é sempre resolvido de forma exata quando as heurísticas falham em resolvê-lo satisfatoriamente. Tanto o problema mestre quanto o subproblema são problemas clássicos, particionamento de conjuntos, e caminho mínimo com restrições de recursos, para os quais vários métodos, exatos e heurísticos, foram propostos na literatura. Nenhum deles tem solução polinomial, porém ambos são resolvidos de forma eficiente dentro do algoritmo de geração de colunas considerado. O problema mestre é resolvido facilmente por pacotes de programação linear, porque dentro do contexto de geração de colunas não contém muitas variáveis de decisão. E mesmo quando o número de variáveis aumenta, as soluções podem ser facilmente obtidas a partir da base da solução obtida anteriormente, pois o problema cresce aos poucos. O subproblema, principalmente no início do algoritmo, tem solução exata um pouco demorada, e nesse ponto as heurísticas obtêm soluções rápidas e tão interessantes quando a solução ótima exata. Desta forma, todos são resolvidos rapidamente.

Apesar de ter sido aplicado em apenas três tipos de instâncias, a técnica é geral e pode ser aplicada praticamente em qualquer problema de alocação de tripulações. As heurísticas e meta-heurísticas utilizadas não foram refinadas e calibradas especialmente para as instâncias consideradas, e nem contam com detalhes tão específicos de cada uma. Os mesmos parâmetros foram utilizados na solução de todas as instâncias, o que mostra que elas são facilmente aplicáveis em outros contextos.

Para instâncias do problema de alocação de tripulações com poucas tarefas, ou cujas restrições de viabilidade de uma jornada de trabalho impeçam que o número de possíveis jornadas esteja na casa dos milhões, é possível simplesmente gerar todas as jornadas e selecionar o melhor subconjunto. Mas mesmo nesses casos, a técnica de geração de colunas se mostra bastante atrativa, obtendo soluções mais rapidamente, e sem utilizar tantos recursos de memória. Quando a geração de novas jornadas é complexa e demorada, o uso de heurísticas contribui de forma significativa para a redução do tempo de execução do método até encontrar a solução ótima e assim, como mostrado na análise de resultados, possibilita também encontrar uma solução satisfatória em um tempo muito curto.

Embora fosse possível incluir algumas técnicas para acelerar a convergência da solução, como: retirar colunas redundantes ou sem potencial de contribuição para uma melhoria na solução; acrescentar cortes no subproblema para acelerar sua solução; calibrar os parâmetros para cada instância e introduzir busca local nas heurísticas; entre outras, o enfoque do trabalho não é resolver as instâncias aqui tratadas, mas estabelecer um algoritmo híbrido com heurísticas e métodos exatos, que seja mais rápido sem perder a garantia de otimalidade, e geral para ser aplicado em outros contextos. Outro contexto significa não apenas outros problemas de alocação de tripulações, mas outros problemas semelhantes, como por exemplo, o problema de roteamento e veículos, que também pode ser modelado

como um problema de particionamento de conjuntos com subproblema de caminho mínimo com restrições de recursos. A aplicação nesse contexto, e a decisão automática de qual das heurísticas deve ser utilizada em cada iteração durante a execução do método, constituem atividades em andamento e que provavelmente rendam novos resultados interessantes.

Outros trabalhos futuros envolvem a aplicação de técnicas para auxiliar a solução dos modelos de programação inteira, mais especificamente: a técnica de *local branching* (Fischetti & Lodi, 2003), uso de cortes para fortalecer a relaxação linear do problema mestre SPP, e tratar o problema de estabilização da geração de colunas. Como geralmente acontece, a implementação da geração de colunas aqui descrita também sobre do problema de estabilização: os valores dos preços duais oscilam muito. Em algumas iterações algumas tarefas possuem preço dual exageradamente atrativo, e jornadas com estas tarefas são geradas pelo subproblema, porém tais jornadas geralmente não fazem parte da solução ótima final. Tratar a estabilização é interessante nesse caso para auxiliar reduzir a geração de tais colunas desnecessárias. Esse problema pode ser tratado de algumas formas, por exemplo, limitando-se cuidadosamente os valores das variáveis duais (Du Merle, Villeneuve, Desrosiers, & Hansen, 1999). Existem vários tipos de cortes que podem ser usados no SPP (Jepsen, Petersen, Spoorendonk, & Pisinger, 2008) (Fischetti, Lodi, Martello, & Toth, 2001) (Letchford, Eglese, & Lysgaard, 2002) (Baldacci, Christofides, & Mingozzi, 2008). Os cortes do tipo clique, por exemplo, definem facetas no politopo do problema de partição de conjuntos (Nemhauser & Wolsey, 1988) e o problema de separação correspondente é de fácil aplicação no problema aqui tratado. Entretanto, como todos esses cortes são representados por restrições no problema mestre, eles possuem um preço dual associado que deve também ser considerado no subproblema, inclusive para impedir a geração de colunas repetidas, criando assim um método de *branch-and-cut-and-price* se usados em todos os nós da árvore de *branch-and-bound*. Em várias das instâncias aqui tratadas a relaxação linear já era forte o suficiente para produzir soluções inteiras, mas os cortes são úteis na solução das demais instâncias.

Todas essas idéias de trabalhos futuros podem ser incorporadas no método de geração de colunas aqui proposto, e pode-se continuar utilizando as heurísticas para acelerar a convergência do método. Inclusive outras heurísticas podem ser facilmente incorporadas, como por exemplo, a programação dinâmica por rotulação de vértices que não se mostrou viável pela quantidade de recursos necessários na modelagem do problema de caminho mínimo por restrições em algumas instâncias, ser resolvida de forma aproximada.

Referências

Balas, E., & Carrera, M. C. (1996). A Dynamic Subgradient-Based Branch and Bound Procedure for Set Covering. *Operations Research , 44*, pp. 875-890.

Balas, E., & Ho, A. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. *Mathematical Programming Study , 12*, pp. 37-60.

Balas, E., & Ng, S. M. (1989). On the set covering polytope: I. all the facets with coefficients in {0,1,2}. *Mathematical Programming , 43*, pp. 57-69.

Balas, E., & Ng, S. M. (1989). On the set covering polytope: II. Lifting the facets with coefficients in {0,1,2}. *Mathematical Programming , 45*, pp. 1-20.

Balas, E., Ceria, S., & Cornuéjols, G. (1996). Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science , 42*, pp. 1229-1246.

Baldacci, R., Christofides, N., & Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Math. Program. , 115*, pp. 351-385.

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., & Vance, P. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research , 46*, pp. 316-329.

Beasley, J. E. (1990). A Lagrangean heuristic for set-covering problems. *Naval Research Logistics , 37*, pp. 151-164.

Beasley, J. E. (1987). An algorithm for set covering problems. *European Journal of Operational Research , 31*, pp. 85-93.

Beasley, J. E. (s.d.). *OR-Library*. Acesso em junho de 2007, disponível em http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Beasley, J. E., & Cao, B. (1996). A tree search algorithm for the crew scheduling problem. *European Journal of Operational Research , 94*, pp. 517-526.

Beasley, J. E., & Chu, P. C. (1996). A Genetic Algorithm for the Set Covering Problem. *European Journal of Operational Research , 94*, pp. 392-404.

Beasley, J. E., & Jornstein, K. (1992). Enhancing an algorithm for set covering problems. *European Journal of Operational Research, , 58*, pp. 293-300.

Bento, L. F., Santos, A. G., & Mateus, G. R. (2008). Airline Crew Scheduling: A hybrid approach using metaheuristics to improve an exact collumn generation algorithm. *International Symposium on Combinatorial Optimization*, (pp. 74-75). Coventry, Inglaterra.

Bento, L. F., Santos, A. G., & Mateus, G. R. (2008). Airline Crew Scheduling: A hybrid approach using metaheuristics to improve an exact collumn generation algorithm. *International Symposium on Combinatorial Optimization.* Coventry, Inglaterra.

Cabral, L. A., Souza, M. J., Maculan, N., & Pontes, R. C. (2000). An heuristic approach for large scale crew scheduling problems at Rio-Sul airlines. *40th International Symposium of the AGIFORS .* Istambul, Turquia.

Caprara, A., Fischetti, M., & Toth, P. (1999). A Heuristic Method for the Set Covering Problem. *Operations Research , 47*, pp. 730-743.

Caprara, A., Toth, P., Vigo, D., & Fischetti, M. (1998). Modeling and Solving the Crew Rostering Problem. *Operations Research , 46(6)*, pp. 820-830.

Carvalho, M. A., Mateus, G. R., & Santos, A. G. (2005). Seleção de Colunas no Problema de Escalonamento de Tripulações utilizando Algoritmo Genético. *XXXVII Simpósio Brasileiro de Pesquisa Operacional, PIC – Prêmio de Iniciação Científica*, (pp. 2512-2519). Gramado, RS.

Cohn, A. M., & Barnhart, C. (2003). Improving Crew Scheduling by Incorporating Key Maintenance Routing Decisions. *Operations Research , 51(3)*, pp. 387-396.

Cornuéjols, G., & Sassano, A. (1989). On the 0,1 facet of the set covering problems. *Mathematical Programming , 43*, pp. 45-55.

Dash Optimization Ltd. (2006). XPRESS-BCL Reference Manual.

Desaulniers, G., Desrosiers, J., & Solomon, M. M. (2005). *Column Generation (Gerad 25th anniversary series).* Springer.

Desrochers, M., & Soumis, F. (1989). A column generation approach to the urban transit crew scheduling problem. *Transportation Science , 23*, pp. 1-13.

Du Merle, O., Villeneuve, D., Desrosiers, J., & Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics , 194*, pp. 229-237.

Ehrgott, M., & Ryan, D. M. (2006). Constructing robust crew schedules with bicriteria optimization. *21st European Conference on Operational Research,.* Reykjavìk, Islândia.

Feo, T. A., & Resende, M. G. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization , 6*, pp. 109-133.

Fischetti, M., & Lodi, A. (2003). Local branching. *Mathematical Programming , 98*, pp. 23-47.

Fischetti, M., Lodi, A., Martello, S., & Toth, P. (2001). A polyhedral approach to simplified crew scheduling and vehicle scheduling problems. *Management Science , 47*, pp. 833-850.

Fisher, M. L., & Kedia, P. (1990). Optimal solution of set covering/partitioning problems using dual heuristics. *Management Science , 36*, pp. 674-688.

Fores, S., Proll, L., & Wren, A. (1996). A column generation approach to bus driver scheduling. In: M. H. Bell, *Transportation Networks: Recent Methodological Advances* (pp. 195-208). Pergamon.

Gamache, M., Soumis, F., & Marquis, G. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research , 47*, pp. 247-262.

Garey, M., & Johnson, D. (1979). *Computers and Intractability: A guide to the theory of NP-completeness.* San Francisco: W.H. Freeman.

Gonçalves, T. L., Santos, A. G., & Silva, J. M. (2008). Estratégia Paralela para Metaheurísticas Busca Tabu Aplicada ao Problema de Programação de Tripulações. *XIV CLAIO - Congreso Latino Ibero Americano de Investigación de Operaciones.* Cartagena de Índias, Colômbia.

Gopalakrishnan, B., & Johnson, E. L. (2005). Airline Crew Scheduling: State-of-the-Art. *Annals of Operations Research , 140*, pp. 305-337.

ILOG S.A. (2005). ILOG CPLEX 10.0 User's Manual and Reference Manual.

Jaszkiewicz, A. A. (2004). A Comparative Study of Multiple-Objective Metaheuristics on the Bi-Objective Set Covering Problem and the Pareto Memetic Algorithm. *Annals of Operations Research , 131*, pp. 135-138.

Jepsen, M., Petersen, B., Spoorendonk, S., & Pisinger, D. (2008). Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows. *Operations Research , 56(2)*, pp. 497-511.

Klabjan, D., Johnson, E. L., Nemhauser, G. L., Gelman, E., & Ramaswamy, S. (2001). Solving large airline crew scheduling problems: Random pairing generation and strong branching. *Comput. Optim. Appl , 20*, pp. 73-91.

Kohl, N. (1995). Exact methods for time constrained routing and related scheduling problems. *Tese de Doutorado* . Denmark: Department of Mathematical Modeling, Technical University of Denmark.

Kohl, N., & Karisch, S. E. (2004). Airline Crew Rostering: Problem Types, Modeling and Optimization. *Annals of Operations Research , 127*, pp. 223-257.

Kwan, R. S., Wren, A., & Kwan, A. S. (2000). Hybrid Genetic Algorithms for Scheduling Bus and Train Drivers. *IEEE Congress on Evolutionary Computation.*

Letchford, A. N., Eglese, R. W., & Lysgaard, J. (2002). Multistars, partial multistars and the capacitated vehicle routing problem. *Math. Program. , 94*, pp. 21-40.

Lourenço, H. R., Paixão, J. P., & Portugal, R. (2001). Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation science , 35*, pp. 331-343.

Lourenço, H. R., Paixão, J. P., & Portugal, R. (2001). Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation science , 35*, p. 331{343.

Makri, A., & Klabjan, D. (2004). A New Pricing Scheme for Airline Crew Scheduling. *INFORMS Journal on Computing , 16*, pp. 56-67.

Marinho, E. H. (2005). Heurísticas Busca Tabu para o Problema de Programação de Tripulações de Ônibus Urbano. *Dissertação de Mestrado* . Niterói, RJ, Brasil: Instituto de Computação - Universidade Federal Fluminense.

Marinho, E. H. (2005). Heurísticas Busca Tabu para o Problema de Programação de Tripulações de Ônibus Urbano. *Dissertação de Mestrado* . Niterói, RJ, Brasil: Universidade Federal Fluminense.

Marsten, R. E., & Shepardson, F. (1981). Exact Solutions of Crew Scheduling Problems Using the Set Partitioning Problem: Recent Successful Applications. *Networks , 11*, pp. 165-177.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs.* New York: Springer.

Mingozzi, A., Boschetti, M. A., Ricciardelli, S., & Bianco, L. (1999). A Set Partitioning Approach to the Crew Scheduling Problem. *Operations Research , 47(6)*, pp. 873-888.

Nemhauser, G. L., & Wolsey, L. A. (1988). *Integer and Combinatorial Optimization.* New York: John Wiley & Sons.

Regulamentação Profissional. (1984). Acesso em 2007, disponível em Sindicato Nacional dos Aeronautas: http://www.aeronautas.org.br/sejur/lei7183_prn.html

Revelle, C., Marks, D., & Liebman, J. C. (1970). An Analysis of Private and Public Sector Location Model. *Management Science , 16*, pp. 692-707.

Ryan, D. M., & Foster, B. A. (1981). An integer programming approach to scheduling. In: A. Wren, *Computer Scheduling of Public Transport* (pp. 269-280). Amsterdam: North-Holland Publishing Company.

Santos, A. G., & Mateus, G. R. (2007). Crew Scheduling Urban Problem: an Exact Column Generation Approach Improved by a Genetic Algorithm. *IEEE Congress on Evolutionary Computation*, (pp. 1725-1731). Cingapura.

Santos, A. G., & Mateus, G. R. (2007). Hybrid approach to solve a crew scheduling problem: an exact column generation algorithm improved by metaheuristics. *Proceedings of the 7th International Conference on Hybrid Intelligent Systems* (pp. 107-112). Kaiserslautern, Alemanha: IEEE Computer Society.

Sen, S. (1993). Minimal cost set covering using probabilistic methods. *ACM/SIGAPP Symposium on Applied Computing*, (pp. 157-164).

Sindicato Nacional dos Aeronautas. (8 de Fevereiro de 1988). *Regulamentação Profissional.* Acesso em 2008, disponível em http://www.aeronautas.org.br/sejur/lei7183_prn.html

Souza, C. C., Moura, A. V., & Yunes, T. H. (2005). Hybrid Column Generation Approaches for Urban Transit Crew Management. *Transportation Science , 39 (2)*, pp. 273-288.

Souza, M. J., Cardoso, L. X., Silva, G. P., Rodrigues, M. M., & Mapa, S. M. (2004). Metaheurísticas Aplicadas ao Problema de Programação de Tripulações no Sistema de Transporte Público. *Tendências em Matemática Aplicada e Computacional , 5(2)*, pp. 357-368.

Vasko, F. J., & Wilson, G. R. (1984). An efficient heuristic for large set covering problems. *Naval Research Logistics Quarterly , 31*, pp. 163-171.

Wren, A., & Wren, D. O. (1995). A genetic algorithm for public transport driver scheduling. *Computer & Operations Research , 22(1)*, pp. 101-110.