Jayme Assunção Casimiro
Orientador: Geraldo Robson Mateus

Modelos e Algoritmos para o Problema de Alocação de Tripulação em Redes de Transporte

Dissertação apresentada ao curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Ciência da Computação

Belo Horizonte
Junho de 2005

Aos meus pais e irmã que sempre me apoiaram, ao meu orientador que tanto me ajudou e a meus amigos do coração.

Sumário

Lista de Figuras

Lista de Tabelas

Abstract

The crew scheduling problem is a common activity that appears on public transport companies. The main goal of this problem is to assign the tasks to be carried out to different crews such that rules and regulations are respected and costs minimized. This problem is considered *NP-Hard* and generally it is split into two other problems: set covering and crew rostering. This thesis focusses on set covering problem. We give a formal definition, a literature revision and we present two heuristics to solve it – a lagrangean and a genetic one. Computational results are also presented to evaluate the performance of the implemented algorithms.
Keywords: Crew Scheduling Problem, Set Covering Problem, Genetic Algorithm, Lagrangean Algorithm, Crew Rostering Problem.

Resumo

O problema de alocação de tripulações em redes de transporte (PAT) é uma tarefa bastante rotineira no contexto de grandes empresas de transporte. Tal atividade envolve, basicamente, subdividir um conjunto de jornadas entre diferentes tripulações respeitando legislações trabalhistas e normas operacionais vigentes e impostas às empresas que atuam nesse setor. Por se tratar de um problema de grande complexidade computacional, o mesmo costuma ser divido em dois subproblemas: problema de recobrimento e problema de seqüenciamento de jornadas. Essa dissertação tem seu foco no Problema de Recobrimento. Além da definição formal do mesmo e de uma revisão da literatura, também são apresentados dois algoritmos para a sua resolução: um algoritmo lagrangeano e um genético. Por fim, resultados computacionais são apresentados com o objetivo de avaliar o desempenho dos algotimos apresentados.
Palavras-chave: Escalonamento de Tripulações, Problema de Recobrimento, Algoritmo Genético, Algoritmo Lagrangeano, Problema de Seqüenciamento de Jornadas.

Capítulo 1

Introdução

1.1 Motivação

O problema de alocação de tripulações em redes de transporte (PAT) é uma tarefa bastante rotineira no contexto de grandes empresas de transporte. Tal atividade envolve, basicamente, subdividir um conjunto de jornadas entre diferentes tripulações respeitando legislações trabalhistas e normas operacionais vigentes e impostas às empresas que atuam nesse setor.

A princípio, o PAT – parte integrante do problema de programação de viagens (PPV), era realizado por um profissional dentro de uma empresa de transporte que, baseado em sua intuição e experiência, realizava todas as tarefas envolvidas nesse problema.

Atualmente, com o crescimento da demanda por transporte e do número de restrições trabalhistas, essas empresas têm abandonado a utilização da intuição humana e despertado interesse pela utilização de ferramentas computacionais. As razões são diversas. Dentre essas podem ser citadas:

- O atual crescimento da demanda por transporte e do número de restrições trabalhistas envolvidas no PAT dificultam e, em alguns casos, impossibilitam tanto a obtenção de soluções de qualidade geradas pela intuição humana, quanto a estipulação de parâmetros capazes de avaliar a solução proposta.

- O cenário econômico atual faz com que empresas de transporte públicas e privadas, devido à crescente concorrência do setor, tenham como um de seus focos a redução de custos operacionais e aprimoramento de processos produtivos mantendo a qualidade de produtos e/ou serviços prestados.

- A mão-de-obra operacional é uma das componentes que mais pesa na planilha de custos de uma empresa de transportes. Assim, qualquer redução de custos nesse item, mesmo que não substancial, pode resultar na economia de valores consideráveis.

- O PAT é considerado um problema de alta complexidade computacional e um algoritmo especializado tende a apresentar uma visão mais ampla e detalhada tanto dos problemas a serem resolvidos quanto do campo de solução dos mesmos.
- Mesmo não sendo totalmente aproveitada, a solução gerada por algoritmos especializados pode servir de ponto de partida para um posterior refinamento minimizando o tempo gasto para a obtenção de uma solução final aceitável.

A área científica também tem manifestado interesse pelo PAT. Tal torna-se evidente devido aos esforços que têm sido feitos para se encontrar soluções de qualidade para esse problema caracterizado como *NP-Difícil* [KL96]. Além disso, a estratégia comumente utilizada para resolvê-lo consiste em dividi-lo em subproblemas. Esses, inúmeras vezes, são problemas clássicos da literatura e acabam por ser também uma das fases de resolução de outros problemas *NP-Difícil*. Logo, investir em estratégias e métodos que melhorem as soluções dos mesmos podem também resultar em um avanço na resolução de diversos problemas de pesquisa operacional e programação matemática.

1.2 Problemas Relacionados

O PAT é apenas uma dos problemas pertencentes ao PPV. Outros problemas como: problema de montagem de horários (PMH) e problema de escalonamento de veículos (PEV) podem também ser citados. A figura [1.1] ilustra os problemas envolvidos e a maneira como os mesmos se inter-relacionam.

1.2.1 Problema de Montagem de Horários

Independente do meio de transporte a ser levado em consideração – aéreo, ferroviário, rodoviário ou naval; a demanda por transporte nos mesmos não é homogênea. Observam-se inúmeras oscilações no nível de demanda pelo serviço. Essas variações podem ter causa sazonal – no caso de transportes de longa ou média distância, ou simplesmente horária – no caso de transportes públicos. Logo, no PMH, é necessário que todas essas variações sejam levadas em consideração. Na prática isso é feito através da divisão do espaço de tempo em períodos que apresentam características homogêneas.

Assim, o PMH envolve dois sub-problemas: a definição da quantidade de viagens que devem ser feitas por faixa horária e a atribuição do horário de partida a cada viagem. Enquanto a quantidade de viagens deve ser suficiente para atender a demanda por transporte, a distribuição das viagens visa satisfazer o usuário do meio de transporte a ser considerado de modo que ele espere o mínimo possível no caso de transportes públicos ou que o mesmo consiga chegar ao seu destino no momento desejado, no caso de transportes de média e longa distâncias. Além disso, o PMH leva também em consideração restrições operacionais que geralmente são impostas por autoridades locais, regionais ou nacionais. Essas, por sua vez, cumprem um papel de fundamental importância para a sociedade porque as mesmas

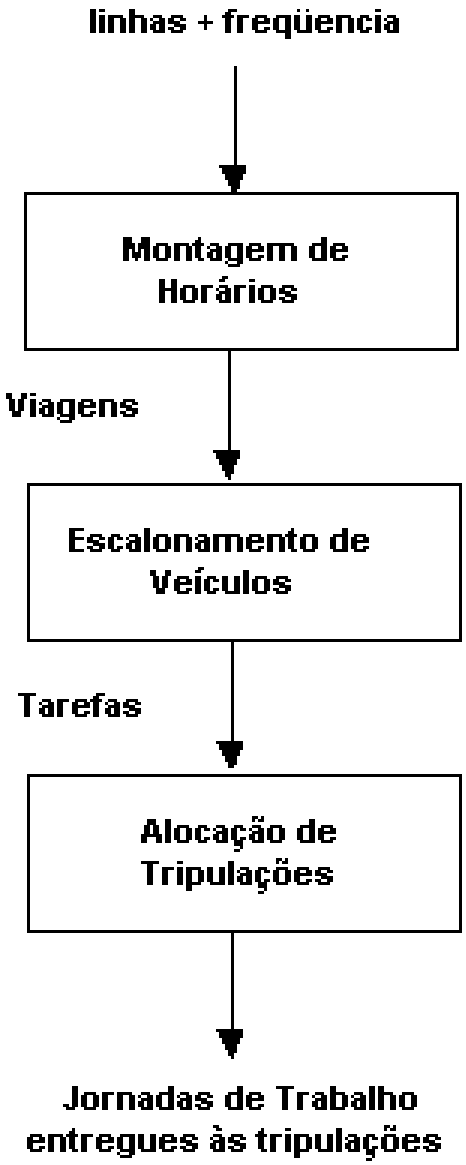

Figura 1.1: Problema de Programação de Viagens

mantém o serviço de transporte funcional até mesmo em horários pouco lucrativos com um patamar de qualidade adequado.

1.2.2 Problema de Escalonamento de Veículos

O PEV, etapa anterior ao PAT, reúne as viagens programadas no PMH em blocos. Um bloco apresenta a seqüência de viagens que um determinado veículo deve realizar em um período de tempo, começando e terminando na garagem. Cada bloco apresenta as *Oportunidades de Troca* (OT). A OT é um intervalo de tempo suficiente para que ocorra troca de tripulações. A cada oportunidade de troca está associado um *Horário de Troca* e um *Ponto de Troca*, os quais representam respectivamente a hora e um local onde poderá haver uma troca de tripulações [Löb99, **?**, FaW99].

A partir do bloco de um veículo são criadas as *Tarefas*. Cada tarefa é um conjunto de viagens reunidas com um custo associado que apresenta apenas duas OT: uma no início e outra no final da tarefa. Assim, em sua realização, não é possível que haja troca de tripulação.

O PEV, portanto, objetiva determinar um conjunto de custo mínimo de tarefas (veículos) de modo que cada uma das viagens programadas seja coberta por pelo menos ou exatamente por uma única tarefa. Alternativamente, o mesmo pode objetivar também minimizar o número total de tarefas e, portanto, de veículos necessários para cobrir todas as viagens programadas.

1.2.3 Problema de Alocação de Tripulação

O PAT consiste em gerar um conjunto de jornadas de trabalho que cubra todas as tarefas previstas e, posteriormente, alocá-las às tripulações que conduzirão a frota do meio de transporte que está em operação.

O conjunto de jornadas gerado deve contemplar todas as viagens sob responsabilidade da empresa e satisfazer a um conjunto de leis trabalhistas e regras operacionais com o menor custo possível. Este problema tem como dados de entrada a programação dos veículos obtida na fase anterior – PEV. Portanto, considera-se que já estejam definidos os blocos de viagens de cada veículo, isto é, as tarefas.

Após serem determinadas as jornadas de trabalho, essas devem ser distribuídas aos tripulantes observando regras de alocação básicas como número de folgas, repouso entre as jornadas, atendimento a pedidos e equilíbrio na distribuição da carga de trabalho para o período considerado.

Várias abordagens têm sido utilizadas para a resolução desse problema. A abordagem aqui utilizada, e tipicamente aplicada para a resolução do problema de escalonamento de mão-de-obra, consiste em dividi-lo em três fases principais: geração de jornadas, problema de recobrimento e seqüenciamento de jornadas:

1. **Geração de Jornadas -** um grande número de jornadas é gerado através da associação de tarefas afins e possíveis de serem executadas consecutivamente por uma

única tripulação. Essas jornadas devem respeitar regras trabalhistas impostas por uma classe de funcionários. Uma revisão sobre o assunto pode ser encontrada em [DW60]. [RMP89, RS94, Sav97] apresentam exemplos práticos de aplicações.

2. **Problema de Recobrimento -** é um problema clássico de otimização combinatória utilizado em inúmeras aplicações como problemas de escalonamento de mão-de-obra, roteamento de veículos e entrega de mercadorias, teste de circuitos VLSI, dentre outras. No contexto aqui aplicado, ele pode ser resumidamente descrito da seguinte maneira: apresentando como dado de entrada um conjunto de tarefas a serem executadas pelas tripulações e um conjunto de jornadas viáveis capazes de cobrir essas tarefas a um determinado custo, esse problema procura por um conjunto de jornadas capazes de atender toda a demanda a um custo mínimo. Várias heurísticas têm sido utilizadas para a resolução desse problema. Dentre essas se destacam: heurísticas lagrangeanas [CFT99, CNS98, Bea90]; algoritmos genéticos [BC96b, Ere98, SPU02]; algoritmo puramente guloso [Chv79] e algoritmos exatos [**?**, Bea87, BC96a].

3. **Problema de Seqüenciamento de Jornadas** - recebendo como entrada as jornadas anteriormente obtidas por meio da solução de (2), esse problema objetiva a criação de *rosters* – seqüência de jornadas possíveis de serem executadas por uma única tripulação dentro de um determinado intervalo de tempo. Além disso, para a construção de *rosters*, há restrições trabalhistas e operacionais de uma dada empresa de transporte que afetam a forma com que a alocação pode ser feita [YMdS00, CTFV98].

   A figura 1.2 mostra como essas três fases se inter-relacionam no PAT.

As duas subseções seguintes fazem um breve apanhado de algumas técnicas utilizadas para a resolução dos problemas de Geração de Jornadas e de Seqüenciamento de Jornadas. O Problema de Recobrimento é detalhado no capítulo 2.

1.2.4 Problema de Geração de Jornadas (PGJ)

Uma maneira natural de se representar o PGJ é através de grafos. Nessa abordagem, é dado um grafo $G = (V, A)$ que apresenta um conjunto V de nós ou vértices representando as tarefas a serem executadas e um conjunto A de arestas representando as possíveis transições entre essas tarefas. Vértices artificiais (depósitos) são criados para indicar o início e o fim de cada jornada.

1.2.4.1 Abordagens para o Problema de Geração de Jornadas

Três abordagens destacam-se dentre aquelas que têm sido utilizadas para a resolução do PGJ utilizando programação linear inteira: modelagem utilizando produto único, multi-produto e geração de colunas.

1.2.4.2 Abordagem utilizando produto único

Para cada arco $(i, j) \in A$ é associada uma variável binária x_{ij}. Jornadas inviáveis são evitadas por meio de restrições. Observa-se no entanto que restrições complicadas de serem representadas através de modelos lineares costumam apresentar relaxações lineares ruins quantitativamente [BC96c, BC98, CDMT89].

1.2.4.3 Abordagem utilizando multi-produto

Para cada arco $(i, j) \in A$ e um depósito k é associada uma variável binária x_{ij}^k. Jornadas inviáveis são também modeladas por meio de restrições. No entanto, a presença de vários depósitos melhora os limites da relaxação linear [Löb99, FHW94].

1.2.4.4 Abordagem utilizando geração de colunas

Essa abordagem tem sido caracterizada de duas maneiras na literatura:

1. **Geração de Colunas Dinâmica** - Baseado nas abordagens clássicas de geração de colunas, inicialmente, um conjunto de jornadas viáveis é gerado. O problema mestre, *Set Partitioning*, relaxado de suas condições de integralidade, é resolvido fornecendo preços para o subproblema – problema de caminho mínimo com restrições. Esse, por sua vez, gera quotas – jornadas com custo reduzido negativo, que serão enviadas ao problema mestre. Na inexistência de quotas com custo reduzido negativo, está provada a otimalidade do problema não inteiro. Passa-se, portanto, para o processo de geração de soluções viáveis inteiras a partir das colunas geradas. Como o politopo linear pode não coincidir com o politopo inteiro e sendo o problema de *Set Partitioning* extremamente degenerado, a convegência através desse método pode ser lenta. No entanto, essa abordagem, além de gerar limites de qualidade, evita, na prática, a geração de todas as colunas do problema [BJN$^+$94].

2. **Geração de Colunas Offline** - Nessa abordagem o PGSJ é divido em duas etapas. Em primeira instância todas as jornadas do problema são geradas respeitando as restrições de viabilidade impostas para uma jornada. Essas jornadas servem de entrada para o problema *Set Partitioning* ou *Set Covering*. Esses, por sua vez, serão os responsáveis por selecionar um conjunto de mínimo custo de jornadas [CFT99]. A geração de colunas, como anteriormente especificado, gera jornadas redundantes e, muitas vezes, um conjunto de dimensões intratáveis computacionalmente. Com isso, de modo a evitar tal situação, existem heurísticas que visam diminuir o número de colunas geradas. No entanto, a perda da otimalidade pode ser um problema com esse tipo de abordagem [KJN01].

1.2.5 Problema de Seqüenciamento de Jornadas

O PSJ também é geralmente representado por meio de grafos. No entanto, dado o grafo $G = (V, A)$, V representa o conjunto de jornadas ótimas do PGSJ e A um conjunto de arestas representando as possíveis transições entre essas jornadas.

O objetivo do PSJ é gerar um seqüenciamento viável de jornadas, começando e terminando em um mesmo depósito, atendendo restrições de seqüenciamento e operacionais específicas para cada aplicação. O esforço de otimização pode focar em diversas direções. A literatura tem apresentado algoritmos que tendem a:

- minimizar o número de horas livres de trabalho de uma tripulação [CFL$^+$98, CTFV98];
- minimizar o número de rosters gerados e, por conseqüência, o número de tripulações necessárias para executá-los [YMdS00];
- maximizar a satisfação de uma tripulação tentando distribuir, balanceadamente, o número de jornadas caracterizadas como "pesadas"[1] entre os diversos rosters.

Como algumas combinações entre os objetivos acima assinalados são conflitantes, opta-se por uma abordagem multi-objetivo ou por aquelas que mais se aproximam da aplicação que se pretende resolver.

1.2.6 Abordagens para o Problema de Seqüenciamento de Jornadas

Dentre as estratégias para a resolução do PSJ as que têm obtido melhores resultados utilizam geração de colunas, geração de colunas com programação por restrições, relaxação lagrangeana e relaxação lagrageana com programação por restrições.

1.2.6.1 Abordagem utilizando geração de colunas

Nessa abordagem utiliza-se o modelo clássico para geração de colunas. O problema mestre irá resolver o problema de *SetPartitioning* relaxado selecionando rosters de custo mínimo e fornecendo preços ao subproblema. O subproblema, por sua vez, resolve um problema de caminho mínimo com restrições fornecendo quotas (rosters) ao problema mestre [BJN$^+$94].

1.2.6.2 Abordagem utilizando geração de colunas e programação por restrições

No PSJ diversas restrições operacionais devem ser levadas em consideração de modo a se obter uma solução viável. Linguagens baseadas em programação por restrições provêem

[1] jornadas que exigem trabalhos noturnos, jornadas com uma carga de trabalho superior a um dado valor, etc.

um arcabouço de facilidades que tornam a especificação de restrições mais próximas da linguagem humana facilitando, em muito, a programação. Além disso, algoritmos, quando devidamente implementados utilizando tais linguagens, tendem a ser eficientes.

A abordagem utilizando geração de colunas e programação por restrições consiste, portanto, em implementar o problema mestre utilizando técnicas da pesquisa operacional – Programção Linear para o presente problema e resolver o subproblema – caminho mínimo com restrições, através da programação por restrições [CFL+98, FJK+99, YMdS00].

1.2.6.3 Abordagem utilizando relaxação lagrangeana

Caprara e outros em [CTFV98] constroem rosters viáveis utilizando informações obtidas a partir do problema de assinalamento [2] e dos limites inferiores encontrados para o problema relaxado.

1.2.6.4 Abordagem utilizando relaxação lagrangeana e programação por restrições

Caprara e outros em [CFL+98] também conjugam técnicas da pesquisa operacional com programação por restrições para a resolução do PSJ. No entanto, a construção de rosters viáveis utilizando essa abordagem é viabilizada utilizando a programação por restrições em conjunção com os limites dinamicamente obtidos através da relaxação lagrangeana [CTFV98].

Essa dissertação tem seu foco em uma das fases do PAT que é o Problema de Recobrimento (PR) e está divida da seguinte forma: o capítulo 2 define o problema a ser tratado, lista alguns problemas a ele relacionado, apresenta um breve histórico das abordagens utilizadas para sua resolução e descreve algumas de suas aplicações típicas. Os capítulos 3 e 4 detalham dois algoritmos implementados para a resolução do *PR*. O capítulo 5 apresenta os resultados obtidos com as implementações realizadas. O capítulo 6 apresenta as conclusões e os trabalhos futuros.

[2] O modelo adotado por esse grupo aplica a relaxação lagrangeana nas restrições que irão garantir a viabilidade operacional de um roster. Com isso, os mesmos recaem sobre um problema de assinalamento.

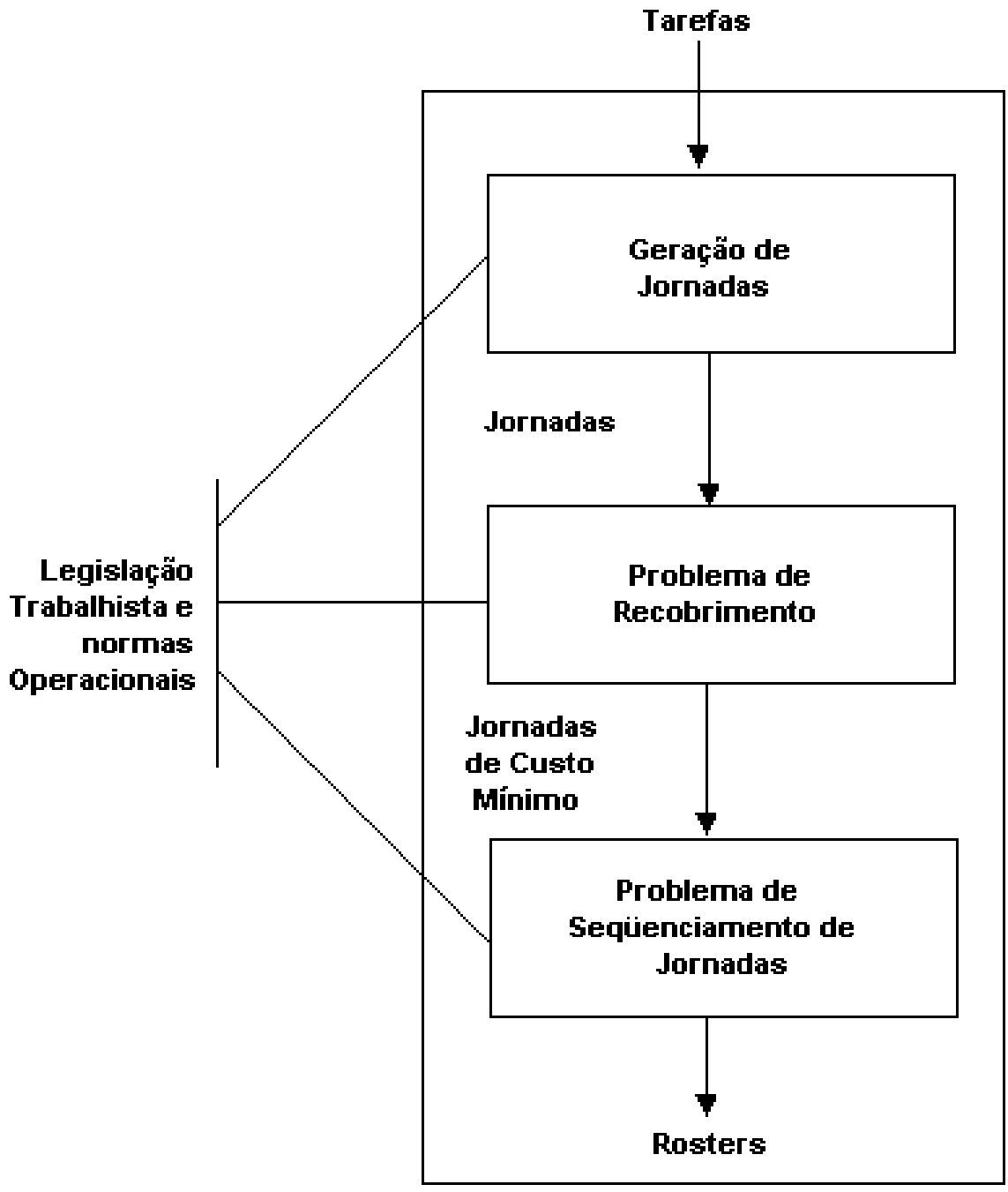

Figura 1.2: Resumo esquemático das fases do problema de alocação de tripulações.

Capítulo 2

Problema de Recobrimento

2.1 Definição do Problema

O problema de recobrimento (PR) corresponde à cobertura de m-linhas de uma matriz 0-1 $[a_{ij}]$, m x n, por um subconjunto de n colunas. Sendo:

- $M = 1, 2, ..., m$, o conjunto de linhas a serem cobertas.
- $N = 1, 2, ..., n$, o conjunto de colunas.
- $c_j (j \in N)$, o custo da coluna j quando a mesma encontra-se na solução do PR.
- $a_{ij} = 1$, se a coluna $j \in N$ cobre a linha $i \in M$.

Pode-se dizer que o PR procura por um subconjunto de mínimo custo $S \subseteq N$ de colunas de tal forma que cada linha $i \in M$ é coberta por, pelo menos, uma coluna $j \in S$. Um modelo matemático para o PR pode ser descrito da seguinte forma:

$$f(PR) = min \sum_{j \in N} c_j x_j \tag{2.1}$$

Sujeito a:

$$\sum_{j \in N} a_{ij} x_j \geq 1, i \in M \tag{2.2}$$

$$x_j \in \{0, 1\}, j \in N \tag{2.3}$$

Para melhor compreensão do texto que se segue também foram definidos:

$$J_i = \{j \in N : a_{ij} = 1\}, \forall i \in M \tag{2.4}$$

$$M_j = \{i \in M : a_{ij} = 1\}, \forall j \in N \tag{2.5}$$

A notação (2.4) explicita o conjunto de colunas que cobrem cada linha $i \in M$. A notação (2.5) explicita o conjunto de linhas cobertas por cada coluna $j \in J$.

2.2 Problemas Relacionados

2.2.1 Problema de Recobrimento de Mínima Cardinalidade (PRMC)

Em algumas situações a diferenciação de custos entre as colunas participantes do PR é desnecessária. Nesses casos, obtém-se um problema cujo custo pode ser abstraído da função objetivo. Ou seja, todas as colunas passam a ter custo de valor unitário. A esse problema damos o nome de Problema de Recobrimento de Conjunto de Mínima Cardinalidade. Formalmente ele pode ser representado da seguinte maneira:

$$f(PRMC) = min \sum_{j \in N} x_j$$

sujeito a:

$$\sum_{j \in N} a_{ij} x_j \geq 1, i \in M$$

$$x_j \in \{0, 1\}, j \in N$$

2.2.2 Problema de Particionamento (PP)

Quando substituimos a restrição de desigualdade da equação (2.2) por uma de igualdade temos o chamado PP. Ou seja, a formulação do PP exige que cada linha do problema seja coberta por uma única coluna j do conjunto N. Obviamente, toda solução para o PP é também uma solução para o PR. No entanto, a recíproca não é verdadeira.

$$f(PP) = min \sum_{j \in N} c_j x_j$$

sujeito a:

$$\sum_{j \in N} a_{ij} x_j = 1, i \in M$$

$$x_j \in \{0, 1\}, j \in N$$

2.2.3 Problema de Empacotamento (PE)

Problemas de empacotamento consistem em colocar, de forma econômica, uma coleção de objetos dentro de recipientes. A formulação mais simples e clássica encontrada na literatura é uma variante do PR. Nela, a restrição de desigualdade maior ou igual é substituída por uma restrição do tipo menor ou igual e a função objetivo é de maximização. Esse problema é de grande aplicação na resolução de problemas de corte geométricos comuns em indústrias têxtil, acondicionamento de embalagens e carregamento de veículos [NS92, ANS00].

$$f(PE) = max \sum_{j \in N} c_j x_j$$

sujeito a:

$$\sum_{j \in N} a_{ij} x_j \leq 1, i \in M$$

$$x_j \in \{0, 1\}, j \in N$$

2.3 Revisão da Literatura

Algoritmos para resolução do PR foram e ainda têm sido muito estudados devido ao grande número de problemas que em todo ou em parte recaem sobre seu modelo. Dentre esses se destacam: algoritmos exatos, heurísticas gulosas, heurísticas lagrangeanas, algoritmos genéticos, algoritmos híbridos, dentre outros.

Dentre os algoritmos exatos os mais utilizados baseiam-se em métodos enumerativos e planos de corte. Nos métodos enumerativos todas as soluções viáveis para o PR são listadas e a melhor dentre essas é a ótima. Apesar de garantir a otimalidade da solução encontrada, esse método é dito ser inviável, uma vez que o tempo dispendido para listar todas as soluções do problema aumenta exponencialmente com o crescimento do número de variáveis do mesmo. Uma alternativa utilizada para otimizar o desempenho dessa classe de algoritmos consiste em utilizar técnicas de enumeração implícita. As mesmas utilizam-se das restrições do problema e da integralidade das variáveis para abandonar enumerações que não conduzem às soluções melhores. Ou seja, esse método permite a obtenção de soluções ótimas sem que todas as possíveis soluções sejam calculadas. Algoritmos enumerativos foram utilizados por Lemke e outros [LSS71] e Etcheberry [Etc77].

Algoritmos de planos de corte deduzem desigualdades suplementares àquelas existentes no PR a partir de suas próprias restrições e condições de integralidade de suas variáveis. Essas desigualdades "cortam"parte da região de viabilidade do PR diminuindo, portanto, o espaço de busca por soluções. No entanto, nenhum dos cortes gerados perde a região de otimalidade do problema a ser resolvido. Em geral, uma abordagem por planos de corte envolve a solução de uma seqüência de problemas da seguinte forma: dado um PR, encontra-se uma solução inicial inicial x^p, de custo c^p. Se uma nova desigualdade puder ser

obtida de tal forma que todas as soluções com custo menor que c^p a satisfazem e x^p não a satisfaz então um novo problema *PR'* é obtido pela inclusão dessa desigualdade. Se tal desigualdade não puder ser deduzida então a solução x^p de custo c^p é a solução ótima do problema. Bellmore e Ratliff [BR71] e Balas e Ho [BH80] utilizam planos cortantes para a resolução do *PR*. Balas e Ho, em especial, testaram diversas técnicas de relaxações lagrangeanas utilizando otimização por subgradiente introduzindo também heurísticas primais e duais combinadas com técnicas de fixação de variáveis e novas regras de ramificação.

Balas e Carrera [BC96a] apresentaram um algoritmo *branch-and-bound* para o *PR* cujo ponto central era um procedimento que integrava limite superior e limite inferior – "otimização do subgradiente dinâmico". Este novo procedimento, aplicado ao dual lagrangeano em todo nó da árvore de busca, combinava o método do subgradiente padrão com heurísticas primais e duais que interagiam para mudar os multiplicadores de lagrange, ajustar os limites superiores e inferiores e fixar variáveis. Este método obteve performance considerável quando comparado com outros algoritmos heurísticos e exatos existentes.

Algoritmos exatos devido ao seu alto custo computacional são, na prática, pouco utilizados. No entanto eles são importantes porque através deles torna-se possível avaliar as soluções produzidas por outra classe de algoritmos – as heurísticas.

Métodos heurísticos são capazes de gerar soluções de "qualidade"para o *PR*. No entanto, não há garantia de otimalidade. Heurísticas, comumente, obtém uma solução viável S para o *PR* iterativamente. Inicialmente nenhuma coluna pertence ao conjunto solução, isto é, $S = \emptyset$. A cada iteração, baseado em algum critério, uma coluna de índice j é selecionada de N, e passa a compor a solução S. Tal procedimento é utilizado repetidamente até que uma solução viável seja encontrada.

Algoritmo 1: Algoritmo Guloso para o PR

```
1 Entrada: M, N, c
2 Saída: Solução viável S para o PR
3 Inicializar variáveis R ← M, S ← ∅
4 enquanto R ≠ ∅ faça
5 |   k = escolheMelhorColuna{f(c_j, |P_j|), ∀j ∈ N};
6 |   S ← S ∪ {j}
7 |   R ← R \ M_k;
8 fim
```

O algoritmo 1 resume o procedimento anteriormente descrito. Nele, as linhas 4-8 constroem, iterativamente, uma solução viável S para o *PR*. A função destacada na linha 5, "*escolheMelhorColuna*"retorna uma coluna k que é selecionada segundo algum critério que, geralmente, leva em conta tanto o custo da coluna associada quanto o número de linhas cobertas por ela.

Chvátal [Chv79] utiliza o algoritmo 1 e seleciona, a cada iteração, uma coluna $j \in N$ que apresenta o menor valor $c_j/|M_j|$ associado. O seu trabalho foi decorrente do aperfeiçoamento das propostas de Johnson [Joh74] e Lovász [Lov79] que podiam ser aplicadas

ao PR não ponderado.

Balas e Ho [**?**] consideram cinco funções de avaliação para selecionar uma coluna, sendo uma delas aquela proposta por Chvátal:

$$c_j \tag{2.6}$$

$$\frac{c_j}{|M_j|} \tag{2.7}$$

$$\frac{c_j}{log_2|M_j|} \tag{2.8}$$

$$\frac{c_j}{|M_j|log_2|M_j|} \tag{2.9}$$

$$\frac{c_j}{|M_j|ln|M_j|} \tag{2.10}$$

Nas fórmulas (2.8) e (2.9), $log_2|M_j|$ é substituído por 1 quando $|M_j| = 1$, e no caso (2.10), $ln|M_j|$ é substituído por 1 quando $|M_j| = 1$ ou 2. A função (2.6) seleciona a coluna de menor custo. Em instâncias cujo valor dos custos das colunas são praticamente iguais ela se torna ineficaz. A função (2.7) minimiza o custo unitário de cobertura de uma linha não coberta. As funções (2.8) a (2.10) apenas modificam o peso que se dá ao fator $|M_j|$. Além de novas funções de avaliação esses autores incluiram a seu algoritmo um passo adicional para a retirada de colunas redundantes da cobertura como mostrado no algoritmo 2.

Algoritmo 2: Algoritmo para Retirada de Colunas Redudantes

```
1 Ordenar a cobertura S em ordem não crescente de custos ou seja:
  S = {j_1, j_2, ..., j_t} correspondendo a {c_j1 ≥ c_j2 ≥, ..., ≥ c_jt}
2 para j = 1 até t faça
3 |   se S \ {j} é cobertura então
4 |   |   S ← S \ {j}
5 |   fim
6 fim
```

Vasco e Wilson [VW84] apresentam uma heurística gulosa que utiliza as funções de avaliação propostas por Balas e Ho [**?**] e adiciona duas funções: (2.11) e (2.12). Eles propõem uma escolha aleatória das sete funções de avaliação. Além da retirada de colunas redundantes há também a busca por soluções vizinhas melhores. Ou seja, dada uma cobertura S^*, se existe uma coluna j ($j \in N$ e $j \notin S^*$) capaz de substituir uma coluna k de S^* de maior custo sem que S^* perca a viabilidade, então j entrará no conjunto solução de S^* e k sairá.

$$\frac{c_j}{|M_j|^2} \tag{2.11}$$

$$\frac{c_j^{\frac{1}{2}}}{|M_j|^2} \tag{2.12}$$

Feo e Rezende [FR89] propõem uma variação não determinística da heurística gulosa proposta por Chvátal. O método por eles proposto, ao invés de selecionar uma coluna $k \in N$ capaz de minimizar a função de avaliação (2.7), seleciona, aleatoriamente, uma coluna dentro de um conjunto de índices candidatos que possuem função de avaliação igual ou menor a um determinado valor. Além disso, a extração de colunas redundantes também é efetuada. A busca aleatória dentro de um conjunto de "boas"colunas permite explorar soluções vizinhas que possam resultar em uma melhora das soluções e evita também a obtenção de soluções de mínimo local.

Beasley [Bea90] introduziu a heurística baseada em Relaxação Lagrangeana e Otimização por Subgradientes que aproveita-se dos multiplicadores lagrangeanos para gerar, a cada iteração do subgradiente, uma cobertura.

Beasley e Chu em [BC96b] utilizam um algoritmo genético com representação binária para a resolução do *PR*. Nele cromossomos inviáveis são transformados em soluções por meio de um operador genético apropriado que além de viabilizar também efetua uma busca local que muitas vezes é capaz de melhorar a solução do cromossomo corrente. Um novo operador de *crossover* – *crossover fusion* é também introduzido nesse artigo.

Um algoritmo genético baseado em uma representação não binária é proposto por Eremeev em [Ere98]. Nessa abordagem um cromossomo é uma string cujo tamanho é igual ao número de linhas do problema a ser resolvido e a i-ésima posição dessa estrutura contém o índice da uma coluna que cobre a correspondente linha. Como conseqüência dessa abordagem tem-se que todos os cromossomos compõem soluções viáveis não havendo, portanto, a necessidade de se empregar uma heurística que garanta a sua viabilidade como em [BC96b]. Heurísticas são utilizadas objetivando a eliminação de colunas redundantes e melhoria de soluções por meio de uma busca local.

Caprara e outros em [CFT99] propõem a heurística *CFT* que, aproveitando-se dos multiplicadores lagrangeanos encontrados na fase de Subgradiente, determina soluções viáveis para o *PR*. O algoritmo por eles proposto trabalha, a cada iteração, com um subconjunto de colunas que é dinamicamente atualizado baseado no custo reduzido lagrangeano de cada coluna. Além disso, mecanismos de fixação de colunas são também utilizados com o objetivo de reduzir o espaço de busca por soluções e melhorar a melhor solução disponível.

Solar e outros em [SPU02] propõem um algoritmo genético paralelo acompanhado de seu respectivo modelo adaptado para a solução do *PR*. Nele cromossomos apresentam estrutura similar àquela proposta por [BC96b].

2.4 Aplicações do Problema de Recobrimento

O *PR* é um modelo essencialmente importante para diversas aplicações da Pesquisa Operacional. Dentre essas, inclui-se:

1. **escalonamento de tripulações:** dado um conjunto de tarefas com tempo de início e término a serem executadas segundo uma periodicidade pré-estabelecida, jornadas – agrupamento de tarefas que podem ser realizadas seqüencialmente sem descanso, são geradas. Nesse caso, jornadas e tarefas representam, respectivamente, colunas e linhas no PR e objetiva-se, analogamente, selecionar um conjunto de mínimo custo de jornadas de modo que todas as tarefas sejam cobertas. As jornadas selecionadas darão origem a vários *rosters* – seqüenciamento viável de jornadas que respeita regras trabalhistas. Cada um desses *rosters* será executado por uma única tripulação [BC96c, BC98, KJN01, FLO00, Lev96, vdAvHS99].

2. **roteamento de veículos:** corresponde à atribuição de viagens programadas a veículos. Dado um conjunto de rotas viáveis geradas de acordo com restrições dependentes do meio do transporte a ser utilizado – colunas no PR, objetiva-se selecionar um subconjunto dessas de modo que todas as viagens – linhas no PR, sejam cobertas. Selecionadas as rotas, um seqüenciamento viável das mesmas é efetuado. Cada um desses seqüenciamentos será atribuído a um veículo [CDMT89, Löb99, FHW94].

3. **minimização de circuitos lógicos:** dada uma função lógica booleana, procura-se por outra equivalente àquela, cuja implementação seja de mínimo custo [Sen93]. A solução ótima pode ser encontrada determinando, em primeira instância, todos os *primos implicantes*[1] de uma dada função e, posteriormente, selecionando um subconjunto desses de tal forma que todos os *mintermos*[2] da mesma sejam cobertos e ao mesmo tempo, a soma dos custos associada a cada *primo implicante* seja minimizada. Considerando a formulação apresentada na seção 2 pode-se dizer que o conjunto de linhas a serem cobertas corresponde aos *mintermos* e cada coluna capaz de cobrir uma ou mais linhas corresponde a um *primo implicante*.

4. **problema de cobertura em grafos:** Dado um grafo $G = (N, A)$ com $n = |N|$ nós e $q = |A|$ arestas podemos definir os seguintes problemas sobre ele:

   - *cobertura de nós:* é o problema de determinar o conjunto de número mínimoN' de nós, $N' \subseteq N$, tal que toda a aresta de G é incidente a pelo menos um nó de N'. Nesse problema os nós correspondem às colunas no PR e as arestas às linhas.
   - *cobertura de arestas:* é o problema de determinar o conjunto de número mínimo A' de arestas, $A' \subseteq A$, de tal forma que todo vértice de G é incidido por pelo menos uma aresta de A'. Para esse problema as arestas correspondem às colunas no PR e as linhas aos vértices.

5. **problema de localização de facilidades:** desejamos localizar um conjunto P de facilidades dentro de um espaço pré-definido. Essas facilidades deverão atender um

[1] Grupo de elementos que podem ser combinados em um *mapa de Karnaugh*

[2] Termo de uma função booleana na forma de soma de produtos.

$$\begin{pmatrix} 1 & 2 & 4 \\ 2 & 3 & 5 \\ 4 & 5 & 7 \\ 5 & 6 & 1 \\ 6 & 7 & 2 \\ 7 & 1 & 3 \end{pmatrix}$$

Figura 2.1: Exemplo de uma tripla de Steiner

conjunto M de pontos de demanda. As posições que as facilidades podem assumir são determinadas a priori e compõem o conjunto N. Cada uma das posições $j \in N$ é capaz de suprir a demanda de um ou mais pontos. Assim, tem-se que o conjunto M corresponde às linhas no PR e o conjunto N às colunas. O objetivo perseguido pode ser o de minimizar o número de facilidades a serem localizadas no caso em que as mesmas apresentam custo unitário ou minimizar o custo das facilidades selecionadas considerando o PR com custos fixos não unitários.

6. **tripla de Steiner:** a dimensão-β de uma matriz A 0-1 m x n, é o número mínimo de colunas que podem ser selecionadas de A, tal que todas as somas de elementos das linhas da sub-matriz resultado seja pelo menos β. As matrizes de incidência obtidas de sistemas de triplas de Steiner possuem as seguintes características [Wed95b]:

   - possuem exatamente 3 uns em cada linha. Dizemos que i, j, k é uma tripla de A se existe uma linha r de A tal que $a_{ri} = a_{rj} = a_{rk} = 1$.
   - para cada par de colunas j_1 e j_2, existe exatamente uma linha i tal que $a_{ij1} = a_{ij2} = 1$.
   - tais matrizes existem se e somente se $n \geq 3$ e $mod(n, 6) = 3$, sendo $m = \frac{1}{6}n(n-1)$.
   - cada coluna de A contém exatamente $\frac{1}{2}(n-1)$ entradas não nulas.

   Assim, triplas de Steiner geradas correspondem às linhas no PR. As colunas, por sua vez, são automaticamente determinadas quando se constrói uma tripla e apresentam custo unitário. Logo, os problemas gerados por sistemas de triplas de Steiner são instâncias do $PRMC$. A figura 2.1 apresenta um exemplo de uma tripla de Steiner com $\beta = 7$.

Capítulo 3

Um Algoritmo Lagrageano Híbrido para o Problema de Recobrimento

3.1 Introdução

Encontrar uma boa solução para um problema *NP-Difícil* requer a consideração de dois aspectos:

- O cálculo de um limite superior tão próximo quanto possível do ótimo;
- O cálculo de um limite inferior tão próximo quanto possível do ótimo.

Para a geração de limites superiores a um custo computacional razoável são utilizadas heurísticas. Para a determinação de limites inferiores, por sua vez, geralmente são utilizadas técnicas de relaxação linear – *RL*. Nela, um modelo de programação inteira tem relaxadas as condições de integralidade de suas variáveis. O modelo resultante pode ser resolvido utilizando algoritmos exatos tradicionais como o simplex ou pontos interiores. A solução obtida é um limite inferior para a solução ótima do problema original.

Outra técnica bastante conhecida para a determinação de limites inferiores é a relaxação lagrangeana – *RLA*. A mesma, inicialmente, prepara um modelo de programação inteira ou mista através dos seguintes passos:

1. um ou mais conjuntos de restrições do problema são escolhidos para serem relaxados;
2. multiplicadores lagrangeanos são anexados às restrições selecionadas de modo a inseri-las na função objetivo.
3. Preparado o modelo relaxado, o mesmo é resolvido de forma exata.

Essa abordagem tem sido bastante utilizada na literatura e se fundamenta nos seguintes fatos:

- Muitos problemas de otimização apresentam um problema fácil (solúvel por um algoritmo de complexidade polinomial) que é "complicado"pela adição de algumas restrições específicas. Abstraindo as restrições que dificultam a resolução do problema, adicionando-as à função objetivo (passo 2), restará um modelo de fácil solução sendo preciso apenas determinar o valor numérico dos multiplicadores de Lagrange.

- Experiências práticas utilizando relaxação lagrangeana indicaram que ela produz limites inferiores de qualidade, a um custo computacional razoável, em muitos problemas de otimização.

A *RLA* foi primeiramente aplicada no início dos anos 70 no problema do caixeiro viajante por Held e Karp [HK70, HK71]. Hoje essa tem sido uma técnica bastante aplicada objetivando a obtenção de limites inferiores e multiplicadores lagrangeanos que podem ser utilizados em substituição àqueles obtidos por meio de uma relaxação linear.

No contexto do *PR* a relaxação lagrangena foi utilizada tanto em algoritmos que objetivavam a obtenção de soluções exatas [BH80, BC96a], quanto em algoritmos heurísticos [Bea90, CTFV98]. Beasley [Bea90] foi o precursor na utilização de uma heurística baseada em relaxação lagrangeana e otimização por subgradientes que gerava uma solução após cada iteração do subgradiente. O algoritmo por ele desenvolvido serviu de base para vários outros que foram implementados [CTFV98].

Este capítulo propõe uma heurística híbrida lagrangena – *HHS*, composta de três fases. A mesma, a cada iteração, seleciona um subconjunto diferente de colunas – primeira fase, que serão utilizadas para a determinação de soluções viáveis na segunda fase. Essas fases são iterativamente alternadas até que o critério de parada seja atingido. A terceira e última fase utiliza o algoritmo Simplex e o *branch-and-bound* para determinar a melhor solução ou uma solução viável para um subconjunto de colunas escolhido por meio de uma função de avaliação.

Esse capítulo é organizado da seguinte forma: primeiramente é mostrado o modelo relaxado lagrangeano para o *PR* (seção 3.2) seguido de duas técnicas bastante utilizadas na literatura para a obtenção de multiplicadores lagrangeanos próximos do ótimo – método subgradiente clássico e método subgradiente feixe (seção 3.2.1). Em seguida o algoritmo *HHS* é apresentado (seção 3.3).

3.2 O modelo lagrangeano para o *PR*

Seguindo-se os passos apresentados na seção anterior para a relaxação lagrangeana de um modelo de programação linear inteira ou mista e, escolhendo-se a restrição (2.2) para ser relaxada, obtém-se o seguinte problema:

$$f(PR) = min_{u \geq 0} \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} u_i \left(1 - \sum_{j=1}^{n} a_{ij} x_j \right)$$

Sujeito a:

$$x_j \in \{0,1\}, j \in N$$

$$L(u) = min \sum_{j=1}^{n} \left[c_j - \sum_{i=1}^{m} u_i a_{ij} \right] x_j + \sum_{i=1}^{m} u_i$$

Aplicando-se as definições (2.4) e (2.5) chega-se a:

$$L(u) = min \sum_{j \in N} c_j(u)\ x_j + \sum_{i=1}^{m} u_i \tag{3.1}$$

$$x_j \in \{0,1\}, j \in N \tag{3.2}$$

Onde:

$$c_j(u) = c_j - \sum_{i \in M_j} u_i \tag{3.3}$$

é o custo lagrangeano associado à coluna $j \in N$.

Dado um vetor $u \geq 0$ tem-se que a solução ótima para (3.1) e (3.2) é: $x_j = 1$ se $c_j(u) < 0$, $x_j = 0$ se $c_j(u) > 0$ e $x_j \in \{0,1\}$ quando $c_j(u) = 0$.

$L(u)$ é um limite inferior para o problema cuja solução por ele apresentada nem sempre será viável no problema original. Entretanto, o custo reduzido das colunas obtido com o mesmo compõe fonte confiável e eficiente para a construção de um limite superior para o problema no sentido que soluções com menor custo reduzido apresentam maior probabilidade de estar na solução ótima do problema.

Nas técnicas de relaxação lagrangeana encontrar os multiplicadores lagrangeanos é de suma importância quando se deseja gerar limites inferiores de qualidade. Afinal, é desejável encontrar limites inferiores que sejam os mais próximos possíveis da solução ótima do problema. A seção seguinte 3.2.1 apresenta dois algoritmos para a determinação de multiplicadores de Lagrange – otimização por subgradiente clássico e feixe.

3.2.1 A determinação de multiplicadores lagrangeanos

Para se determinar vetores de multiplicadores lagrangeanos próximos do ótimo dentro de um pequeno intervalo de tempo, uma boa aproximação consiste em utilizar um vetor subgradiente $s(u) \in R^m$ associado a um dado u definido da seguinte forma:

$$s_i(u) = 1 - \sum_{j \in J_i} x_j(u) \quad , \qquad i \in M \tag{3.4}$$

Com esse subgradiente é possível gerar uma seqüência de vetores $u^0, u^1, u^2...$ não negativos onde u^0 pode ser arbitrariamente definido e os restantes u^k, $k \geq 1$ podem ser determinados por meio da seguinte seqüência proposta por Held e Karp [HK71]:

$$u_i^{k+1} = max \left\{ u_i^k + \lambda \frac{UB - L(u^k)}{\|s(u^k)\|^2}, 0 \right\}, \qquad i \in M \tag{3.5}$$

onde: UB é um limite superior para (2.1) e $\lambda \geq 0$ corresponde a um salto na direção de crescimento da função (3.1).

Abordagens como o método subgradiente – Bertsekas [Ber95], método subgradiente feixe - Kokott e Löbel [KL96] e método da coordenada ascendente - Wedelin [Wed95a], são utilizadas com o objetivo de se determinar multiplicadores lagrangeanos próximos do ótimo.

Algoritmo 3: Algoritmo de Otimização pelo Método Subgradiente Clássico

Entrada: UB, M, N, c, passo inicial $\lambda \geq 0$, $p > 0$ igual ao número limite de iterações do método.
Saída: u^*, vetor de multiplicadores lagrangeanos ótimo ou próximo do ótimo e $L(u^*)$ o limite inferior para o problema

$k \leftarrow 0$
Obtenha u^k através da fórmula $u_i^k = min_{j \in Ji} \frac{c_j}{|I_j|}$, $i \in M$.
$\lambda \leftarrow 0.1$
$L(u^k) \leftarrow \infty$
$s(u_i^k) \leftarrow \infty$, $\forall i \in M$

enquanto *nas últimas p iterações, houver melhora no valor de* $L(u)$ **ou** $L(u^k) <> UB$) **ou** $s(u^k) <> 0$ **faça**
    Calcule o vetor de custos reduzidos da iteração k, $c_j^k(u)$, através da equação (3.3).
    Calcule a solução do problema da iteração k para todo $j \in N$

$$x_j^k = \begin{cases} 0 & se\ c_j^k(u) > 0 \\ 0\ ou\ 1 & se\ c_j^k(u) = 0 \\ 1 & se\ c_j^k(u) < 0 \end{cases}$$

    Determine o valor do limite inferior da iteração k, $L^k(u)$, através da equação (3.1)
    Atualize o melhor valor encontrado para o limite inferior $L(u^*)$ e (u^*) caso seja necessário.
    Calcule o vetor subgradiente da iteração k, s^k, por meio da equação (3.4).
    Calcule o vetor de multiplicadores de Lagrange da iteração $k+1$, u^{k+1}, por meio de (3.5).
    Se o melhor e o pior $L(u)$ das últimas p iterações diferem-se por mais de 1% divida o valor de λ por 2. Caso contrário, sendo essa diferença menor que 0.1%, multiplique λ por 2.
    $k = k + 1$
fim
Retorne u^* e $L(u^*)$.

O algoritmo 3 apresenta o método subgradiente utilizado em muitas implementações para obtenção de limites inferiores e multiplicadores lagrangeanos. Ele recebe como entrada os conjuntos N e M, um limite superior UB disponível, um passo na direção de crescimento da função – λ, o custo de cada coluna pertencente a N e um parâmetro – p, que estipula a condição de parada do método – p iterações consecutivas sem melhora no limite inferior. Para a atualização do passo λ foi utilizada a metodologia implementada por Caprara e

outros [CFT99]. O procedimento por eles adotado apresenta a seguinte motivação: quando as seqüências de limites inferiores geradas nas últimas p iterações pouco variam, uma melhora no mesmo pode ser obtida aumentando o passo na direção de crescimento da função a ser minimizada. Por outro lado, quando esses valores estão muito distantes uns dos outros, a diminuição do passo pode contribuir para a estabilização do limite inferior e posterior convergência do algoritmo.

O método subgradiente clássico, na maioria das vezes, antes de alcançar o ponto ótimo, produz uma seqüência de $L(u)'s$ que oscila demasiadamente. O método subgradiente feixe tenta, através da utilização da combinação ponderada de vetores subgradiente de iterações anteriores, superar esse problema. O algoritmo 4 estende o método subgradiente do algoritmo 3 e apresenta o método subgradiente feixe.

Algoritmo 4: Algoritmo de Otimização pelo Método Subgradiente feixe

Entrada: UB, M, N, c, passo inicial $\lambda \geq 0$, $p > 0$ igual ao número limite de iterações do método.

Saída: u^*, vetor de multiplicadores lagrangeanos ótimo ou próximo do ótimo e $L(u^*)$ o limite inferior para o problema

$k \leftarrow 0$

Obtenha u^k através da fórmula $u_i^k = min_{j \in Ji} \frac{c_j}{|I_j|}$, $i \in M$.

$\lambda \leftarrow 0.1$

$L(u^k) \leftarrow \infty$

$s(u_i^k) \leftarrow \infty$, $\forall i \in M$

enquanto *nas últimas p iterações, houver melhora no valor de* $L(u)$ **ou** $UB <> L(u^k)$ **ou** $s(u^k) <> 0$ **faça**

  Calcule o vetor de custos reduzidos da iteração k, $c_j^k(u)$, através da equação (3.3).

  Calcule a solução do problema da iteração k para todo $j \in N$

$$x_j^k = \begin{cases} 0 & se\ c_j^k(u) > 0 \\ 0\ ou\ 1 & se\ c_j^k(u) = 0 \\ 1 & se\ c_j^k(u) < 0 \end{cases}$$

  Determine o valor do limite inferior da iteração k, $L^k(u)$, através da equação (3.1)

  Atualize o melhor valor encontrado para o limite inferior $L(u^*)$ e (u^*) caso seja necessário.

  Calcule o vetor subgradiente da iteração k, s^k, por meio da equação (3.4)

  $s^k \leftarrow 0.6s^k + 0.2s^{k-1} + 0.1s^{k-2} + 0.1s^{k-3}$

  Calcule o vetor de multiplicadores de Lagrange da iteração $k+1$, u^{k+1}, por meio de (3.5)

  Se o melhor e o pior $L(u)$ das últimas p iterações diferem-se por mais de 1% divida o valor de λ por 2. Caso contrário, sendo essa diferença menor que 0.1%, multiplique λ por 2

  $k = k + 1$

fim

Retorne u^* e $L(u^*)$.

3.3 Algoritmo Híbrido (HHS)

Nessa seção é apresentada uma heurística – HHS que tem como base a Relaxação Lagrangeana e o Método de Otimização simplex para obtenção de soluções viáveis para o *PR*. Essa heurística apresenta três fases que são iterativamente executadas:

- **Fase do Subgradiente (*FSB*):** essa fase é responsável pela obtenção de limites inferiores e multiplicadores de lagrange a serem utilizados na caracterização de quão importante é cada uma das colunas do problema na solução ótima. Essas informações serão utilizadas na fase seguinte da heurística implementada.

- **Fase Seleção de Colunas (*FSC*):** a obtenção de soluções heurísticas ou exatas, muitas vezes, são facilitadas quando o espaço de solução do problema a ser resolvido é reduzido. Aproveitando-se de tal característica, a heurística HHS utiliza, a cada iteração, um subconjunto não crescente e diferente de colunas – Problema Core (PC). Essa fase é, portanto, responsável por escolher as colunas que irão compôr o PC sejam elas reaproveitadas de iterações anteriores ou selecionadas do subconjunto de colunas que ainda não participou do mesmo.

- **Fase Heurística (*FH*):** dado o subconjunto de colunas selecionado na *FSC*, essa fase foi implementada com o intuito de obter soluções viáveis para o *PR*. Para tal, uma abordagem híbrida, que conjuga um algoritmo lagrangeano e o Método de Otimização simplex é utilizada.

Na seção seguinte (3.3.1) terminologias importantes para a compreensão da heurística *HHS* são introduzidas. Além disso, uma descrição de suas fases também é apresentada.

3.3.1 Terminologias utilizadas e Idéia Geral do Algoritmo

Como definido na seção 2.1, n representa o número de colunas do problema a ser resolvido. A todo este conjunto, $N = 1, 2, ..., n$, chamamos de colunas do repositório – R.

Cada coluna $j \in N$, contém seu conjunto próprio de atributos que pode tanto ser obtido como dado de entrada do problema – custo real de uma coluna, quanto determinado no transcorrer da execução do algoritmo – custo reduzido lagrangeano, determinado após a execução do método subgradiente feixe; custo reduzido do simplex, inicializado ou atualizado a cada vez que uma coluna participa do subconjunto PC de colunas para a qual se deseja obter uma solução viável.

O algoritmo aqui apresentado, após determinar um vetor de multiplicadores próximo do ótimo por meio do método subgradiente feixe (FSB), seleciona colunas de R e reaproveita outras de iterações anteriores com o objetivo de compor, a cada iteração, um novo conjunto de colunas a fazer parte do PC (FSC).

Essa seleção é efetuada escolhendo-se no máximo t colunas de R e reaproveitando-se k dentre aquelas que já participaram, pelo menos uma vez, do PC. Assim, sendo nc

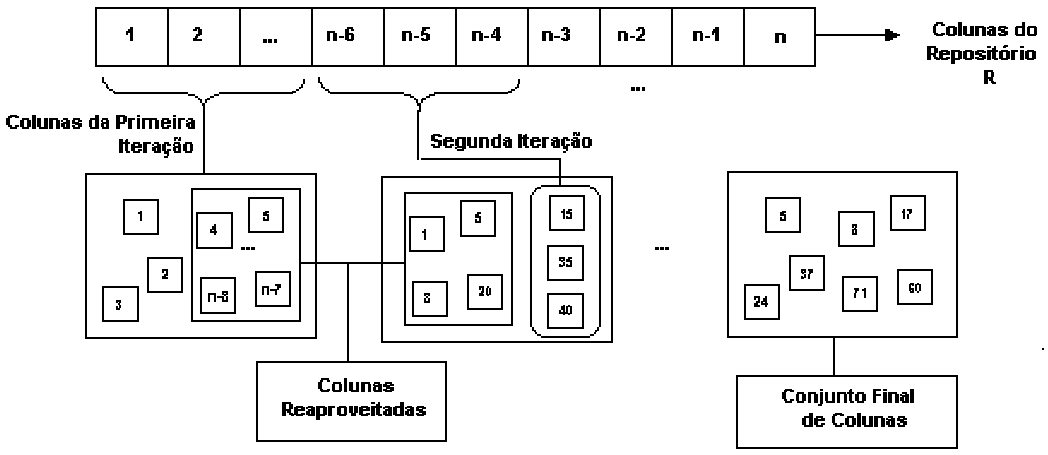

Figura 3.1: Resumo esquemático da Fase Seleção de Colunas do HHS

uma constante definida como o número de colunas a estarem presentes no PC a partir da segunda iteração do algoritmo (o valor para essa constante depende de cada instância testada e será definida na seção 5), pode-se definir:

$$\text{número de iterações do algoritmo } (iter) = \begin{cases} n \; div \; t & \text{, se } n \; mod \; t = 0 \\ (n \; div \; t) + 1 & \text{, caso contrário.} \end{cases} \tag{3.6}$$

Selecionadas as colunas a fazerem parte do PC, as mesmas são submetidas à *FH*. Essa fase, que conjuga o método de otimização simplex e informações provenientes do vetor de multiplicadores próximo do ótimo calculado para o subproblema em questão, objetiva obter soluções viáveis para o PC em questão.

A figura 3.1 ilustra a dinâmica utilizada para a seleção de colunas na *FSC*. Nela, R representa o conjunto de colunas do repositório que, em um momento anterior à primeira iteração do HHS, coincide com o conjunto N. Na primeira iteração do algoritmo, um subconjunto de colunas é retirada de R (não mais pertencerá a esse conjunto), passando essas a fazer parte do PC (vide rótulo "Colunas da Primeira Iteração"na figura). Esse conjunto de colunas passará pelas três fases do algoritmo. Assim, desse conjunto, algumas colunas serão reaproveitadas e novas colunas serão extraídas de R. Os passos anteriores são repetidos até que o conjunto R esteja vazio.

As sub-seções seguintes detalham cada uma das três fases da *HHS*.

3.3.1.1 Fase do Subgradiente *(FSB)*

Apresentando como entrada as colunas e linhas do problema ou subproblema a ser resolvido, essa fase objetiva, além de encontrar um limite inferior para o conjunto de colunas sub-

metido como entrada, obter um vetor de multiplicadores lagrangeanos próximo do ótimo. Esse vetor, segundo Caprara e outros [CFT99], quando utilizado para caracterizar o custo reduzido de uma coluna (3.3), faz desse uma métrica de "qualidade"bem mais confiável que o próprio custo real de uma coluna.

Algoritmo 5: Fase Subgradiente

1 **Entrada:** linhas, colunas e custos reais das colunas do problema a ser resolvido
2 **Saída:** vetor de multiplicadores ótimo ou próximo do ótimo, limite inferior do problema ou subproblema a ser considerado
3 **Procedimento:** Executar o método subgradiente feixe (algoritmo 4) para o conjunto de colunas submetido como entrada para a fase corrente. No algoritmo em questão essa fase será executada levando em consideração todas as colunas do problema uma única vez. Nesse caso, o limite inferior encontrado será o global. Em outras iterações em que a mesma for executada apenas um subconjunto de colunas do problema original será submetido como entrada para a fase FSB.

3.3.1.2 Fase Seleção de Colunas *(FSC)*

Essa fase pretende, a cada iteração do algoritmo, determinar um novo conjunto de colunas a ser utilizado na *FH*. Seja u^* o melhor vetor de multiplicadores lagrangeanos determinado na *FSB* e $c_{simplex}$ um vetor de custos reduzidos inicializado com um valor muito alto para todas as colunas do problema no início do algoritmo e atualizado após a execução do método simplex. A partir de u^* é possível obter, para cada coluna $j \in N$, o seu respectivo custo reduzido $c_{ju}(j)$ por meio de (3.3). Logo, para toda coluna $j \in N$, obtém-se a seguinte fórmula que caracteriza o quão boa é cada coluna para a solução do problema: $c_{LS}(j) = c_{ju}(j) + c_{simplex}(j)$. A inserção da parcela $c_{simplex}(j)$ tem a seguinte motivação: o algoritmo simplex retorna uma solução que apresenta variáveis básicas e não básicas. As colunas da base, assim como as colunas não básicas que produzirão soluções alternativas para o problema de mesmo custo, apresentarão custo reduzido simplex igual a zero. Essas colunas têm alta probabilidade de estarem na solução ótima do PR. O restante terá custo reduzido maior que zero e portanto, pequena tendência a estar na solução ótima do PR.

Assim, na primeira iteração do algoritmo, t colunas são randomicamente retiradas do repositório R passando a compôr o PC. Essas colunas serão submetidas à fase *FH*. Nela, os valores de $c_{simplex}$ serão atualizados para todas as colunas pertencentes ao PC.

Nas $iter - 1$ iterações posteriores, novamente, no máximo t colunas serão randomicamente retiradas do repositório e irão compor o novo PC. Entretanto, k colunas serão reaproveitadas e passarão, também, a compor o PC, através da ordenação ascendente do vetor c_{LS} e posterior seleção das k primeiras melhores colunas dentre aquelas presentes no mesmo.

Algoritmo 6: Fase Seleção de Colunas

Entrada: PC e R
Saída: PC atualizado
Procedimento: Selecionar t colunas de R e reaproveitar k colunas de menor valor de PC por meio da ordenação do vetor c_{LS}.

3.3.1.3 Fase Heurística *FH*

A fase *FH* objetiva encontrar uma solução viável para o subproblema da iteração corrente do algoritmo. Para tal, a seguinte seqüência de passos é executada:

- **Execução do Método Simplex** – as colunas selecionadas na fase *FSC* são integralmente transferidas ao método Simplex que tem sua execução limitada em q segundos. Esse parâmetro depende da instância a ser considerada e terá, na seção 5, seu valor especificado. Além disso, o custo reduzido das colunas pertencentes ao PC são utilizados para atualizar os vetores $c_{simplex}$ e c_{LS}. O resolvedor utilizado foi o Glpk versão 4.1.

- **Heurística Lagrangeana** – se após os q segundos instanciados a melhor solução inteira encontrada não for ótima, a fase *FSB* é novamente executada (nessa situação, apenas as colunas pertencentes ao PC serão levadas em consideração) objetivando a obtenção de um vetor multiplicadores lagrangeanos capaz de caracterizar a importância das colunas pertencentes ao subproblema PC. Esse vetor, por sua vez, é passado como parâmetro a um procedimento guloso com o objetivo de obter soluções de melhor qualidade. Esse procedimento, que é parte daquele proposto por Caprara e outros [CFT99], é descrito no algoritmo 7.

  O algoritmo 9 resume a *FH*.

Algoritmo 7: Procedimento Guloso

Entrada: vetor de multiplicadores lagrangeanos u.
Saída: solução S.
Inicialize M^* com todas as linhas do problema a ser resolvido – M.
Iniciliza $S = \emptyset$ como o cojunto de colunas a fazer parte da solução do problema.
repita
    para cada $j \in N \setminus S$ **faça**
        Calcule o valor da função $Score$ de cada coluna j através do algoritmo 8 e seja $j*$ a coluna com menor score.
        Faça S := S $\cup$ $\{j^*\}$ e M := M $\setminus$ I_{j*}
    fim
até $M^* = \emptyset$;

Algoritmo 8: Procedimento Score

Seja: $\mu_j = |I_j \cap M^*|$
Entrada: vetor de multiplicadores lagrangeanos u, conjunto de linhas ainda não cobertas por alguma coluna em M^*, e coluna j para a qual se deseja determinar o score.
Saída: score $\sigma_j$da coluna j.
Calcule $\gamma_j := c_j - \sum_{i \in I_j \cap M^*} u_i^k$
se $\gamma_j > 0$ **então**
    $\sigma_j = \gamma_j \mu_j$
fim
senão se $\gamma_j < 0$ **então**
    $\sigma_j = \gamma_j . \mu_j$
fim
senão
    $\sigma_j = \infty$
fim

Algoritmo 9: Fase Heurística

Entrada: PC
Saída: Melhor solução atualizada
Procedimento: Executar o algoritmo Simplex limitando a sua execução em q segundos. Atualizar os vetores $c_{simplex}$ e c_{LS}. Se a melhor solução não for encontrada dentro de q segundos, executar a heurística lagrangeana.

As fases *FSC* e *FH* são iteradas até que não mais haja colunas a serem buscadas do repositório. O algoritmo 10 resume todo a heurística *HHS* proposta.

Algoritmo 10: Algoritmo HHS

Inicialize:
$R \leftarrow N$
$u \leftarrow 0$ para as suas m posições
$c_{simplex}(j) \leftarrow 0$ para as suas n posições
$c_{LS}(j) \leftarrow 0$ para as suas n posições
$c_{ju}(j) \leftarrow 0$ para as suas n posições
$UB \leftarrow +\infty$
Início
Executar a fase *FSB* do algoritmo tendo como entrada todas as colunas e linhas da instância em questão.
Inicializar o vetor $c_{ju}(j)$ por meio de (3.3).
Compor o conjunto PC selecionando as t primeiras colunas do conjunto R - fase *FSC*.
Retire as t colunas inseridas em PC de R.
Executar **Procedimento Simplex** (algoritmo 11)
enquanto *há colunas em* R **faça**
    Compor o conjunto PC selecionando as t colunas seguintes do conjunto R e reaproveitando as r colunas de menor valor por meio da ordenação do vetor c_{LS}, *FSC*.
    Retire as t colunas inseridas em PC de R.
    Executar Procedimento Simplex () (algoritmo 11)
fim
Fim

Algoritmo 11: Procedimento Simplex

Executar o algoritmo Simplex por q segundos.
Atualizar o vetor $c_{simplex}$ para as colunas pertencentes ao PC.
Atualizar o vetor c_{LS} para as colunas pertencentes ao PC utilizando a fórmula $c_{LS}(j) = c_{ju}(j) + c_{simplex}(j)$.
se *após q segundos a solução ótima para o PC não for encontrada* **então**
    Executar a fase *FSB* do algoritmo tendo como entrada as colunas pertencentes ao PC.
    Executar a fase *FH* do algoritmo na tentativa de melhorar o valor de UB.
fim

Capítulo 4

Um Algoritmo Genético Híbrido para o Problema de Recobrimento

4.1 Introdução

Algoritmos Genéticos (AG) são métodos adaptativos introduzidos por Holland [Hol75] e extensamente utilizados na resolução de problemas tais como escalonamento [BC96c, Col95, Fan94, WW95], modelagem de estruturas biológicas [JZ00], planejamento de redes óticas especializadas [SPM99, CGKK97, KMC98], otimização de plantão médico hospitalar [AMA94]. Baseados na teoria da seleção natural apresentada por *Charles Darwin* em 1858, eles apresentam métodos e operadores que, analogamente à evolução de organismos biológicos, possibilitam a obtenção de soluções para problemas reais.

As principais características dos algoritmos genéticos são:

1. **paralelismo explícito:** o alto grau de paralelismo intrínseco aos AGs pode ser facilmente verificado se considerarmos o fato de que cada indivíduo da população existe como um ente isolado e é avaliado de forma independente.

2. **busca estocástica:** ao contrário de outros métodos de busca de valores ótimos, os algoritmos genéticos não apresentam um comportamento determinístico. No entanto, afirmar que tal busca se dá de forma completamente aleatória é incorreto. Afinal, o fator aleatório opera no nível local da busca servindo como orientação para o comportamento do algoritmo.

3. **busca cega:** um algoritmo genético tradicional opera ignorando o significado das estruturas que manipula e qual a melhor maneira de trabalhar sobre estas. Tal característica lhe confere o atributo de não se valer de conhecimento específico ao domínio do problema, o que lhe traz generalidade por um lado, mas uma tendência a uma menor eficiência por outro.

4. **eficiência mediana:** por constituir um método de busca cega, um algoritmo genético tradicional tende a apresentar um desempenho menos adequado que alguns tipos de busca heurística orientadas ao problema. Para resolver tal desvantagem, a tática mais utilizada é a hibridação, onde heurísticas provenientes de outras técnicas são incorporadas.

5. **paralelismo implícito:** ao fazer uma busca por populações, a evolução de um algoritmo genético tende a favorecer indivíduos que compartilhem determinadas características, sendo assim capaz de avaliar implicitamente determinadas combinações ou esquemas como mais ou menos desejáveis, efetuando o que chamamos uma busca por hiperplanos, de natureza paralela.

Resumidamente, podemos dizer que, AG's empregam uma estratégia de busca paralela e aleatória, que é voltada em direção ao reforço de pontos de "alta aptidão", ou seja, pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos). Apesar de aleatórios, eles não executam caminhadas aleatórias não direcionadas, pois exploram informações históricas para encontrar novos pontos de busca onde são esperados melhores desempenhos.

4.2 Algoritmo Genético Tradicional

AG's trabalham com uma população de indivíduos que representam soluções para um dado problema. A cada indivíduo é assinalado um *score* ou função de avaliação (FV) que caracteriza o quão boa é a sua solução para o problema. Assim, aos indivíduos que apresentam *score* alto são dadas maiores oportunidades de reprodução. Essa reprodução, por sua vez, produz filhos que compartilham de algumas características com seus pais. Em contrapartida, membros de uma população cujo *score* é baixo estão menos sujeitos a serem selecionados para reprodução e, por conseqüência, a propagarem seus genes.

Algoritmo 12: Descrição geral de um algoritmo genético

```
1  gere população inicial;
2  calcule a FV para cada indivíduo;
3  finished ← false;
4  enquanto finished = false faça
5  |  para i = 1 até tamanhoPopulação/2 faça
6  |  |  selecione dois indivíduos da geração atual para cruzamento;
7  |  |  recombine os dois indivíduos anteriormente selecionados para gerar filhos;
8  |  |  calcule o score de cada um dos dois filhos gerados;
9  |  |  insira os filhos na nova geração;
10 |  fim
11 |  se população convergiu então
12 |  |  finished ← true;
13 |  fim
14 fim
```

A aplicação de AG's na resolução de problemas requer uma codificação ou representação satisfatória do mesmo, a definição de uma função de avaliação capaz de caracterizar o quão boa é cada uma das soluções geradas, a escolha de mecanismos de reprodução e recombinação para a criação de novas gerações e, finalmente, a escolha de um modelo para substituição de populações. Esses requisitos podem ser sumarizadas da seguinte forma:

- **Codificação**: É assumido que uma solução potencial para um problema deve ser representada por uma série de parâmetros. Esses parâmetros, conhecidos como genes, são agrupados para formar uma cadeia de valores - cromossomos.

  Para o *PR* há duas abordagens que têm sido bastante utilizadas: codificação com alfabeto binário e não-binário. Na primeira abordagem, utilizada em [BC96c, SPU02] cada indivíduo é coficado por meio de uma seqüencia de $\mid N \mid$ bits. Cada bit pode apresentar os valores 0 ou 1. Assim sendo, se o bit de ordem j ($j \in N$) de um indivíduo apresentar o valor 1, a correspondente coluna será utilizada na solução do mesmo. De maneira análoga, apresentando esse bit o valor 0, a coluna correspondente não é utilizada na solução. A abordagem não-binária, introduzida por Eremeev e outros em [Ere98] para o *PR*, codifica um indivíduo por meio de $\mid M \mid$ bits. Cada posição i ($i \in M$) tem como valor uma coluna j tal que $M_j \cap i = \{i\}$.

- **Função de Avaliação**: Cada problema apresenta uma função de avaliação particular. A mesma retorna um valor numérico para cada cromossomo. Esse valor é, supostamente, proporcional à utilidade ou habilidade de um indivíduo.

  Para o *PR* essa função corresponde à própria função objetivo (2.1) a ser minimizada.

- **Reprodução**: Durante a fase reprodutiva de um algoritmo genético, indivíduos da população são selecionados (pais) e recombinados produzindo filhos que irão compor

a próxima geração. Essa seleção, geralmente, tende a privilegiar indivíduos que, após terem seu "potencial"numericamente explicitado pela função de avaliação, apresentarem melhor resultado.

Tendo sido selecionados dois pais, seus cromossomos são recombinados, tipicamente utilizando mecanismos de *crossover* e *mutação*. As formas básicas desses dois operadores funcionam da seguinte forma:

1. **Crossover**: seleciona dois indivíduos de uma população e divide seus cromossomos em uma posição randomicamente escolhida originando 4 segmentos – duas cabeças e duas caudas. Os segmentos de cada cauda são trocados produzindo dois cromossomos completos. Essa técnica gera filhos que herdam características de cada pai. Obviamente, esse operador pode gerar cromossomos que representam soluções inviáveis para o problema considerado. Logo, após a aplicação do crossover, alguma metodologia deve ser aplicada visando viabilizar ou descartar o mesmo.
2. **Mutação**: é aplicada a cada filho individualmente após o crossover. Esse operador altera randomicamente cada gene de um cromossomo com uma probabilidade pequena – um gene em todo o cromossomo.

- **Modelo para substituição de populações:** duas abordagens têm sido tradicionalmente utilizadas para esse propósito. A primeira delas seleciona, a cada geração, apenas dois indivíduos para cruzar. Um único filho, portanto, é gerado. Esse, por sua vez, irá substituir o indivíduo com menor FV na população. Esse modelo é conhecido na literatura como "Modelo de Substituição Incremental". Outro modelo bastante utilizado é chamado "Modelo de Substituição Geracional". Nessa abordagem, a cada geração, uma nova população de filhos é gerada e a população de pais é totalmente substituída. Beasley & Chu, em [BC96c] afirmam, baseado em dados experimentais que, na primeira abordagem, a população tende a convergir mais rapidamente pois, a cada geração, a melhor solução é sempre mantida. Em contrapartida há a desvantagem de se obter convergência prematura – situação na qual a solução ótima global não foi ainda atingida e a população fica "presa"em um mínimo local.

4.3 Algoritmo Genético: intensificação x diversificação

Em um algoritmo genético o balanceamento da taxa de intensificação (TI) de sua população além de ser importante para a sua diversificação, caracteriza também uma estratégia fundamental para se evitar que o conjunto de soluções representado por ela recaia sobre um mínimo local.

A TI caracteriza o quão diversa é uma população em uma dada geração e, por conseqüência, remete à possibilidade de serem gerados filhos que apresentam FV numericamente superior à de seus pais.

Populações cuja TI são muito baixas apresentam indivíduos cujas FV pouco variam. Em contrapartida, TI muito elevadas indicam populações cujos indivíduos estão dispersamente distribuídos dentro do conjunto viável de soluções do problema. Em um AG nenhuma dessas possibilidades são adequadas. A primeira, impossibilita a geração de indivíduos diferentes e, possivelmente, melhor adaptados[1]. Afinal, no cruzamente de indivíduos semelhantes dificilmente serão geradas soluções diversificadas. Populações com características randômicas, por sua vez, não dão indícios de que a mesma convirja.

Um AG tradicional deve adotar medidas que vizem balancear a TI de uma população. Tal objetivo pode ser alcançado quando são criados operadores reprodutivos específicos a serem utilizados em cada uma das situações anteriormente descritas.

Nesse capítulo dois grupos de algoritmos de reprodução foram propostos: um heurístico e outro randômico. Os algoritmos heurísticos são utilizados quando a TI de uma população está sob controle ou elevada. Neles são utilizados métodos que, intuitivamente, levam a geração de indivíduos cujas FV são melhores que a de seus antecedentes. O grupo de algoritmos randômicos, por sua vez, visa diversificar uma população com indivíduos semelhantes evitando, portanto, que a mesma convirja para regiões de mínimos locais. Os mesmos são detalhados na seção seguinte.

4.4 O algoritmo Híbrido (AHGS)

Nessa seção será apresentado um algoritmo iterativo para a resolução do PR. O mesmo utiliza um método heurístico – algoritmo genético (FAG) e um exato – Simplex ($FSXG$), para a obtenção de soluções viáveis. O $AHGS$ apresenta as seguintes características principais:

- Todo o conjunto de colunas do problema N é submetido como entrada ao PR.
- A fase inicial, FPI, gera, randomicamente, uma população inicial para ser utilizada na primeira iteração da fase FAG.
- A fase $FSXG$ é responsável pela melhoria da solução encontrada pela fase FAG e tem sua execução limitada por tempo.
- A fase FAG é responsável pela geração de soluções viáveis para o PR e posterior otimização através de procedimentos de busca na vizinhança. Ela também seleciona um subconjunto de colunas do problema a serem utilizadas na fase $FSXG$.
- O algoritmo genético utiliza o modelo geracional para substituição de populações e codificação binária de cromossomos.

Algoritmos genéticos, da maneira como foram concebidos por Holland em [Hol75], caracterizam uma solução para um dado problema por meio de cromossomos ou indivíduos. Atrelados aos mesmos há uma maneira particular de codificá-los como especificado na seção

[1]Indivíduos melhor adaptados, nesse texto, indicam indivíduos com melhores FV.

4.1. Independente da codificação utilizada, à medida que os indivíduos de uma população passam pelos processos convencionais de reprodução – seleção, cruzamento e mutação, genes que apresentam maior aptidão tendem a permanecer ativos de uma geração para outra. Assim, após um determinado número de iterações, seguindo a propriedade que os algoritmos genéticos têm de preservar informações críticas relativas à solução de um problema, espera-se que uma população convirja e, por conseqüência, o conjunto de genes (colunas) ativos[2] não varie muito de um indivíduo para outro. Essa tendência é explorada no *AHGS*. Ou seja: terminada a execução do *FAG* apenas os genes ativos são submetidos à fase *FSXG*.

O algoritmo 13 resume o *AHGS*. Nas sub-seções que se seguem cada uma das fases serão detalhadas.

Algoritmo 13: Descrição geral do algoritmo AHGS

```
1 Seja itlim o número limite de gerações para o algoritmo genético.
2 Execute a Fase 1 – FPI
3 para i = 1 até itlim faça
4 |  Execute a Fase 2 – FAG
5 fim
6 Execute a Fase 3 – FSXG
```

4.4.1 Fase 1: Geração de População Inicial (*FPI*)

O primeiro passo a ser observado na geração da população inicial de um algoritmo genético é o número de indivíduos que irá compô-la. Essa população deve apresentar uma boa amostragem de todas as colunas do problema. No *AHGS* o tamanho de uma população depende da instância do problema para o qual se deseja obter uma solução (os valores estipulados para esse parâmetro será apresentado na seção 5).

O segundo passo consiste em determinar como essa população será gerada. Para tal, a seguinte abordagem foi utilizada: para cada linha linha $i \in M$, foi selecionada, randomicamente, uma coluna dentro do conjunto J_i.

A população das gerações subseqüentes é dinamicamente atualizada por meio do operador genético de Seleção.

4.4.2 Fase 2: Algoritmo Genético (*FAG*)

Essa fase, recebendo como entrada as n colunas do problema a ser resolvido, gera soluções viáveis para o *PR* através de um algoritmo genético e provê informações importantes para a execução da *FSXG*.

[2]um gene é considerado ativo quando a coluna representada por ele está na solução do *PR*.

O algoritmo genético implementado segue a descrição apresentada no algoritmo 12 com condição de convergência limitada pelo número de iterações cujo valor depende da instância a ser trabalhada.

Os operadores reprodutivos implementados dividem-se em três grupos: heurísticos, randômicos e híbridos. Os operadores heurísticos são utilizados quando a TI da população está alta. Operadores randômicos, por sua vez, são executados quando a TI da população está baixa. O único operador híbrido implementado é o de seleção. Ele é sempre executado independente da TI da população.

Os conceitos associados a uma TI alta ou baixa foram empiricamente estipulados e podem ser definidos da seguinte maneira: Seja P o conjunto de indivíduos de uma população em uma geração. Me e DP a média e o desvio padrão em relação à média respectivamente. Se DP é menor que 8% do valor de Me então a TI da população é dita alta e os operadores randômicos são aplicados à P. Caso contrário a TI da população é dita baixa e os operadores heurísticos serão aplicados.

A seguir cada um dos operadores são apresentados seguidos de sua implementação heurística, randômica e híbrida, quando for o caso.

4.4.2.1 Operador de Seleção

O operador de seleção objetiva gerar uma população intermediária a partir da população corrente que, posteriormente, irá passar pelos processos reprodutivos.

Para implementação desse operador a seguinte metodologia foi utilizada:

1. O indivíduo da população que apresenta o menor valor de sua FV é diretamente copiado para a população da próxima geração. Ou seja, de uma geração para outra, o indivíduo de menor FV é sempre mantido.

2. Seja f_i o valor da FV de um indivíduo $i \in P$ e $\bar{f}$ o valor de $(\sum_{i \in P} f_i)/|P|$. Para cada indivíduo $i \in P$ em que $\bar{f}/f_i$ é maior que 1.0, a parte inteira desse número indica o número de cópias do mesmo que serão diretamente colocadas na população intermediária. Para todos os indivíduos, incluindo aqueles cujo valor de $\bar{f}/f_i$ forem menor que 1.0, uma cópia adicional do mesmo será colocada na população intermediária com probabilidade correspondente à parte fracionária de $\bar{f}/f_i$. Exemplo: se um indivíduo apresenta $\bar{f}/f_i = 1.78$ então uma cópia do mesmo será colocada na população intermediária. Além disso, o mesmo receberá outra cópia adicional com uma chance de 0.78. Um indivíduo cujo valor de $\bar{f}/f_i = 0.38$ terá uma chance de 0.38 de uma cópia dele estar na população intermediária.

3. Utilizando o passo acima, não há garantia quanto ao número de indivíduos a existir na população intermediária. Assim, se o número final gerado for ímpar, o indivíduo de menor FV tem uma cópia adicional inserida na população intermediária. Esse procedimento justifica-se porque o cruzamento sempre ocorre entre pares de indivíduos.

4. Gerada a população intermediária, é necessário parear os indivíduos que irão cruzar. Esse pareamento foi efetuado randomicamente.

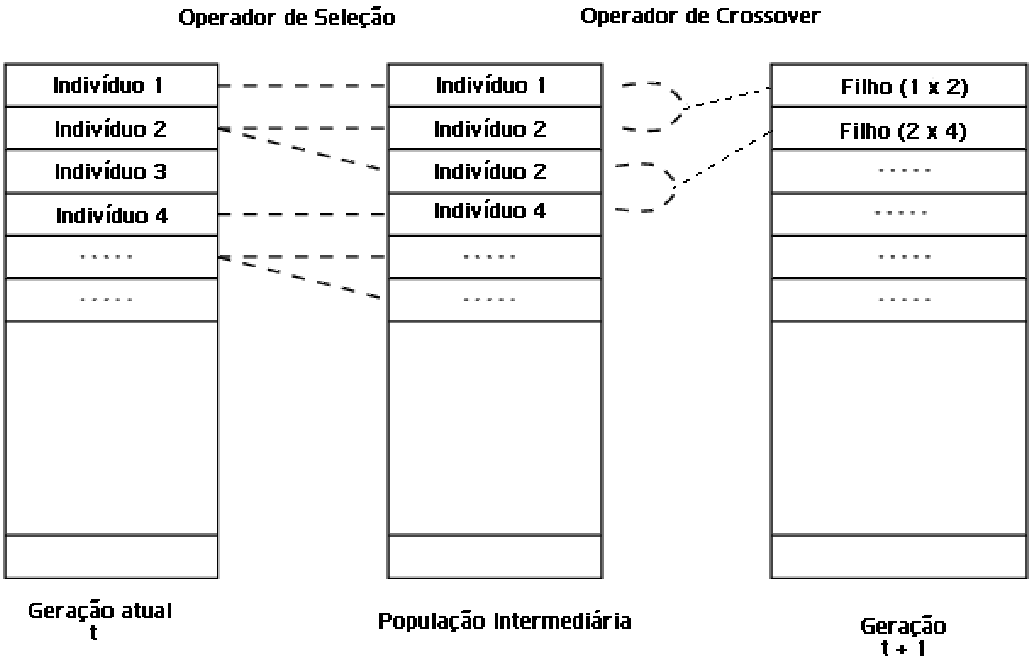

Figura 4.1: Operador de Seleção

A figura 4.1 ilustra o processo de preparação da população intermediária para cruzamento.

4.4.2.2 Operador de Crossover

Foram implementados dois algoritmos de *crossover*: randômico e heurístico. Os mesmos podem ser da seguinte maneira descritos:

1. **Crossover randômico**: utilizou-se uma variação do operador tradicional de crossover proposta por Beasley em [BC96c] – *CrossOver Fusion*. Nela, filhos são gerados da seguinte maneira: sejam f_{p1} e f_{p2} o valor das funções de avaliação de P_1 e P_2 selecionados durante o processo de seleção, e C a string a ser gerada para o filho a partir dos pais. Assumindo um problema de minimização, para todo $j \in N$:

   (a) se $P_1[j] = P_2[j]$, então $C[j] = P_1[j] = P_2[j]$,

   (b) se $P_1[j] \neq P_2[j]$, então:

   - $C[j] := P_1[j]$ com probabilidade $p = f_{p2}/(f_{p1} + f_{p2})$,
   - $C[j] := P_2[j]$ com probabilidade $1 - p$.

2. **Crossover heurístico**: Sejam P_1 e P_2 dois indivíduos selecionados para cruzamento, $P'_1 = \{i \in N / P_1[i] = 1\}$, $P'_2 = \{i \in N / P_2[i] = 1\}$, $Q = P'_1 \cup P'_2$, $M' \leftarrow M$ e S, de tamanho n o filho a ser gerado após a aplicação do operador.

   Define-se também o vetor mj, de tamanho n, para cada coluna $j \in Q$ como $|M_j|$ e o vetor *score*, também de tamanho n, responsável por estipular um valor que irá caracterizar a "importância"de uma determinada coluna para a composição de uma solução S para o problema. A determinação desse valor é efetuada por meio de uma função – $DeterminaScoreColuna$ que recebe como entrada três parâmetros: o vetor m_j o índice da coluna para a qual se deseja estipular um valor e um número r randomicamente escolhido entre 1 e 5. O valor número r seleciona uma das equações (2.6) - (2.10) apresentadas na seção 2.3 a ser utilizada para a determinação da "utilidade"de uma coluna. O algoritmo 14 resume esse operador.

Algoritmo 14: Crossover Heurístico

Inicialize:
$P'_1 = \{i \in N / P_1[i] = 1\}$
$P'_2 = \{i \in N / P_2[i] = 1\}$
$Q = P'_1 \cup P'_2$
$M' \leftarrow M$
$S \leftarrow 0$ para j de 1 até n
$r \leftarrow$ número randomicamente escolhido entre 1 e 5.
para cada $j \in N$ **faça**
    se $j \in Q$ **então**
        $m_j \leftarrow |M_j|$
    fim
    senão
        $m_j \leftarrow \infty$
    fim
fim
enquanto $M' \neq \emptyset$ **faça**
    para cada $j \in Q$ **faça**
        se $(mj[j] > 0)$ **então**
            $score[j] \leftarrow DeterminaScoreColuna(mj, j, r)$
        fim
        senão
            $score[j] \leftarrow \infty$
        fim
        Ordene vetor *score*.
        w = menorValorOrdenadoVetorScore().
        $M' \leftarrow M' \setminus M_w$
        $mj[w] \leftarrow mj[w] - 1$
        $S[w] = 1$
    fim
fim

4.4.2.3 Operador de Mutação

1. **Mutação randômica**: Esse operador é utilizado apenas quando o *crossover* randômico for aplicado na geração de um filho S. Dado o vetor S esse operador muta cada uma das posições do mesmo com probabilidade igual $1/n$. Ou seja, em média, uma posição

do vetor S terá seu valor trocado de 0 para 1 ou de 1 para 0.

2. **Mutação heurística**: Seja F e D os cojuntos compostos pelas colunas i pertencentes a N cujo valor $S[i] = 0$ e $S[i] = 1$, respectivamente. Esse operador vai mudar um ou mais valores do vetor S apenas se melhores soluções puderem ser alcançadas com esse procedimento. Foram implementados 4 operadores de mutação heurística. Os mesmos só serão utilizados quando o operador *crossover* heurístico for aplicado no cruzamento.

   (a) **OtimizacaoPorSubstituicaoDeUmaColuna ():** Se existe uma coluna $j \in F$ e uma coluna $k \in D$ tal que $c[j] < c[k]$ e, além disso, a retirada de k de S e inserção de j em S não afetam as condições de viabilidade do PR, então faça $S[j] := 1$ e $S[k] := 0$.

   (b) **OtimizacaoPorSubstituicaoDeDuasColunas ():** Se existe uma coluna $j \in F$ e duas colunas $(k, l) \in D$ tal que $c[j] < c[k] + c[l]$ e, além disso, a retirada de k e l de S e a inserção de j em S não afetam as condições de viabilidade do PR então faça $S[j] := 1$, $S[k] := 0$ e $S[l] := 0$.

   (c) **OtimizacaoPorSubstituicaoDeTresColunas ():** Se existe uma coluna $j \in F$ e três colunas $(k, l, m) \in D$ tal que $c[j] < c[k] + c[l] + c[m]$ e, além disso, a retirada de k, l e m de S e inserção de j em S não afetam as condições de viabilidade do PR então faça $S[j] := 1$, $S[k] := 0, S[l] := 0$ e $S[m] := 0$.

   (d) **OtimizacaoPorSubstituicaoDeDuasColunasDeForaPorUmaDeDentro():** Se existem duas colunas j e $l \in F$ e uma coluna $k \in D$ tal que $c[j]+c[l] < c[k]$ e, além disso, a retirada de k de S e inserção de j e l em S não afetam as condições de viabilidade do PR então faça $S[j] = 1, S[l] = 1$ e $S[k] = 0$.

4.4.2.4 Operador de Viabilização de Soluções

A aplicação dos operadores crossover e mutação randômicos não garantem a viabilidade de uma solução S. Várias abordagens têm sido utilizadas na literatura: descartar a solução inviável, viabilizá-la ou mantê-la dentro da população de indivíduos penalizando-a. No $AGHS$ a viabilização de S foi utilizada. O algoritmo 15 resume esse operador.

Dado M', um vetor de tamanho m, as linhas 1-12 do algoritmo 15 determinam o conjunto de linhas cobertas pelas colunas de S. Essas linhas terão suas respectivas posições inicializadas em 1, enquanto as restantes terão o valor 0. Para cada linha i que não é coberta – $M'[i] = 0$, uma coluna é randomicamente escolhida dentre aquelas pertencentes ao conjunto M_i (linhas 13-17). Posteriormente, o vetor M', é atualizado.

Algoritmo 15: Operador de viabilização de soluções

Sejam: S o vetor solução que passou pelos operadores de crossover e mutação randômicos.
M' vetor de tamanho $|M|$.
para *i := 1 até m* **faça**
    $M'[i] \leftarrow 0$
fim
para *j := 1 até n* **faça**
    se $S[j] = 1$ **então**
        para cada $linha \in M_j$ **faça**
            $M'[linha] \leftarrow 1$
        fim
    fim
fim
para *i := 1 até m* **faça**
    se $M'[i] = 0$ **então**
        Seleciona randomicamente uma coluna w dentre aquelas pertencentes a J_i.
        $S[w] \leftarrow 1$
        para cada $linha \in M_w$ **faça**
            $M'[linha] \leftarrow 1$
        fim
    fim
fim

4.4.3 Fase 3: Aplicação do Algoritmo Simplex (*FSXG*)

A fase 2 é executada por um número previamente estipulado de iterações submetido como entrada ao *PR*. Após as mesmas, toda a população da geração atual é inspecionada e, das n colunas submetidas ao problema, formar-se-á um subconjunto PS constituído pelas colunas que fazem parte da solução de pelo menos um indivíduo. As colunas de PS, por sua vez, são integralmente transferidas ao algoritmo simplex que tem sua execução limitada em q segundos. Esse parâmetro depende da instância a ser considerada e terá, mais a frente, seu valor especificado. O resolvedor utilizado foi o Glpk versão 4.1.

Capítulo 5

Resultados Computacionais

Nesta seção serão apresentados os resultados computacionais obtidos ao submeter instâncias reais e da literatura [Lib] ao algoritmo *AHGS* apresentado na seção 4.4 e ao algoritmo *HHS* apresentado na seção 3.

Os algoritmos foram desenvolvidos utilizando a linguagem de programação C em ambiente Linux. Os testes foram efetuados em um Pentium IV 2.4 GHz com 500 MB de memória RAM.

5.1 Característica das Instâncias Testadas

A seguir serão explicitadas as características das instâncias utilizadas para testar os algoritmos implementados.

A tabela 5.1 caracteriza as instâncias que podem ser encontradas na biblioteca mantida por Beasley [Lib] para *benchmark* de algoritmos. Grande parte dos algoritmos que tratam do *PR*, desde 1990 – ano em que essas instâncias foram publicadas [Bea90], as têm utilizado. Nessa tabela, *Tamanho* e *Densidade* são, respectivamente as dimensões e densidade[1] da matriz $m \times n$ submetida como entrada ao algoritmo e *Custo* corresponde à faixa de valores que o custo de cada cada coluna do problema pode assumir.

As tabelas 5.2 e 5.3 apresentam as características das instâncias de alocação de tripulações aéreas de Wedelin [Wed95a] e as instâncias reais de Balas e Carrera [BC96a] para redes de transporte aéreo – classe AA e rodoviário – classe BUS, respectivamente.

Enquanto as instâncias providas por Beasley e Balas e Carreras trabalham com custos moderados – 1 a 100 para aquelas e 35 a 3.619 para essas, as instâncias de Wedelin trabalham com custos que podem atingir valores de até 7 dígitos – o que acrescenta um complicante a mais para o *PR*.

[1] número de 1s existentes na matriz $m \times n$ dividido pelo número de elementos existente na mesma

Teste	Tamanho	Densidade(%)	Custo
4.1	200 x 1.000	2	1-100
4.2	200 x 1.000	2	1-100
4.3	200 x 1.000	2	1-100
4.4	200 x 1.000	2	1-100
4.5	200 x 1.000	2	1-100
4.6	200 x 1.000	2	1-100
4.7	200 x 1.000	2	1-100
4.8	200 x 1.000	2	1-100
4.9	200 x 1.000	2	1-100
4.10	200 x 1.000	2	1-100
5.1	200 x 2.000	5	1-100
5.2	200 x 2.000	5	1-100
5.3	200 x 2.000	5	1-100
5.4	200 x 2.000	5	1-100
5.5	200 x 2.000	5	1-100
5.6	200 x 2.000	5	1-100
5.7	200 x 2.000	5	1-100
5.8	200 x 2.000	5	1-100
5.9	200 x 2.000	5	1-100
5.10	200 x 2.000	5	1-100
6	200 x 1.000	2	1-100
A	300 x 3.000	2	1-100
B	300 x 3.000	5	1-100
C	400 x 4.000	2	1-100
D	400 x 4.000	5	1-100
E	500 x 5.000	10	1-100
F	500 x 5.000	20	1-100
G	1.000 x 10.000	2	1-100
H	1.000 x 10.000	5	1-100

Tabela 5.1: Característica das instâncias de OR-Library.

Classe	Tamanho	Densidade(%)	Custo
B727scratch	29 x 157	8.2	1.600-11.850
ALITALIA	118 x 1.165	3.1	2.200-2.110.900
A320	199 x 6.931	2.3	1.600-2.111.450
A320coc	235 x 18.753	1.9	1.900-1.812.000
SASjump	742 x 10.370	0.6	4.720-55.849
SASjump2	1.366 x 25.032	0.3	3.860-35.200

Tabela 5.2: Característica das instâncias de Wedelin.

Classe	Tamanho	Densidade(%)	Custo
AA03	106 x 8.661	4,05	91-3.619
AA04	106 x 8.002	4,05	91-3.619
AA05	105 x 7.435	4,05	91-3.619
AA06	105 x 6.951	4,11	91-3.619
AA11	271 x 4.413	2,53	35-2.966
AA12	272 x 4.208	2,52	35-2.966
AA13	265 x 4.025	2,60	35-2.966
AA14	266 x 3.868	2,50	35-2.966
AA15	267 x 3.701	2,58	35-2.966
AA16	265 x 3.558	2,63	35-2.966
AA17	264 x 3.425	2,61	35-2.966
AA18	271 x 3.314	2,55	35-2.966
AA19	263 x 3.202	2,63	35-2.966
AA20	269 x 3.095	2,58	35-2.966
BUS1	454 x 2.241	1,89	120-877
BUS2	681 x 9.524	0,51	120-576

Tabela 5.3: Característica das instâncias de Balas e Carrera.

Classe	Num. Col. Iter.	Num. Col. Reap. (%)	Tempo (s)
4	1.000	0	30
5	2.000	0	30
6	1.000	0	30
A	3.000	0	60
B	3.000	0	60
C	4.000	0	60
D	4.000	0	60
E	3.000	0,1	300
F	5.000	0,3	300
G	2.000	0,1	300
H	2.000	0,1	300

Tabela 5.4: Parâmetros do HHS para as instâncias OR-LIBRARY.

5.2 Parâmetros utilizados nos algoritmos implementados

5.2.1 Parâmetros utilizados no HHS

As tabelas 5.4, 5.5 e 5.6 especificam os valores dos parâmetros utilizados para cada classe de problemas testada quando os problemas pertencentes a ela são submetidos como entrada ao algoritmo *HHS* (esses parâmetros foram determinados empiracamente). Nela, *Num.Col.Iter* é o número máximo de colunas a estar presente em cada iteração do *HHS*; *Num.Col.Reap.*, por sua vez, representa a percentagem de colunas a serem reaproveitadas da iteração anterior. Esse valor é calculado tendo como base aqueles especificados na coluna *Num.Col.Iter*. Nos campos em que o número de colunas por iteração é igual ao número de colunas da instância, o campo *Num. Col. Reap.* apresenta valor igual a zero. A coluna *Tempo*, por fim, estipula um limite de tempo, em segundos, para a execução do algoritmo Simplex.

5.2.2 Parâmetros do AHGS

Como especificado na seção 4.4, o algoritmo AHGS recebe como entrada um conjunto de parâmetros. Alguns desses parâmetros são comumente utilizados na implementação de um algoritmo genético convencional. Outros fazem parte da própria estrutura algorítmica

Classe	Num. Col. Iter.	Num. Col. Reap. (%)	Tempo (s)
B727scratch	157	0	20
ALITALIA	1.165	0	20
A320	6.931	0	20
A320coc	18.753	0	20
SASjump	10.370	0	200
SASjump2	25.032	0	200

Tabela 5.5: Parâmetros do HHS para as instâncias de Wedelin.

Classe	Num. Col. Iter.	Num. Col. Reap. (%)	Tempo (s)
AA03	8.661	0	20
AA04	8.002	0	20
AA05	7.435	0	20
AA06	6.951	0	20
AA11	4.413	0	30
AA12	4.208	0	30
AA13	4.025	0	30
AA14	3.868	0	30
AA15	3.701	0	30
AA16	3.558	0	30
AA17	3.425	0	30
AA18	3.314	0	30
AA19	3.202	0	30
AA20	3.095	0	30
BUS1	2.241	0	30
BUS2	9.524	0	30

Tabela 5.6: Parâmetros do HHS para as instâncias de Balas e Carrera.

Classe	População	Num. Ger.	Tempo Lim. (s)
4	50	2	100
5	50	2	100
6	50	2	100
A	60	2	100
B	60	2	100
C	60	2	100
D	60	2	100
E	60	5	500
F	60	5	500
G	60	5	500
H	60	5	500
Wedelin	30	3	400
Balas	30	2	500

Tabela 5.7: Parâmetros do AHGS para as instâncias testadas.

desenvolvida.

A tabela 5.7 especifica valores de parâmetros utilizados para cada classe de problemas testada. Nela, *População* e *Num.Ger.* são, respectivamente, o número de indivíduos a fazer parte da população inicial do AG – Fase FAG, e a duração dessa fase em número de gerações. *Tempo Lim.*, por sua vez, caracteriza o limite de tempo estipulado, em segundos, para a execução da fase *FSXG*.

5.3 Resultados numéricos

5.3.1 Instâncias OR-LIbrary

As tabelas 5.8 e 5.9 apresentam os resultados computacionais obtidos pelos algoritmo implementados AHGS e HHS . A solução reportada por esses algoritmos é a melhor encontrada dentre as 100 execuções dos mesmos. O tempo computacional presente nas tabelas 5.8, 5.9 e 5.10 é a média aritmética entre as 100 execuções. Os mesmos são comparados com o algoritmo genético de Beasley e Chu [BC96c] – BeCh, com as heurísticas rápidas de Beasley [Bea90] – Be e Balas e Carrera [BC96a] – BaCa, com a bem sucedida heurística lagrangena de Caprara e outros [CFT99] – CFT e, finalmente, com o algoritmo genético paralelo de Solar e outros [SPU02] – Pa. Nela, *Instância* identifica a instância assim como a mesma

pode ser encontrada na literatura, *Ótima* é o valor da solução ótima para a instância considerada, *Sol* e *Tempo* são, respectivamente, os valores das melhores soluções e o tempo envolvido na obtenção desses valores em segundos para cada um dos algoritmos.[2]

A observação dessas tabelas mostra que para instâncias de pequeno porte os algoritmos implementados chegam à solução ótima em todas as instâncias consideradas em um intervalo de tempo pequeno. Comparado ao algoritmo CFT, que apresenta os melhores resultados da literatura, pode-se dizer que os mesmos são competitivos tanto no valor das soluções encontradas quanto no tempo computacional dispendido para se encontrar as mesmas. Comparado ao restante dos algoritmos o AHGS e o HHS apresentam desempenho similar ou superior. Afinal, em algumas das instâncias eles são capazes de obter soluções de melhor qualidade.

A tabela 5.10 apresenta os resultados computacionais obtidos pelos algoritmos AHGS e HHS quando aos mesmos foram submetidas instâncias da literatura para as quais as soluções inteiras ótimas não são conhecidas. A interpretação de seus rótulos é idêntica àquelas apresentadas na tabela 5.8. No entanto, o rótulo *Ótima* foi substituído pelo *MS*.

Os algoritmos desenvolvidos foram capazes de encontrar a melhor solução disponível na literatura para todas as instâncias dessa tabela. Quando comparados com o CFT eles obtiveram desempenho computacional, na média, inferior, porém aceitável – máximo de 4 horas para o AHGS – instância G5 e 1 hora para o HHS – instância G2. Comparado ao restante dos algoritmos ele encontrou soluções de mesmo valor ou de qualidade superior.

As figuras 5.1 e 5.2 mostram um gráfico comparativo entre os tempos de execução dos algoritmos AHGS, HHS e CFT. O gráfico foi plotado em escala logarítmica para que as diferenças entre os valores fossem melhor visualizadas. Mesmo não sendo uma métrica perfeitamente adequada para a comparação de algoritmos, em problemas de otimização, nas situações em que as soluções encontradas são iguais ou pouco variam, essa medida é aceitável. Observando-se os gráficos é possível notar que para as instâncias de A a D, na maioria das vezes, o CFT acha a solução ótima para a instância testada com um tempo maior que os algoritmos AHGS e HHS. Comparando-se os algoritmos AHGS e HHS nota-se que o primeiro apresenta comportamento mais estável e, na maioria das vezes, leva um menor intervalo de tempo para alcançar as soluções ótimas ou melhores soluções disponíveis. Considerando-se as instâncias de E a H, a situação anteriormente observada é invertida. Ou seja, o algoritmo CFT passa a apresentar os menores tempos para alcançar as soluções melhores soluções de cada instância sendo precedido pelo HHS e AHGS nessa ordem.

[2]Os rótulos *o* e # caracterizam os campos das tabelas 5.8 , 5.9 e 5.10 em que, respectivamente, os dados não foram disponibilizados pelo artigo em questão e as intâncias não foram disponibilizadas até a publicação do artigo. O algoritmo CFT foi testado utilizando um DECstation 5000/240. Os outros algoritmos tiveram seus tempos computacionais convertidos para o DECstation 500/240 no mesmo artigo em que o CFT é descrito.

Instância	Ótima	AHGS		HHS		BeCh		Be		BaCa		CFT		Par	
		Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo
4.1	**429**	**429**	2,76	**429**	0,72	**429**	294,8	**429**	11,0	**429**	0,8	**429**	2,3	432	*o*
4.2	**512**	**512**	0,24	**512**	0,52	**512**	9,0	**512**	11,1	**512**	1,2	**512**	1,1	517	*o*
4.3	**516**	**516**	2,04	**516**	0,47	**516**	16,4	**516**	6,8	**516**	1,0	**516**	2,1	*o*	*o*
4.4	**494**	**494**	2,46	**494**	0,57	**494**	142,0	495	12,2	**494**	1,7	**494**	9,8	513	*o*
4.5	**512**	**512**	0,24	**512**	0,41	**512**	44,1	**512**	7,0	**512**	0,5	**512**	2,1	*o*	*o*
4.6	**560**	**560**	2,16	**560**	1,26	**560**	16,1	561	15,8	**560**	7,5	**560**	19,3	*o*	*o*
4.7	**430**	**430**	0,26	**430**	1,42	**430**	138,6	130	9,2	**430**	1,0	**430**	2,7	*o*	*o*
4.8	**492**	**492**	2,87	**492**	1,23	**492**	818,7	493	11,5	493	6,4	**492**	22,2	*o*	*o*
4.9	**641**	**641**	3,2	**641**	1,26	**641**	136,1	**641**	20,6	**641**	9,2	**641**	1,8	*o*	*o*
4.10	**514**	**514**	0,24	**514**	1,35	**514**	13,5	**514**	11,9	**514**	1,1	**514**	1,8	*o*	*o*
5.1	**253**	**253**	4,6	**253**	2,9	**253**	42,1	255	17,4	254	11,1	**253**	3,3	271	*o*
5.2	**302**	**302**	8,6	**302**	2,9	**302**	1332,6	304	20,9	307	20,1	**302**	2,3	*o*	*o*
5.3	**226**	**226**	2,9	**226**	1,0	228	11,0	**226**	10,1	**226**	1,0	**226**	2,1	*o*	*o*
5.4	**242**	**242**	4,2	**242**	2,7	**242**	10,1	**242**	11,5	243	11,4	**242**	1,9	*o*	*o*
5.5	**211**	**211**	0,4	**211**	0,9	**211**	14,9	**211**	7,2	**211**	2,1	**211**	1,2	*o*	*o*
5.6	**213**	**213**	4,0	**213**	0,9	**213**	29,9	**213**	11,3	**213**	1,3	**213**	0,9	*o*	*o*
5.7	**293**	**293**	5,0	**293**	2,6	**293**	194,9	294	18,1	**293**	7,5	**293**	15,0	*o*	*o*
5.8	**288**	**288**	5,4	**288**	1,3	**288**	3733,3	**288**	20,7	**288**	4,3	**288**	1,6	*o*	*o*
5.9	**279**	**279**	0,4	**279**	1,0	**279**	13,5	**279**	15,7	**279**	1,5	**279**	2,6	*o*	*o*
5.10	**265**	**265**	0,5	**265**	0,9	**265**	19,2	**265**	9,8	**265**	1,1	**265**	1,3	*o*	*o*
6.1	**138**	**138**	2,1	**138**	5,3	**138**	46,1	141	16,8	140	9,2	**138**	22,6	*O*	*O*
6.2	**146**	**146**	1,8	**146**	4,5	**146**	210,5	**146**	14,5	147	9,2	**146**	17,8	*O*	*O*
6.3	**145**	**145**	1,6	**145**	2,7	**145**	11,8	**145**	15,0	**145**	11,5	**145**	2,3	*O*	*O*
6.4	**131**	**131**	1,83	**131**	3,0	**131**	4,8	**131**	10,3	**131**	8,3	**131**	1,8	*O*	*O*
6.5	**161**	**161**	3,2	**161**	1,6	**161**	12,1	162	13,3	163	10,4	**161**	2,2	*O*	*O*

Tabela 5.8: Resultado das instâncias 4 a 6 testadas provenientes da biblioteca OR-Library.

Instância	Ótima	AHGS		HHS		BeCh		Be		BaCa		CFT		Par	
		Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo
A.1	**253**	**253**	13,7	**253**	21,8	**253**	222,4	255	36,0	258	39,0	**253**	82,0	258	*o*
A.2	**252**	**252**	15,4	**252**	15,8	**252**	327,9	256	44,2	254	40,9	**252**	116,2	*o*	*o*
A.3	**232**	**232**	19,9	**232**	17,1	**232**	127,0	234	28,1	237	28,6	**232**	249,9	*o*	*o*
A.4	**234**	**234**	20,0	**234**	4,4	**234**	45,5	235	33,5	235	36,3	**234**	4,7	*o*	*o*
A.5	**236**	**236**	11,7	**236**	8,2	**236**	23,7	237	19,0	**236**	26,2	**236**	80,0	*o*	*o*
B.1	**69**	**69**	13,1	**69**	121,9	**69**	20,0	70	28,4	**69**	29,0	**69**	4,0	*o*	*o*
B.2	**76**	**76**	26,2	**76**	4,6	**76**	11,6	77	40,8	**76**	29,0	**76**	6,1	*o*	*o*
B.3	**80**	**80**	12,6	**80**	49,9	**80**	709,7	**80**	25,4	81	35,1	**80**	18,0	82	*o*
B.4	**79**	**79**	30,4	**79**	4,6	**79**	29,9	80	37,0	**79**	29,0	**79**	6,3	*o*	*o*
B.5	**72**	**72**	11,9	**72**	4,5	**72**	5,3	**72**	26,0	**72**	32,6	**72**	3,3	*o*	*o*
C.1	**227**	**227**	25,6	**227**	28,3	**227**	187,9	230	42,4	230	116,2	**227**	74,0	*o*	*o*
C.2	**219**	**219**	40,4	**219**	16,0	**219**	40,7	223	66,0	220	56,1	**219**	64,2	230	*o*
C.3	**243**	**243**	60,5	**243**	27,1	**243**	541,3	251	75,1	248	61,7	**243**	70,2	*o*	*o*
C.4	**219**	**219**	72,1	**219**	23,3	**219**	144,6	224	63,4	224	68,1	**219**	61,6	*o*	*o*
C.5	**215**	**215**	36,4	**215**	26,5	**215**	80,6	217	39,9	217	64,6	**215**	60,3	*o*	*o*
D.1	**60**	**60**	26,3	**60**	9,0	**60**	13,8	61	40,9	61	39,0	**60**	82,0	*o*	*o*
D.2	**66**	**66**	40,0	**66**	9,8	**66**	198,6	68	52,7	67	40,9	**66**	116,2	*o*	*o*
D.3	**72**	**72**	65,4	**72**	8,1	**72**	785,3	75	55,8	74	28,6	**72**	249,9	*o*	*o*
D.4	**62**	**62**	33,5	**62**	8,3	**62**	73,5	64	36,5	63	36,3	**62**	4,7	65	*o*
D.5	**61**	**61**	21,3	**61**	7,3	**61**	79,8	62	36,7	**61**	26,2	**61**	80,0	*o*	*o*

Tabela 5.9: Resultado das instâncias A a D testadas provenientes da biblioteca OR-Library.

Instância	MS	AHGS		HHS		BeCh		Be		BaCa		CFT		Par	
		Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo
E.1	**29**	**29**	66,1	**29**	17,9	**29**	35,3	**29**	72,6	**29**	55,3	**29**	26,0	**29**	*o*
E.2	**30**	**30**	3.137,6	**30**	705,8	**30**	343,0	32	92,7	32	76,0	**30**	408,0	*o*	*o*
E.3	**27**	**27**	178,0	**27**	18,4	**27**	2,8360,2	28	92,7	28	80,9	**27**	94,2	*o*	*o*
E.4	**28**	**28**	570,3	**28**	18,7	**28**	539,9	30	100,3	29	77,5	**28**	26,3	*o*	*o*
E.5	**28**	**28**	66,6	**28**	18,9	**28**	35,0	**28**	80,8	**28**	61,6	**28**	36,6	*o*	*o*
F.1	**14**	**14**	130,6	**14**	31,7	**14**	76,4	15	43,9	**14**	67,5	**14**	33,2	*o*	*o*
F.2	**15**	**15**	66,2	**15**	28,7	**15**	78,1	16	102,6	**15**	88,6	**15**	31,2	*o*	*o*
F.3	**14**	**14**	154,9	**14**	326,7	**14**	266,8	15	124,7	15	76,5	**14**	248,5	16	*o*
F.4	**14**	**14**	138,6	**14**	102,5	**14**	209,7	15	118,2	15	74,8	**14**	31,0	*o*	*o*
F.5	**13**	**13**	7.487,7	**13**	333,29	**13**	1.3192,6	14	129,3	14	62,2	**13**	201,1	*o*	*o*
G.1	**176**	**176**	5.259,8	**176**	375,4	**176**	30.2000,0	184	287,8	184	325,6	**176**	147,0	*o*	*o*
G.2	**154**	**154**	1.567,1	**154**	2.956,6	155	360,5	163	204,9	161	370,1	**154**	783,4	*o*	*o*
G.3	**166**	**166**	5.339,7	**166**	2.354,2	**166**	7.841,6	174	318,2	175	378,6	**166**	978,0	*o*	*o*
G.4	**168**	**168**	2.414,2	**168**	765,3	**168**	25.304,7	176	292,0	176	332,2	**168**	378,5	*o*	*o*
G.5	**168**	**168**	14.549,6	**168**	436,9	**168**	549,3	175	277,5	172	262,6	**168**	237,2	*o*	*o*
H.1	**63**	**63**	4.908,4	**63**	1.531,2	64	1.682,1	68	317,7	68	488,4	**63**	1.451,1	*o*	*o*
H.2	**63**	**63**	9.989,6	**63**	1.923,2	64	530,3	66	293,9	67	380,7	**63**	887,0	*o*	*o*
H.3	**59**	**59**	4.596,3	**59**	3.256,2	**59**	1.803,5	65	325,1	63	443,1	**59**	1.560,3	*o*	*o*
H.4	**58**	**58**	10.630,6	**58**	10.800,9	**58**	27.241,8	63	333,5	62	354,7	**58**	237,6	*o*	*o*
H.5	**55**	**55**	12.927,6	**55**	48,5	**55**	449,5	60	303,0	58	321,3	**55**	155,4	*o*	*o*

Tabela 5.10: Resultado das instâncias E a H testadas provenientes da biblioteca OR-Library

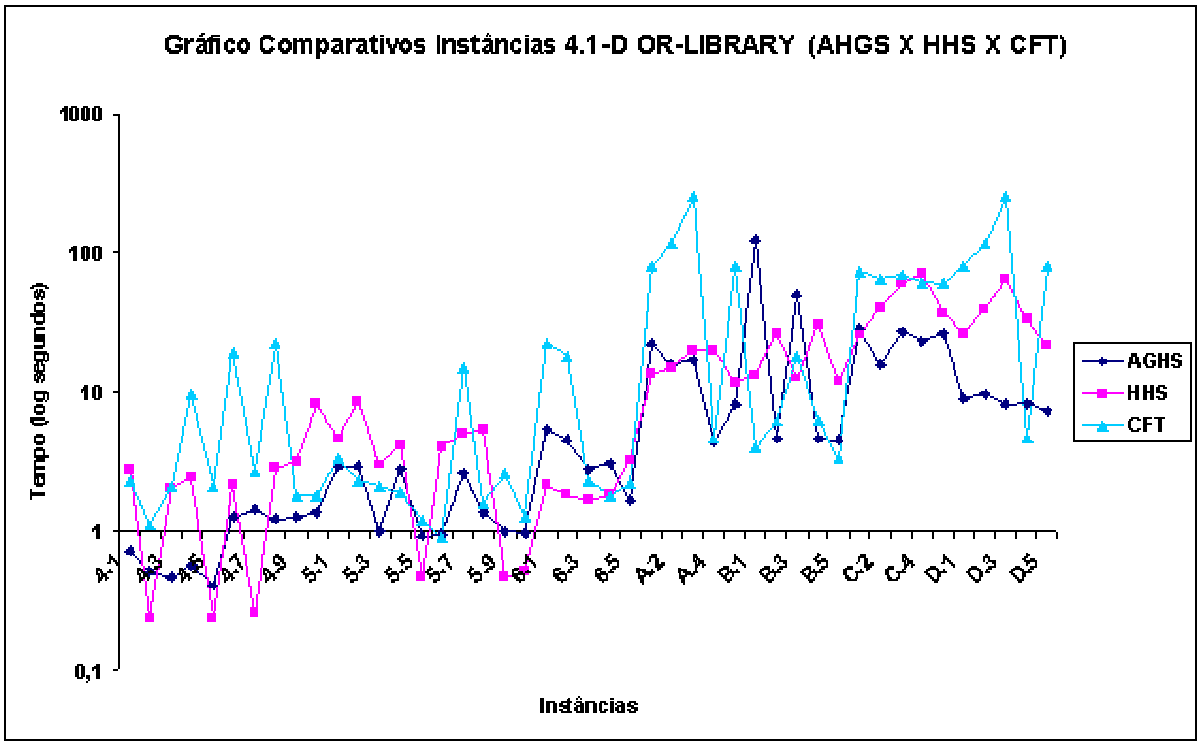

Figura 5.1: Gráfico Comparativo AHGS X HHS X CFT - Instâncias A - D (OR-LIBRARY)

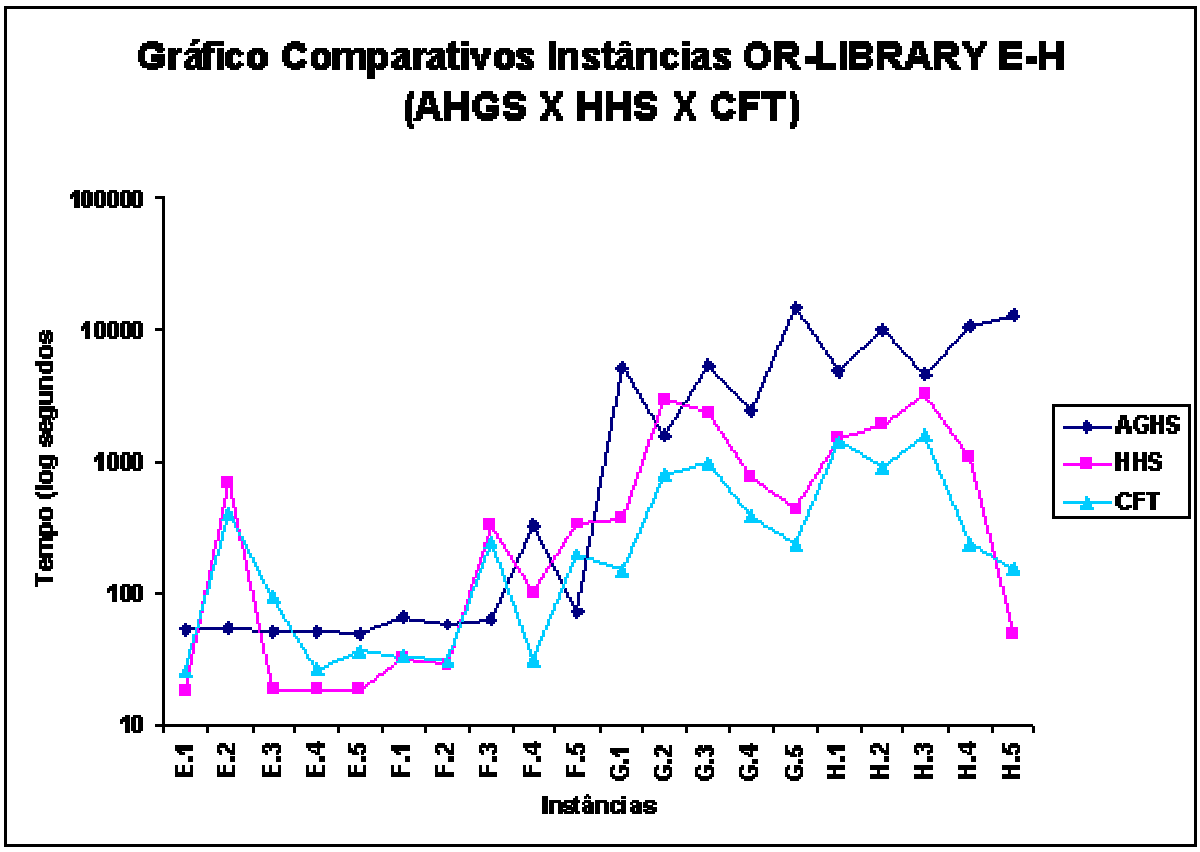

Figura 5.2: Gráfico Comparativo AHGS X HHS X CFT - Instâncias E - H (OR-LIBRARY)

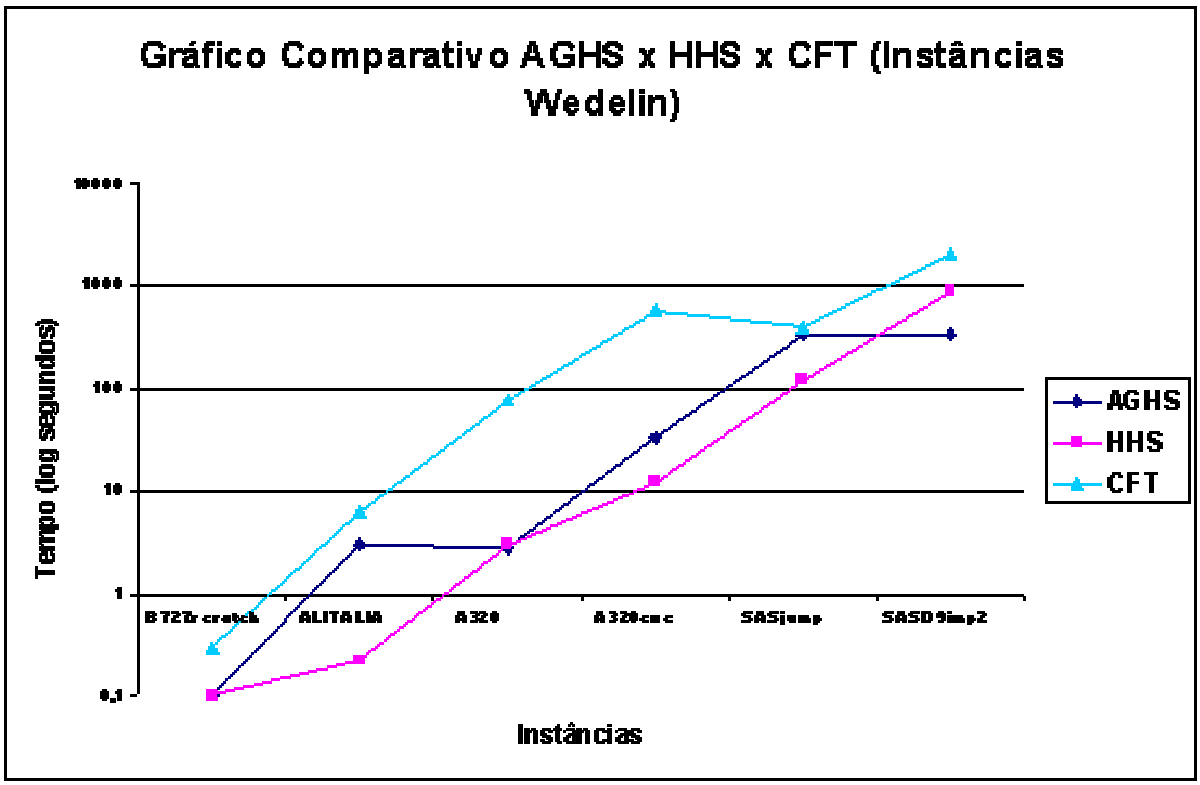

Figura 5.3: Gráfico Comparativo AHGS X HHS X CFT - Instâncias Wedelin

5.3.2 Instâncias Aéreas de Wedelin e Balas e Carrera

A tabela 5.11 apresenta os resultados computacionais obtidos pelos algoritmos implementados AHGS e HHS quando aos mesmos são submetidas as instâncias de alocação de tripulações de Wedelin [Wed95a]. Essa tabela apresenta também os resultados por Wedelin [Wed95a] e Caprara e Outros [CFT99] para esse conjunto de instâncias. A tabela 5.12, por sua vez, mostra os resultados obtidos pelos algoritmos implementados para as instâncias de Balas e Carrera [BC96a] e os compara com aqueles obtidos pelo autor das instâncias em Caprara e outros. Os rótulos dessas tabelas têm significado idênticos àqueles já especificados na tabela 5.10.

A observação dessas tabelas mostra que os algoritmos AHGS e HHS alcançaram a melhor solução da literatura em todas as instâncias consideradas. Além disso, para as instâncias SASjump e SASDimp9 a solução por eles encontrada foi melhor que aquelas reportadas pelo algoritmo CFT.

As figuras 5.5 e 5.6 mostram o tempo dispendido pelos algoritmos AHGS, HHS e CFT para alcançar a solução reportada nas tabelas 5.11 e 5.12 respectivamente . Observa-se que, para as instâncias de Wedelin, AHGS e HHS obtiveram, sempre, melhor desempenho quando comparados ao CFT. Para as instâncias de Balas e Carrera, na maioria das instâncias, o algoritmo AHGS apresenta melhor desempenho que os outros algoritmos.

Instância	MS	AHGS		HHS		Wedelin		CFT	
		Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo
B727scratch	**94.400**	**94.400**	0,1	**94.400**	0,1	**94.400**	4,7	**94.400**	0,3
ALITALIA	**27.258.300**	**27.258.300**	3,0	**27.258.300**	0,22	**27.258.300**	37,2	**27.258.300**	6,2
A320	**12.620.100**	**12.620.100**	2,8	**12.620.100**	3,0	**12.620.100**	216,9	**12.620.100**	79,5
A320coc	**14.495.500**	**14.495.500**	33,0	**14.495.500**	12,43	**14.495.500**	1.023,7	14.495.600	577,8
SASjump	**7.338.844**	**7.338.844**	340,0	**7.338.844**	119,9	7.340.777	806,8	7.339.537	396,3
SASD9imp2	**5.262.040**	**5.262.040**	347,1	**5.262.040**	889,3	5.262.190	1.579,7	5.263.640	2.082,1

Tabela 5.11: Resultado das instâncias de Wedelin

Instância	MS	AHGS		HHS		BaCa		CFT	
		Sol	Tempo	Sol	Tempo	Sol	Tempo	Sol	Tempo
AA03	**33.155**	**33.155**	40,0	**33.155**	7,4	33.157	96,4	**33.155**	61,0
AA04	**34.573**	**34.573**	0,6	**34.573**	9,2	**34.573**	39,2	**34.573**	3,6
AA05	**31.623**	**31.623**	0,6	**31.623**	6,2	**31.623**	53,9	**31.623**	3,1
AA06	**37.464**	**37.464**	0,5	**37.464**	6,3	**37.464**	44,4	**37.464**	5,2
AA11	**35.384**	**35.384**	0,7	**35.384**	15,2	35.478	72,3	**35.384**	193,7
AA12	**30.809**	**30.809**	2,5	**30.809**	9,6	30.815	48,0	**30.809**	53,8
AA13	**33.211**	**33.211**	2.9	**33.211**	8,5	**33.211**	19,6	**33.211**	8,3
AA14	**33.219**	**33.219**	21.2	**33.219**	23,1	33.222	86,2	**33.219**	30,3
AA15	**34.409**	**34.409**	0.5	**34.409**	12,0	34.510	39,9	**34.409**	18,8
AA16	**32.572**	**32.572**	0.4	**32.572**	11,1	32.858	54,5	**32.572**	33,6
AA17	**31.612**	**31.612**	4.2	**31.612**	1,7	31.717	47,0	**31.612**	10,9
AA18	**36.782**	**36.782**	0.5	**36.782**	11,3	36.866	66,2	**36.782**	13,5
AA19	**32.317**	**32.317**	0.4	**32.317**	1,7	**32.317**	27,6	**32.317**	5,9
AA20	**34.912**	**34.912**	2.0	**34.912**	8,1	35.160	34,4	**34.912**	13,6
BUS1	**27.947**	**27.947**	2.9	**27.947**	0,3	**27.947**	62,8	**27.947**	5,0
BUS2	**67.760**	**67.760**	36.9	**67.760**	0,8	67.868	356,0	**67.760**	19,2

Tabela 5.12: Resultado das instâncias de Balas e Carrera

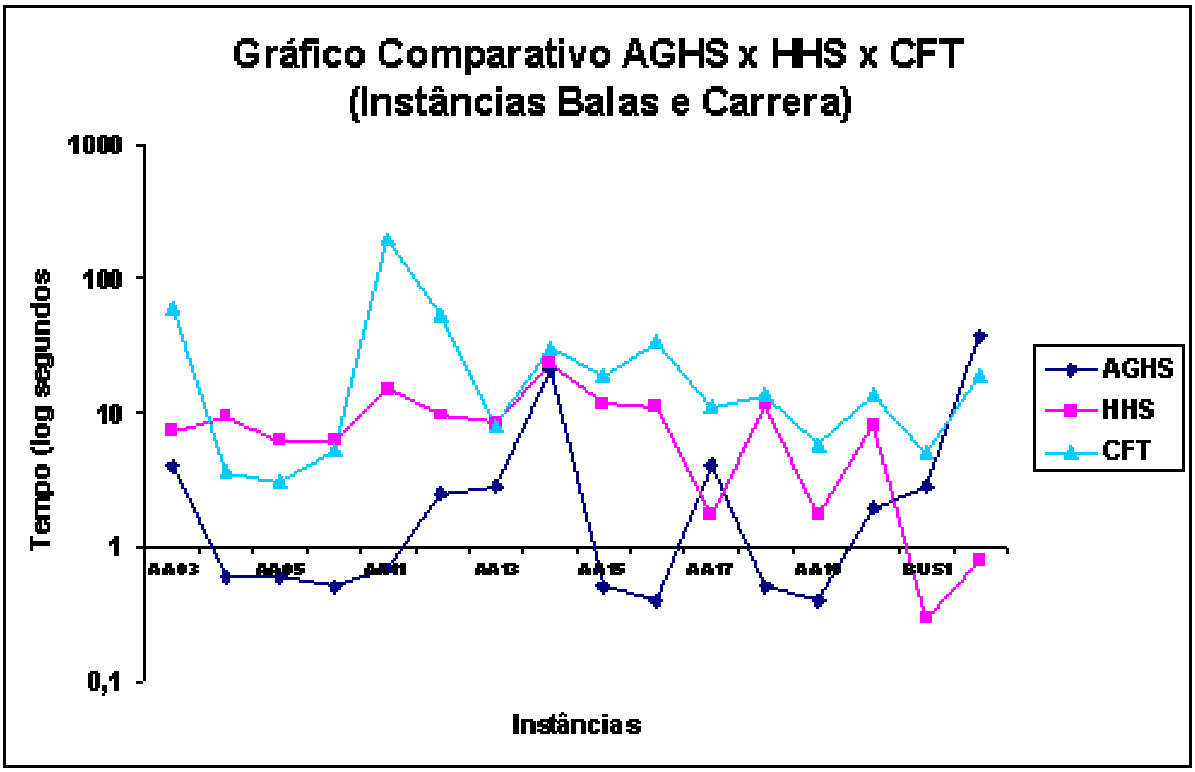

Figura 5.4: Gráfico Comparativo AHGS X HHS X CFT - Instâncias Balas e Carrera

5.3.3 O Algoritmo Genético como Seletor de Colunas e Gerador de Soluções Viáveis

Algoritmos genéticos, para instâncias de grande porte, são, muitas vezes, incapazes de alcançar as soluções ótimas para uma dada instância do *PR*. Para tanto, optou-se por uma estratégia híbrida em que o algoritmo genético além de sua função original de gerar soluções viáveis de qualidade passa também a, dado um conjunto de colunas, selecionar algumas que têm potencial de gerar boas soluções. No caso do AHGS essas colunas são selecionadas no final da fase FAG e repassadas para a fase *FSX*.

Os dados apresentados mostram que a fase *FAG* do AHGS foi bem sucedida na função a ela apresentada – além de encontrar, para a maioria das instâncias, soluções ótimas ou próximas das ótimas, selecionou um conjunto de colunas que, em alguns casos proporcionou a melhora da melhor solução anteriormente encontrada.

A tabela 5.13 mostra que o algoritmo genético antes de selecionar colunas a fazerem parte da fase posterior – FSX, obteve soluções quase próximas da melhor solução disponível para cada instância. Nessa tabela, *Nome* representa cada uma das classes de instâncias da biblioteca Or-Library.*Per*, por sua vez, indica o percentual de desvio da melhor solução encontrada pela fase FAG em relação à melhor solução disponível. Esse valor foi obtido da seguinte maneira: sejam NIC, SG , MS o número de instâncias de uma classe de problemas, a melhor solução encontrada na fase FAG e a melhor solução de uma instância respectivamente e *P = (SG - MS)/MS*. Assim, para cada classe de instância, os valores de *P* são somados e, posteriormente, divididos por NIC.

A tabela 5.14 mostra o número médio de colunas eliminadas (NCE) para cada classe de instâncias testada. Esse valor foi obtido através da média aritmética entre o número de colunas eliminadas para cada um dos testes executados para uma dada classe de instâncias. A tabela mostra que, para algumas classes de instâncias, mais de 90% das colunas foram eliminadas antes de serem submetidas à fase FSX.

Observa-se que a seleção de colunas na fase FAG é de fundamental importância no algoritmo porque ela diminui o espaço de busca por soluções e, ao mesmo tempo, reduz o tempo computacional para se encontrar uma solução na fase *FSX*. No entanto, é válido ressaltar que o procedimento por ela utilizado não garante que a redução do espaço de amostragem efetuado será feita de tal forma que colunas que possam vir a fazer parte da solução ótima do problema não sejam eliminadas. Na prática essa fase mostrou-se robusta, o que pode ser observado por meio da tabela 5.15 que mostra soluções médias sempre próximas das melhores soluções encontradas. Esses valores foram calculados como aqueles presentes na tabela 5.13. No entanto, o valor de SG, para essa situação, representa a média das soluções encontradas para cada instância.

Nome	Per (%)
4	0,04
5	0,21
6	0,4
A	0,49
B	0,51
C	1,77
D	0,59
E	0
F	0
G	2,89
H	1,64
Balas e Carrera	0,95
Wedelin	7,2

Tabela 5.13: Média, em percentual, da melhor solução do genético para a melhor solução, em cada classe de instâncias.

Nome	NCE	Per (%)	Nome	NCE	Per	Nome	NCE	Per
4	893,8	89,3	B727scratch	106	67,5	AA12	4,028	95,7
5	1887,2	94,3	ALITALIA	1,047,5	89,9	AA13	3,940	97,8
6	898,8	89,8	A320	6,836	98,6	AA14	3,523	91,0
A	2810,6	93,6	A320coc	17,280	92,1	AA15	3,449	93,1
B	2,844,1	94,8	SASjump	8,155	78,6	AA16	3,342	93,9
C	3,744,5	93,6	SASD9imp2	18,126	72,4	AA17	3,337	97,4
D	3,827,8	95,6	AA03	8,447	97,8	AA18	3,005	90,6
E	4,903,9	98,0	AA04	7,887	98,5	AA19	3,128	97,6
F	4,913,7	98,2	AA05	7,319	98,4	AA20	2,898	93,6
G	8,826,3	88,2	AA06	6,832	98,2	BUS1	2,131	95,0
H	9,832,5	98,3	AA11	4,092	92,7	BUS2	4,574	48,0

Tabela 5.14: Média de colunas eleminadas em cada classe de instâncias testadas

Nome	Per (%)
4	0,06
5	0,01
6	0,16
A	0,02
B	0
C	0
D	0,01
E	0,05
F	0,06
G	0,5
H	2,18
Balas e Carrera	0,12
Wedelin	0,16

Tabela 5.15: Média, em percentual, da média das soluções encontradas para a melhor solução, em cada classe de instâncias.

5.4 AHGS X HHS

Os algoritmos AHGS e HHS, apesar de serem ambos híbridos e utilizarem, em algum momento, o Simplex com *branch-and-bound* para obtenção de soluções, apresentam princípios de funcionamento diferentes. Enquanto o primeiro apresenta caráter aleatório e nenhuma garantia de seu funcionamento e inexistência de critérios que permitam a avaliação da solução por ele obtida, o segundo, baseado na relaxação lagrangeana, tem o seu funcionamento formalmente demonstrado e gera um limite inferior que indica, a distância da melhor solução viável encontrada para a solução ótima.

Nas subseções anteriores foram apresentadas as soluções e o tempo computacional obtidos pelos algoritmos para cada grupo de instâncias. As figuras apresentam a evolução das soluções desses dois algoritmos para duas instâncias da literatura F5 e H5[3]. Os dois algoritmos, logo na primeira iteração, geram soluções que estão bem próximas da ótima, o que indica que eles podem também ser utilizados para obtenção de soluções rápidas. O algoritmo HHS, em particular, nas duas instâncias apresentadas, gerou uma solução inicial de melhor qualidade. Esse padrão repete-se para a maioria das instâncias testadas.

Logo após a primeira iteração, o AHGS tende a ter uma queda brusca no valor das soluções obtidas e, em seguida, as melhoras passam a ser de menor magnitude. Isso pode ser justificado pelo fato de que a população inicial do AG é gerada randomicamente. Após a primeira iteração, procedimentos heurísticos já foram realizados e, por conseqüência, existe uma melhora considerável na solução inicial.

A observação do comportamento da evolução das soluções no HHS mostra que o mesmo tende, ao longo das iterações, a realizar melhoras mais equilibradas.

Em resumo não se pode salientar a superioridade de um algoritmo sobre o outro. Ambos apresentam performance e desempenho similares e são relativamente simples de ser implementados. O único fator que pode levar a escolha da implementação do HHS em detrimento do AHGS é que ele gera um limite numérico capaz de avaliar a solução obtida pelo algoritmo.

[3]Observe que o tempo computacional não está relacionado com o número de iterações gasto pelo algoritmo a cada iteração.

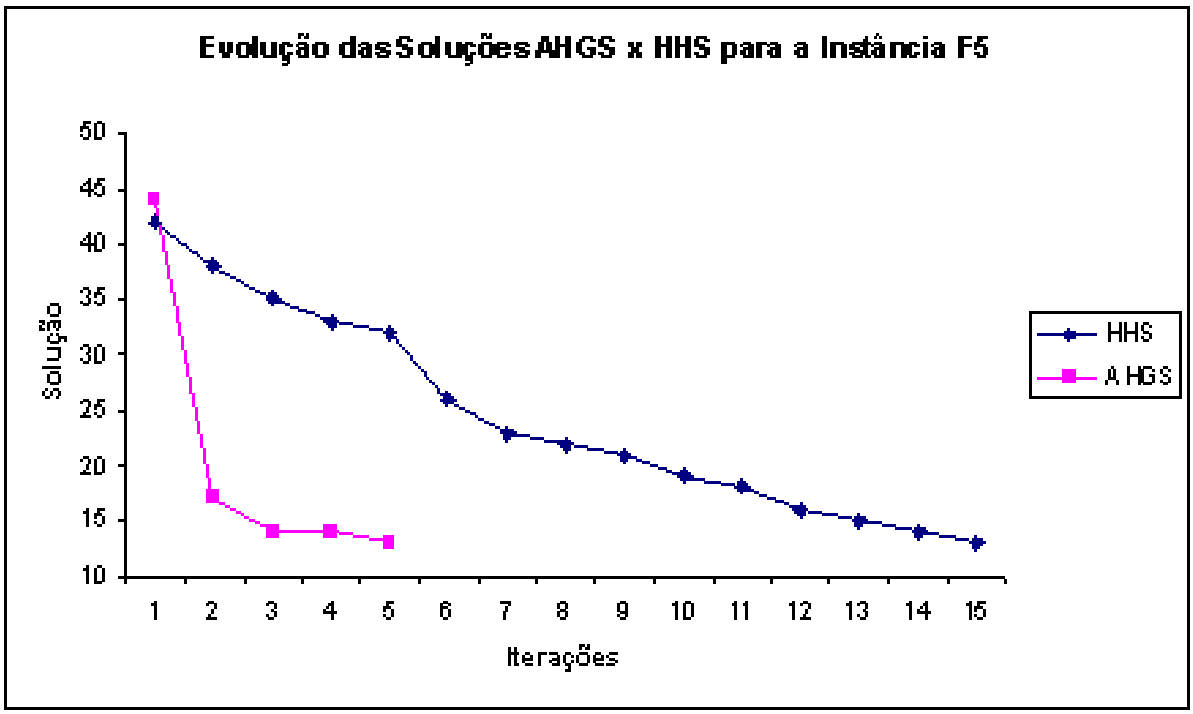

Figura 5.5: Evolução da Solução dos algoritmos AHGS e HHS para a instância F5 da biblioteca Or-Library.

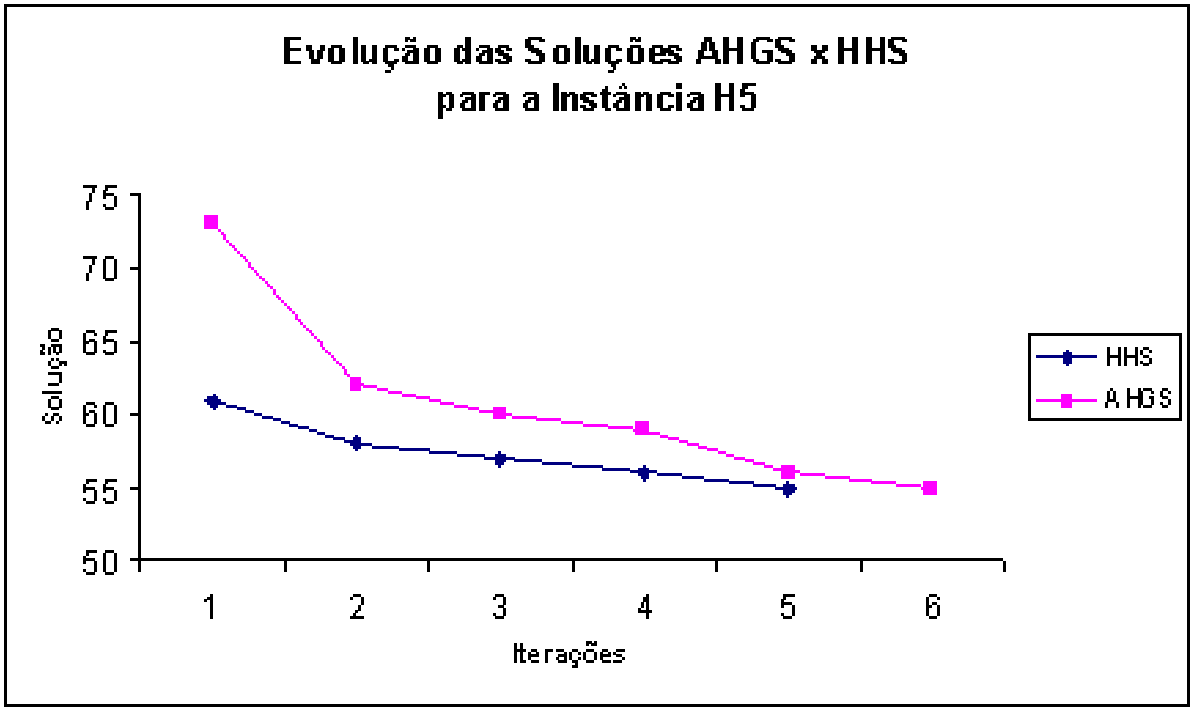

Figura 5.6: Evolução da Solução dos algoritmos AHGS e HHS para a instância H5 da biblioteca Or-Library.

Capítulo 6

Conclusão e Trabalhos Futuros

Nessa dissertação foram estudados os problemas envolvidos no Problema de Programação de Viagens e a maneira como eles se relacionam para obtenção de uma solução viável. Em particular, focou-se em um dos subproblemas do Problema de Alocação de Tripulações – o Problema de Recobrimento.

Para a obtenção de soluções possivelmente ótimas, seguiu-se para a implementação de duas heurísticas híbridas: o HHS, baseado na relaxação lagrangeana e o AHGS, um algoritmo genético.

Os algoritmos implementados mostraram-se eficazes quando a eles foram submetidas instâncias reais e da literatura para *benchmark* de algoritmos. Se comparados com os mais bem sucedidos resultados já reportados, pode-se dizer que os mesmos são competitivos, e, em algumas situações, melhores. Mesmo assim, algumas otimizações, ainda não implementadas, podem melhorar a performance dos mesmos:

- O Algoritmo HHS utiliza-se do simplex com o método *branch-and-bound* para obtenção de soluções. No entanto, a cada nova iteração, nenhum aproveitamento da base da iteração anterior é efetuado. Garatindo que colunas pertencentes à base da iteração anterior não são eliminadas de uma iteração para outra, pode-se fazer com que o simplex, ao invés de sempre iniciar o processo de solução convencional, inicie sempre a partir de uma base viável. Tal procedimento pode melhorar o desempenho do algoritmo uma vez que tem-se a expectativa de que, à medida que o algoritmo caminha em direção à solução ótima, poucas mudanças tendem a acontecer na base associada à solução ótima.

- A fase heurística do algoritmo HHS implementa parte do algoritmo guloso proposto por Caprara e outros [CFT99]. Eles implementam uma fase adicional – a fixação de colunas. Segundo eles essa fase foi responsável pela obtenção das melhores soluções disponíveis em diversas instâncias testadas. O algoritmo HHS, em algumas instâncias, só alcança a melhor solução disponível da literatura quando executa o simplex seguido de *branch-and-bound*. Assim, pode-se também optar pela implementação da fixação de colunas na heurística gulosa de modo a evitar a busca pela

melhor solução em um algoritmo exato que requer, na maioria das vezes, maior intervalo de tempo para executar. O operador de cruzamento heurístico apresentado no algoritmo AHGS pode também ter a fixação de colunas inserida em seu procedimento.

- Dentre as colunas e linhas submetidas como entrada aos algoritmos implementados, pode existir um subconjunto das mesmas que seja redundante. Adicionalmente, existem também técnicas que eliminam colunas baseadas no limite superior disponível e no custo reduzido encontrado na relaxação linear. Essas e outras abordagens têm sido aplicadas no pré-processamento dos arquivos de entrada no problema de recobrimento e mostram-se eficazes na diminuição do número de variáveis ou restrições do problema.

O PR é apenas um dos problemas envolvidos no PAT. Esse, por sua vez, é também parte de um problema maior – o PPV.

Portanto faz-se também necessária a implementação dos algoritmos para a resolução dos outros problemas envolvidos no PAT: problema de geração de jornadas e problema de seqüenciamento de jornadas de forma que uma solução única para o problema como um todo seja gerada.

A partir daí, os outros blocos de problemas: problemas de montagem de horários e problema de escalonamento de veículos devem também ser implementados. Por fim, uma interface única, que integra todos os problemas do PPV deve ser criada gerando, como saída, uma solução para esse problema.

Referências Bibliográficas

[AMA94] Jan Ahmad, Yamamoto Masahito, and Ohuchi Azuma. Genetic evolutionary algotithms for nurse scheduling problem. *Hokkaido University*, 1994.

[ANS00] Alper Atamtuerk, George L. Nemhauser, and Martin W.P. Savelsbergh. The mixed vertex packing problem. *Mathematical Programming*, 89:35–53, 2000.

[BC96a] E. Balas and M. C. Carrera. A dynamic subgradient-based branch and bound procedure for set covering. *Operations Research*, 44(6):875–890, 1996.

[BC96b] J. E. Beasley and P.C. Chu. A genetic algorithm for the set covering. *European Journal of Operational Research*, 94(2):392–404(13), 1996.

[BC96c] J.E. Beasley and B. Cao. A tree search algorithm for the crew scheduling problem. *European Journal of Operational Research*, 94:517–526, 1996.

[BC98] J.E. Beasley and B. Cao. A dynamic programming based algorithm for the crew scheduling problem. *Computers and Operations Research*, 25:567–582, 1998.

[Bea87] J.E. Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31:85–93, 1987.

[Bea90] J. E. Beasley. A lagrangian heuristic for set covering problems. *Naval Research Logistics*, 37:151–164, 1990.

[Ber95] D. P. Bertsekas. *Nonlinear Programming. Athena Scientific, Belmont, Massachusetts*. Athena Scientific, Belmont, Massachusetts, 1995.

[BH80] E. Balas and A. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. *Mathematical Programming*, (12):37–60, 1980.

[BJN+94] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savellberg, and P.H.Vance. Branch-and-price: Column generation for solving huge integer programs, 1994.

[BR71] M. Belmore and HD Ratliff. Set covering and involuntary bases. *Management Science*, (18):194–206, 1971.

[CDMT89] G. Carpaneto, M. Dell'Amico, M.Fischetti, and P. Toth. A branch and bound algorithm for the multiple depot vehicle scheduling problem. *Networks*, 19:531–548, 1989.

[CFL+98] A. Caprara, F. Focacci, E. Lamma, P. Mello, M. Milano, P. Toth, and D. Vigo. Integrating constraint logic programming and operations research techniques for the Crew Rostering Problem. *Software: Pracice and Experience*, 28(1):49–76, 1998.

[CFT99] A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47:730–743, 1999.

[CGKK97] P. Calégari, F. Guidec, P. Kounen, and D. Kobler. Parallel island-based genetic algorithm for radio network design. *Journal of Parallel and Distributed Computing*, (47):86–90, 1997.

[Chv79] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[CNS98] S. Ceria, P. Nobili, and A. Sassano. A lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81:215–228, 1998.

[Col95] Emma Collingwood. Investigation of a multiple chromosome evolutionary algorithm for bus driver scheduling and other problems. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, 1995.

[CTFV98] A. Caprara, P. Toth, M. Fischetti, and D. Vigo. Modeling and solving the crew rostering problem. *Operations Research*, 46:820–830, 1998.

[DW60] G.B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[Ere98] A.V. Eremeev. A genetic algorithm with a non-binary representation for the set covering problem. *In Proc. of OR'98, Springer-Verlag*, 29:175–181, 1998.

[Etc77] J. Etcheberry. The set covering problem: A new implicit enumeration algorithm. *Operations Research*, (13):760–772, 1977.

[Fan94] Hsiao-Lan Fang. *Genetic algorithms in timetabling and scheduling.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1994.

[FaW99] R. Freling, J. ao, and A. Wagelmans. Models and algorithms for vehicle scheduling. *Report 9562/A, Econometric Institute, Erasmus University Rotterdam*, 1999.

[FHW94] M.A. Forbes, J. N. Holt, and A.M. Watts. An exact algorithm for the multiple depot bus scheduling problem. *European Journal of Operational Research*, 72(1):115–124, 1994.

[FJK+99] T. Fahle, U. Junker, S. Karisch, N. Kohl, M. Sellmann, and B. Vaaben. Constraint programming based column generation for crew assignment. Technical Report tr-ri-99-212, University of Paderborn, 1999.

[FLO00] Richard Freling, Ramon M. Lentink, and Michiel A. Odijk. Scheduling train crews: a case study for the dutch railways. Technical Report EI2000-17/A, Econometric Institute, 2000.

[FR89] T. A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[HK70] M. Held and R. M Karp. The travelling-salesman problem and minimum spanning trees. *Operations Research*, (18):1138–1162, 1970.

[HK71] M. Held and R. M Karp. The travelling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, (1):6–25, 1971.

[Hol75] J. H. Holland. Adaptation in natural and artificial systems. *MIT Press*, 1975.

[Joh74] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[JZ00] Szustakowski JD and Weng Z. Protein structure alignment using a genetic algorithm. *Proteins*, 4(38):428–440, 2000.

[KJN01] D. Klabjan, E.L. Johnson, and G. L. Nemhauser. Solving large airline crew scheduling problems: random pairing generation and strong branching. *Computational Optimization and Applications*, 20:73–91, 2001.

[KL96] A. Kokott and A. Löbel. *Lagrangean relaxations and subgradient methods for multiple-depot vehicle scheduling problems.* Preprint SC 96-22, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996.

[KMC98] S. Khuri, W. Melody Moh, and F. Chung. Code assignment in cdma networks: Distributed algorithms and genetic algorithm heuristics. *Proceedings of the IEEE Singapore International Conference on Networks (SICON), Singapore*, pages 91–105, 1998.

[Lev96] D. Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers and Operations Research*, 6(23):547–558, 1996.

[Lib] OR Library. http://mscmga.ms.ic.ac.uk/jeb/orlib/scpinfo.html.

[Lov79] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1979.

[LSS71] C. E. Lemke, H. M. Salkin, and K. Spielberg. Set covering by single-branch enumeration with linear-programming subproblems. *Operations Research*, (19):998–1022, 1971.

[Löb99] A. Löbel. Vehicle scheduling in public transit and lagrangean pricing. *Management Science*, 44(12):1637–1649, 1999.

[NS92] G.L. Nemhauser and G. Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *Journal of Operations Research Society*, 43:443–457, 1992.

[RMP89] C.C. Ribeiro, M. Minoux, and M.C. Penna. An optimal column-generation-with-ranking algorithm for very large scale set partitioning problems in traffic assignment. *European Journal of Operations Research*, 41:232–239, 1989.

[RS94] C.C. Ribeiro and F. Soumis. A column generation approach to the mutiple-depot vehicle scheduling problem. *Operational Research*, 42:1:41–52, 1994.

[Sav97] M.W.P. Saveslsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45:6:831–841, 1997.

[Sen93] Sandip Sen. Minimal cost set covering using probabilistic methods. In *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*, pages 157–164. ACM Press, 1993.

[SPM99] D. Saha, M.D. Purkayasthaa, and A. Mukherjeeb. An approach to wide area wdm optical network design using genetic algorithm. *Computer Communications*, 22(2):156–172, 1999.

[SPU02] M. Solar, V. Parada, and R. Urrutia. A parallel genetic algorithm to solve the set-covering problem. *Computers and Operations Research*, 29:1221–1235, 2002.

[vdAvHS99] J. van den Akker, C. van Hoesel, and M. Savelsbergh. A polyhedral approach to single-machine scheduling problems. *Mathematical Programming*, (85):541–572, 1999.

[VW84] F.J. Vasko and G. R. Wilson. An efficient heuristic for large set covering problems. *Naval Research Logistics Quaterly*, 31:163–171, 1984.

[Wed95a] D. Wedelin. An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. *Annals of Operational Research*, (57):283–301, 1995.

[Wed95b] D. Wedelin. The design of a 0-1 integer optimizer and its application in the carmen system. *European Journal of Operational Research*, (87):722–730, 1995.

[WW95] Anthony Wren and David O. Wren. A genetic algorithm for public transport driver scheduling. *Computers and Operations Research*, 22(1):101–110, 1995.

[YMdS00] Tallys H. Yunes, Arnaldo V. Moura, and Cid C. de Souza. Modeling and solving a crew rostering problem with constraint logic programming and integer programming. Technical Report 00-04, Institute of Computing, UNICAMP, 2000.