3
Revisão Bibliográfica

3.1
Busca Tabu

Busca Tabu é uma estratégia para resolução de problemas de otimização combinatória. É um procedimento adaptativo, utilizado em conjunto com outros métodos, os guiando para a superação das limitações da otimalidade local (Glover, 1989). Utilizaremos um método mais familiar, o *Algoritmo de Gradiente* (também conhecido como Subida de Encosta), para ilustrar tais limitações e demonstrar as vantagens em utilizar a Busca Tabu.

Diversos métodos de otimização podem ser convenientemente caracterizados por seqüências de movimentos que levam de uma solução corrente a uma outra (Glover, 1989). Sendo S o conjunto de movimentos, cada solução x está associada a um conjunto de movimentos $S(x) \subseteq S$ que podem ser aplicados a ela. Quando um movimento $s \in S(x)$ é aplicado a x, esta solução converte-se em uma outra, $s(x)$.

O Algoritmo de Gradiente progride unidirecionalmente do seu ponto de início até um ótimo local. Inicialmente é selecionada uma solução inicial x_0 (possivelmente de forma aleatória), e esta solução é adotada como solução corrente $\bar{x} \leftarrow x_0$. A cada iteração do método, são analisados os movimentos $S(\bar{x})$ que podem ser aplicados a $\bar{x}$. Se não houver movimento que melhora $\bar{x}$, a busca é encerrada, pois um ótimo local foi alcançado. Caso contrário, é escolhido um movimento $s \in S(\bar{x})$ tal que o custo da solução resultante é melhor que o corrente (ou seja, $c(s(\bar{x})) < c(\bar{x})$ em um problema de minimização), a solução resultante é adotada como solução corrente ($\bar{x} \leftarrow s(\bar{x})$) e o procedimento é repetido.

Apesar de sua notável simplicidade, o Algoritmo de Gradiente é suficiente para resolução ótima de diversos problemas, nos quais a função de custo e a vizinhança são especificadas de tal forma que todo ótimo local é também um ótimo global. O algoritmo Simplex para resolução de Programas Lineares, por exemplo, pode ser interpretado como um Algoritmo de Gradiente (Glover, 1989). Para problemas de otimização combinatória em geral, o Algo-

ritmo de Gradiente pode ser utilizado como heurística, com a limitação de que o ótimo local encontrado pode não ser um ótimo global.

A Busca Tabu pode ser utilizada para guiar uma heurística deste tipo, dando continuidade à exploração mesmo quando não há movimento que melhore a solução corrente. A busca prossegue, porém com restrições para evitar ciclos, ou seja, evitar o retorno a soluções já visitadas. Para alcançar este objetivo é estabelecido um conjunto de movimentos $\mathcal{T} \subset S$ considerados proibidos, ou *tabu*.

São diversos os esquemas utilizados para restringir o espaço de busca. Em geral, porém, a cada iteração é escolhido o melhor movimento não-tabu (mesmo que este piore a solução corrente), logo é de se esperar que novos ótimos locais sejam encontrados conforme a busca prossiga.

3.1.1 Esquemas tabu estrito e não-estrito

Dizemos que é utilizado um esquema tabu *estrito* (*strict tabu*) quando configurações já visitadas são explicitamente proibidas. Este esquema tem convergência relativamente lenta, uma vez que na iteração n é necessária uma computação de ordem $O(n)$ para tal verificação explícita, o que pode ser aliviado pelo uso de *hashing codes* para representar as configurações (Battiti & Giampietro, 1994).

É bastante comum o uso de esquemas baseados em um tamanho máximo para $\mathcal{T}$. Tais esquemas não são estritos, portanto existe a possibilidade da ocorrência de ciclos.

3.1.2 Estratégias comuns de implementação

Para decidir quais movimentos serão incluídos em $\mathcal{T}$ a cada iteração, uma estratégia comum é proibir os movimentos que desfaçam os últimos movimentos realizados (Santos et al., 2005). Estes movimentos "destrutivos" são incluídos em $\mathcal{T}$, sendo removidos após um certo número de iterações, quando espera-se que a busca tenha sido direcionada para uma nova região.

Em relação à representação computacional do conjunto $\mathcal{T}$, pode-se manter uma estrutura de dados como uma representação direta de $\mathcal{T}$ (uma *lista tabu*, por exemplo) e para saber se um movimento é tabu, verifica-se a sua pertinência a esta estrutura. Porém, isto não é comum, devido ao custo da verificação de pertinência. O que geralmente utiliza-se é um vetor (ou função) $TabuEnd$ que associa cada movimento a um inteiro indicando até qual iteração este movimento é considerado tabu. Quando na iteração n

queremos que o movimento s torne-se tabu por τ iterações, é suficiente atribuir $TabuEnd(s) \leftarrow n+\tau$. Este valor τ é chamado *tabu tenure*, e quando o seu valor é fixo, o efeito equivale à manutenção de uma lista tabu de mesmo tamanho.

3.1.3 Busca Tabu Reativa

O problema da definição dos valores apropriados de parâmetros é recorrente em heurísticas. Uma escolha apropriada da dimensão do conjunto tabu (grande o suficiente para evitar ciclos, mas pequeno o suficiente para não restringir demasiadamente a busca) é crítica para o sucesso do algoritmo (Battiti & Giampietro, 1994).

Visando maior robustez, uma estratégia que pode ser adotada é adaptar τ dinamicamente de acordo com o comportamento da busca. Tal esquema *reativo* pode ser implementado verificando-se a cada movimento se a configuração obtida foi visitada recentemente, incrementando τ em caso positivo, ou decrementando suavemente caso contrário (Battiti & Giampietro, 1994).

Um outro esquema reativo, mais simplificado, orienta-se somente pelo valor da função objetivo, e não necessita a verificação explícita da repetição de configurações recentes. Se há pouca flutuação do valor da função a partir de um certo número de iterações atrás, entende-se que a busca está presa a uma região reduzida, portanto τ é progressivamente incrementado, aumentando a *diversificação*. Por outro lado, se há muita flutuação, τ é reduzido, para que a busca concentre-se em uma região específica, o que é chamado de *intensificação* (Blöchliger & Zufferey, 2008).

3.2 Simulated Annealing

Simulated Annealing, também conhecido como *Recozimento Simulado*, é um método de resolução de problemas de otimização combinatória inspirado no processo físico de *recozimento* (Kirkpatrick et al., 1983, Černỳ, 1985). É um método de busca local, assim como o Algoritmo de Gradiente e a Busca Tabu, que a cada iteração analisa um conjunto de movimentos $S(\bar{x})$ que podem ser aplicados à solução corrente $\bar{x}$, e pode optar por efetuar um destes movimentos, $s \in S(\bar{x})$ e consequentemente adotar uma nova solução corrente, $s(\bar{x})$.

Assim como a Busca Tabu, o método *Simulated Annealing* também busca superar as limitações da otimalidade local, discutidas na seção 3.1. No entanto, aqui utiliza-se uma abordagem probabilística para aceitação e recusa de movimentos que deterioram a qualidade da solução corrente, no lugar de

haver explicitamente um conjunto de movimentos proibidos, como é o caso na Busca Tabu.

3.2.1
Aceitação probabilística de movimentos

Um movimento que melhora a solução corrente é sempre aceito, enquanto um movimento que a deteriora é aceito de acordo com uma probabilidade que depende do grau de deterioração da qualidade da solução, de forma que quanto pior for o movimento, mais difícil será a sua aceitação. Formalmente, um movimento é aceito de acordo com uma distribuição de probabilidade conhecida como *distribuição de Metropolis*:

$$P(\delta) = \begin{cases} 1 & \text{se } \delta < 0 \\ e^{-\frac{\delta}{\tau}} & \text{caso contrário} \end{cases}$$

onde o parâmetro τ é o valor da *temperatura* corrente. De acordo com a analogia com o sistema físico, enquanto a temperatura está alta, o processo está mais suscetível a grandes mudanças, logo é maior a probabilidade de aceitação de movimentos que deterioram localmente a qualidade da solução, o oposto ocorrendo em baixas temperaturas.

3.2.2
Parametrização de Simulated Annealing

Dois componentes importantes do processo de *Simulated Annealing* são portanto a configuração da temperatura inicial e a programação do resfriamento. Uma abordagem robusta para configuração da temperatura inicial é executar uma fase inicial de calibragem do algoritmo, estimando um valor mediano $\tilde{\delta}$ da variação do valor da solução, por amostragem sobre a instância a ser resolvida. Em relação a programação do resfriamento, é comum ser utilizado um resfriamento geométrico: periodicamente, a temperatura corrente $\bar{\tau}$ é multiplicada por uma constante α (chamada de *taxa de resfriamento*), onde $0 < \alpha < 1$.

Também são comuns implementações onde ocorre o *reaquecimento*, ou seja, a elevação da temperatura em um estágio avançado do processo, para evitar que o algoritmo fique muito tempo preso a uma mesma região do espaço de busca, com baixa probabilidade de mover-se para outra região devido à baixa temperatura.

Estratégias diversas para a configuração dos parâmetros do *Simulated Annealing* aplicado ao PHC (ver seção 2.2) são experimentadas em (Rossi-Doria et al., 2003).

3.3 Formulações para o Problema de Coloração de Grafos

O Problema de Coloração de Grafos (PCG) é um dos modelos mais úteis da teoria dos grafos, sendo utilizado como base para modelar problemas nas mais diversas áreas, como por exemplo problemas de programação de horários, problemas de alocação de registradores (para o projeto de compiladores) (Smith et al., 2004), e problemas de alocação de freqüências (para a área de telefonia) (Aardal et al., 2007).

O PCG pode ser apresentado da seguinte forma: dado um grafo simples não-orientado $G = (V, E)$ e um inteiro k, uma k-coloração é um mapeamento $c : V \rightarrow \{1, \ldots, k\}$ que atribui uma cor $c(v)$ (ou rótulo) a cada vértice $v \in V$. Quando uma k-coloração é tal que todos os pares de vértices ligados por arestas recebem cores diferentes, ou seja, quando $c(u) \neq c(v)$ para toda aresta $(u, v) \in E$, diz-se que esta é uma k-coloração *válida*.

Alguns termos serão úteis em discursões posteriores: o subconjunto formado pelos vértices de cor c, denotado por $V^c \subseteq V$, é também chamado de uma *classe* de cor. Quando temos uma k-coloração válida, cada uma das k classes de cores forma um *conjunto independente*, o que significa que para quaisquer dois vértices $u, v \in V^c$, não existe aresta $(u, v) \in E$.

Na versão de decisão do PCG temos um k fixo, e o problema (chamado de k-PCG) é determinar se existe alguma k-coloração válida. Este problema é comprovadamente NP-completo (Garey & Johnson, 1979). Em sua versão de otimização, o problema é NP-difícil, e consiste em determinar o menor k para o qual existe uma k-coloração válida.

Uma boa revisão bibliográfica de diferentes formulações de Programação Linear Inteira Mista para o PCG é encontrada em (Burke et al., 2007). Aqui apresentaremos duas formulações para o PCG, que utilizaremos neste trabalho como base para as formulações propostas para o PPHCPM.

3.3.1 Formulação Padrão para o PCG

Uma formulação natural para o PCG utiliza $k|V|$ variáveis binárias:

– $\boldsymbol{x_{vc}}$, variável que indica se um vértice v recebe uma cor c. Esta variável está definida para todo vértice v e toda cor c:

$$x_{vc} \in \{0, 1\}, \qquad \forall v \in V, c \in \{1, \ldots, k\}$$

O problema consiste em determinar se existe uma solução válida para o sistema composto por dois grupos de restrições, o primeiro grupo garante que

todo vértice v receba uma cor c, e o segundo evita que vértices u e v adjacentes recebam uma mesma cor c:

$$\sum_{c\in\{1,\ldots,k\}} x_{vc} = 1 \qquad \forall v \in V \tag{3-1}$$

$$x_{uc} + x_{vc} \leq 1 \qquad \forall (u,v) \in E, c \in \{1,\ldots,k\} \tag{3-2}$$

A relaxação linear desta formulação, no entanto, produz limites inferiores muito fracos, pois as soluções obtidas tendem a ser extremamente fracionárias. Por exemplo, quando $k \geq 2$, a atribuição $x_{vc} = 1/k$ para todas as variáveis é uma solução viável para o sistema, conforme apontado em (Mehrotra & Trick, 1996).

3.3.2 Formulações de Representantes e de Representantes Assimétricos para o PCG

Uma das formulações originais mais recentes para o PCG, que chamamos de Formulação de Representantes Assimétricos, foi proposta em (Campêlo et al., 2008), fundamentada na formulação proposta anteriormente em (Campêlo et al., 2004), que chamamos simplesmente de Formulação de Representantes.

Em ambas as formulações, uma valoração viável das variáveis define uma partição $\{V^1,\ldots,V^k\}$ do conjunto de vértices V, onde cada parte V^c forma um *conjunto independente*. Associando cada um dos k conjuntos independentes a uma cor distinta, temos uma k-coloração válida.

Cada um dos conjuntos independentes é *representado* por um de seus elementos, considerado o seu *representante*. Os demais elementos também são representados pelo representante. Note que, como se trata de conjuntos independentes, um vértice pode representar somente (e ser representado por) aqueles que *não* estão em sua adjacência. Dado que $\bar{E}$ é o conjunto de arestas do grafo complementar $\bar{G} = (V, \bar{E})$, definimos para evitar ambiguidade:

- $\boldsymbol{\Delta(v)}$, adjacência de v: $\Delta(v) = \{u \in V : u \neq v, (u,v) \in E\}$
- $\boldsymbol{\bar{\Delta}(v)}$, *anti-adjacência* de v: $\bar{\Delta}(v) = \{u \in V : u \neq v, (u,v) \in \bar{E}\}$
- $\boldsymbol{\bar{\Delta}[v]}$, anti-adjacência *inclusiva* de v: $\bar{\Delta}[v] = \bar{\Delta}(v) \cup \{v\}$

A última notação, referente à anti-adjacência inclusiva, é útil por que um vértice também pode representar a si mesmo (é o caso do representante do conjunto independente).

De acordo com (Campêlo et al., 2004), estas formulações podem ser vistas como variações da Formulação de Conjuntos Independentes Maximais

proposta por (Mehrotra & Trick, 1996), porém sem utilizar uma quantidade exponencial de variáveis para representar os conjuntos independentes do grafo.

Formulação de Representantes para o PCG

Na Formulação de Representantes, proposta em (Campêlo et al., 2004), um vértice pode ser representado por qualquer vértice em sua anti-adjacência inclusiva. A definição da relação $Rep : V \rightarrow 2^V$, entre um vértice e aqueles que podem lhe representar, é bastante simples.

- $\boldsymbol{Rep(v)}$, conjunto de vértices que *podem representar* um vértice $v \in V$ definido como segue:

$$Rep(v) = \bar{\Delta}[v]$$

  Por conveniência, definimos também a relação inversa Rep^{-1}.

- $\boldsymbol{Rep^{-1}(u)}$, conjunto de vértices que *podem ser representados por* um vértice $u \in V$:

$$Rep^{-1}(u) = \{v \in V : u \in Rep(v)\}$$

Para esta formulação, são definidas variáveis binárias com o significado a seguir.

- $\boldsymbol{x_{uv}}$, variável que indica se um vértice u *representa* um vértice v. Esta variável está definida para todo vértice v e todo vértice u que pode representá-lo. Quando $u = v$, esta variável indica se u representa a si mesmo, e por consequência, se u é um vértice *representante* de um conjunto independente.

$$x_{uv} \in \{0, 1\}, \qquad \forall v \in V, u \in Rep(v)$$

O problema de decisão (k-PCG) consiste em encontrar uma solução válida para o sistema a seguir:

$$\sum_{u \in Rep(v)} x_{uv} \geq 1 \qquad \forall v \in V \tag{3-3}$$

$$x_{uv} + x_{uw} \leq x_{uu} \qquad \forall v \in V, w \in \Delta(v), u \in Rep(v) \cap Rep(w) \tag{3-4}$$

$$\sum_{v \in V} x_{vv} \leq k \tag{3-5}$$

O grupo de restrições 3-3 assegura que todo vértice v seja representado por algum vértice u que pode representá-lo. O grupo 3-4 garante que somente um dentre dois vértices v e w adjacentes (ou seja, em conflito) pode ser

representado por um vértice u que pode representar ambos, fazendo com que o conjunto representado por u seja independente. Este grupo também garante que um vértice v só seja representado por um vértice u que representa a si mesmo. Por último, a restrição 3-5 assegura que no máximo k vértices representam a si mesmo, e consequentemente que há no máximo k conjuntos independentes, o que implica em uma k-coloração válida.

Para a versão de otimização do PCG, onde queremos minimizar k, poucas alterações são necessárias no modelo: basta remover a restrição 3-5 e introduzir uma função objetivo minimizando o número de vértices representantes:

$$\min \sum_{v \in V} x_{vv}$$

3.3.3 Formulação de Representantes Assimétricos para o PCG

A Formulação de Representantes Assimétricos proposta em (Campêlo et al., 2008) apresenta uma redução significativa na quantidade de variáveis definidas. Para quebrar a simetria da formulação, é definida uma ordem $\prec$ sobre os vértices, e é estabelecido que se u e v são vértices tais que $v \prec u$, então u não pode representar v, e consequentemente o valor destas variáveis x_{uv} é fixado em 0, permitindo que as ignoremos.

A formulação é basicamente a mesma que a Formulação de Representantes, com exceção da definição da relação Rep que muda (e consequentemente, seu inverso, Rep^{-1}). Como um vértice não pode ser representado por outro de ordem menor, Rep agora define-se como segue:

$$Rep(v) = \{u \in \bar{\Delta}[v] : \neg(v \prec u)\}$$

Além disso, com a introdução da ordem, um subconjunto dos vértices $S \subseteq V$ é tratados de forma especial: trata-se dos vértices que formam um clique no início da ordem. Como cada vértice $u \in S$ pode apenas ser representado por si mesmo, o valor destas variáveis x_{uu} é fixado em 1, portanto é possível substituí-las por valores constantes nas expressões onde estariam presentes.

Naturalmente, $|S|$ é um limite inferior trivial da quantidade de cores necessárias para colorir o grafo (sendo somado explicitamente na função objetivo), levando a crer que seja vantajoso definir a ordem $\prec$ de forma que S seja o maior possível.

3.4

Local Branching

A Programação Linear Inteira Mista (PLIM) é uma ferramenta importante para a modelagem e resolução de problemas NP-difíceis de otimização combinatória. Os *softwares* para resolução exata de modelos de PLIM tem evoluído bastante, sendo capazes hoje de resolver, com otimalidade provada, problemas bem maiores do que há poucos anos atrás, o que tem fomentado o interesse no desenvolvimento de métodos de resolução baseados em PLIM.

No entanto, dado o tamanho dos problemas de interesse prático, em muitos casos é extremamente improvável que a solução exata seja obtida em tempo aceitável. Na prática, muitas vezes o importante é obter rapidamente uma solução viável de boa qualidade, por isso as heurísticas desempenham um papel importante.

Heurísticas Matemáticas de propósito geral tem sido propostas por diversos autores, buscando aproveitar o poder dos resolvedores de PLIM e sua generalidade (aplicabilidade em diversos tipos de problemas). Uma delas, chamada de *Local Branching*, proposta por (Fischetti & Lodi, 2003), pode ser vista como uma forma de *busca local* onde as vizinhanças são definidas através da inserção de *cortes de local branching*.

Apesar das idéias serem aplicáveis em modelos que também contém variáveis contínuas e inteiras, consideramos aqui um Programa Linear Inteiro, (PI), composto somente por variáveis binárias, para simplificar a discussão:

$$\begin{array}{lll} (PI) \quad \min & cx & \\ \quad s.a. & Ax \leq b & \\ & x_i \in \{0,1\} & \forall i \in \mathcal{I} \end{array}$$

onde $\mathcal{I}$ é o conjunto de índices $\{1,\ldots,n\}$ do vetor binário $x = (x_1,\ldots,x_n)$.

Dada uma *solução de referência* $\bar{x}$ de (PI), viável, formulamos cortes de *local branching*, que inseridos no modelo, restringem a busca a somente uma parte do espaço de soluções, formado pelas soluções na vizinhança de $\bar{x}$. Um modo simples de definir a vizinhança é utlizando um parâmetro k e considerando que uma solução vizinha $x \in \mathcal{N}(\bar{x}, k)$ é uma tal que no máximo k variáveis são diferentes de suas correspondentes em $\bar{x}$. Quando atribuímos valor zero ao parâmetro k, que indica o grau de liberdade da vizinhança, naturalmente a vizinhança se restringe à própria solução de referência, ou seja, $\mathcal{N}(\bar{x}, 0) = \{\bar{x}\}$. No outro extremo, quando fazemos k suficientemente grande, $\mathcal{N}(\bar{x}, k)$ compreende todo o espaço de soluções.

A adição da desigualdade a seguir serve exatamente para definir uma vizinhança. O conjunto $\mathcal{I}^0$ é formado pelos índices das variáveis que têm valor

0 na solução de referência, enquanto $\mathcal{I}^1$ são os índices das variáveis de valor 1:

$$\sum_{i \in \mathcal{I}^0} x_i + \sum_{i \in \mathcal{I}^1} (1 - x_i) \leq k$$

Como vemos, o primeiro somatório conta a quantidade de variáveis que eram 0 e mudaram para 1, enquanto o segundo somatório conta quantas variáveis mudaram de 1 para 0. Uma solução x viável de (PI) que também respeita esta desigualdade apresenta no máximo k diferenças em relação a $\bar{x}$.

Ainda, quando o conjunto $\mathcal{I}^1$ tem uma cardinalidade fixa para todas as soluções viáveis (o que é comum acontecer), podemos formular um corte mais simples, pois sabemos que a cada alteração de uma variável com índice em $\mathcal{I}^1$, obrigatoriamente corresponderá uma alteração de variável de índice em $\mathcal{I}^0$:

$$\sum_{i \in \mathcal{I}^1} (1 - x_i) \leq k/2$$

A idéia de *local branching* é que mesmo que o resolvedor de PLIM não consiga resolver (PI) em um tempo aceitável, ele pode ser capaz de resolver o modelo bem mais restrito obtido após a adição destes cortes. Caso a solução obtida, $\bar{x}' = \min_{x \in \mathcal{N}(\bar{x},k)} x$, seja melhor que $\bar{x}$, ela é adotada como solução de referência para a formulação de novos cortes semelhantes.

Uma estratégia ótima para exploração de todo o espaço de busca também é apresentada em (Fischetti & Lodi, 2003). Esta estratégia considera a adição do inverso de um corte (ou seja, trocando a desigualdade de $\leq$ para $>$) após o seu uso, para evitar que um mesmo espaço seja explorado múltiplas vezes. Um comportamento heurístico pode ser obtido simplesmente finalizando a busca e retornando a melhor solução encontrada até então, quando se atinge um limite de tempo definido.

Em (Hansen et al., 2006), é proposta uma heurística de Busca de Vizinhança Variável, baseada em *local branching*, porém sugerindo uma estratégia mais sistemática para as fases de *intensificação* e *diversificação* da busca. Neste trabalho são reportados resultados expressivos, melhorando vários dos melhores resultados conhecidos até então para um conjunto de 29 instâncias difíceis considerado também em (Fischetti & Lodi, 2003).

3.5 Geração de Colunas

3.5.1 Decomposição de Dantzig-Wolfe

O princípio da decomposição foi introduzido por (Dantzig & Wolfe, 1960), e levou ao uso de algoritmos de geração de colunas para a resolução de Programas Lineares (PL) de larga escala. Apresentamos aqui uma idéia de como funciona a decomposição, similar à forma como é apresentado em (Vanderbeck, 1994).

Muitas vezes é possível identificar na matriz de coeficientes de um PL uma estrutura composta por um ou mais subsistemas independentes, ligados por um conjunto de *restrições de ligação*. Consideremos um Programa Linear (P) da forma a seguir, onde as restrições 3-8 formam um subsistema e as restrições 3-7 são de ligação:

$$
\begin{aligned}
(P) \qquad \min \quad & cx && (3\text{-}6)\\
s.a. \quad & Ax = b && (3\text{-}7)\\
& Dx \leq d && (3\text{-}8)\\
& x \geq 0 && (3\text{-}9)
\end{aligned}
$$

As restrições 3-8 definem um poliedro no $\mathbb{R}^n_+$, $\{x \in \mathbb{R}^n_+ : Dx \leq d\}$ (que assumimos ser limitado e não-vazio). Considerando a matriz Q, cujas colunas Q_j representam todos os pontos extremos do poliedro, é possível elaborar um programa linear (MP) equivalente a (P), chamado de *programa linear mestre*, como segue.

$$
\begin{aligned}
(MP) \qquad \min \quad & cQ\lambda && (3\text{-}10)\\
s.a. \quad & AQ\lambda = b && (3\text{-}11)\\
& \mathbf{1}\lambda = 1 && (3\text{-}12)\\
& \lambda \geq 0 && (3\text{-}13)
\end{aligned}
$$

Note que devido à restrição 3-12, para qualquer vetor $\bar{\lambda}$ viável em (MP), $Q\bar{\lambda}$ equivale a um ponto $\bar{x}$ viável em (P), uma vez que $Q\bar{\lambda}$ é uma combinação convexa dos pontos extremos do poliedro.

3.5.2 Programa Linear Mestre

Na maioria das vezes, resolver o programa linear mestre diretamente não é fácil, devido à quantidade enorme (exponencial) de variáveis na formulação. Uma idéia para contornar este problema é trabalhar com um conjunto reduzido

de variáveis de cada vez, e acrescentar variáveis conforme seja necessário. Esta é a idéia do procedimento de geração de colunas.

Em cada iteração do procedimento de geração de colunas, é resolvido o *programa linear mestre restrito*, considerando somente um subconjunto $\tilde{Q} \subseteq Q$.

$$
\begin{aligned}
(RMP) \qquad \min \quad & c\tilde{Q}\lambda && (3\text{-}14)\\
s.a. \quad & A\tilde{Q}\lambda = b && (3\text{-}15)\\
& \mathbf{1}\lambda = 1 && (3\text{-}16)\\
& \lambda \geq 0 && (3\text{-}17)
\end{aligned}
$$

Como o (RMP) é mais restrito que o (MP), ele é apenas capaz de fornecer um limite superior (no caso de minimização) em relação ao valor da solução ótima do (MP), uma vez que possivelmente a adição de certas colunas melhora o custo da solução. No entanto, é possível verificar se a introdução de alguma coluna $Q_j \in Q \setminus \tilde{Q}$ no (RMP) é vantajosa, bastando examinar o *custo reduzido* de tal coluna, que explicaremos a seguir.

Como a coluna Q_j não está em (RMP), naturalmente ela não pertence à base de sua solução ótima, e o valor da variável λ_j associada a Q_j é zero. Somente será vantajoso introduzir Q_j no modelo se sua entrada modificar o custo da solução, e isso só pode ocorrer caso Q_j passe a fazer parte da base. Para cada unidade que aumentamos da variável λ_j, é necessário remover um pouco do valor de outras variáveis que já pertencem à base, o que provoca uma diferença no custo da solução: esta diferença é justamente o que chamamos de custo reduzido. Os custos reduzidos das colunas, mesmo as que não estão presentes no (RMP), podem ser calculados *em função dos valores das variáveis duais* de (RMP).

Uma forma de encontrar a coluna de menor custo reduzido é através da resolução de um outro problema, chamado de subproblema de *pricing*. A solução ótima dual obtida pela resolução do (RPM), $(\pi, \mu) \in \mathbb{R}^m \times \mathbb{R}$ é utilizada no subproblema:

$$v(\pi, \mu) = \min\{(c - \pi A)x : Dx \leq d, x > 0\} + \mu$$

Quando $v(\pi, \mu) \geq 0$, sabemos que não há coluna de custo reduzido negativo, e portanto programa linear mestre está resolvido. Caso contrário, a solução x^* do subproblema define uma nova coluna $Q_j = Ax^*$, de custo $c_j = cx^*$, a ser adicionada no programa linear mestre restrito, que pode eventualmente reduzir o seu valor.

3.5.3 Reformulação de um Programa Linear Inteiro

No mesmo espírito da reformulação de um programa linear em um programa linear mestre, quando trabalhamos com programas lineares inteiros também é possível formular um programa equivalente, chamado de *programa linear inteiro mestre*, e resolvê-lo via geração de colunas. No entanto, dada a natureza diferente destes problemas, as justificativas teóricas para a equivalência das formulações são diferentes. Uma apresentação bastante completa sobre a reformulação de programas lineares inteiros pode ser encontrada em (Vanderbeck, 1994, Vanderbeck & Wolsey, 1996).

Em resumo, queremos reformular um programa linear inteiro da forma:

$$(IP) \quad \min \quad cx \tag{3-18}$$
$$s.a. \quad Ax = b \tag{3-19}$$
$$Dx \leq d \tag{3-20}$$
$$x \in \{0,1\} \tag{3-21}$$

Considerando a matriz Q onde cada coluna Q_j representa um ponto x viável, ou seja, em $x \in \{x \in \{0,1\}^n : Dx \leq d\}$, decompomos o problema no seguinte *programa linear inteiro mestre*:

$$(IMP) \quad \min \quad cQ\lambda \tag{3-22}$$
$$s.a. \quad AQ\lambda = b \tag{3-23}$$
$$\mathbf{1}\lambda = 1 \tag{3-24}$$
$$\lambda \in \{0,1\} \tag{3-25}$$

Note que a composição da matriz Q é bem diferente do caso anterior, em que somente os pontos extremos do poliedro compunham a matriz. Analisando as restrições sobre as variáveis λ, é fácil ver que somente uma delas terá valor 1 em uma solução viável do (IMP). Em uma solução ótima do (IMP), a variável λ_j que assume valor 1 corresponderá a uma solução x_j ótima para o problema (IP) original.

Assim como no caso linear, é comum a adoção de um esquema de geração de colunas para resolução do (IMP), partindo de um *programa linear inteiro mestre restrito* em que apenas um subconjunto $\tilde{Q} \subseteq Q$ é considerado, e gerando colunas conforme seja necessário. Aqui também é possível utilizar os custos reduzidos para orientar a geração de colunas.

3.5.4 Estabilização da Geração de Colunas

Conforme observado por (Pigatti et al., 2005), algoritmos baseados em geração de colunas estão sujeitos a problemas de convergência. Para a resolução de instâncias distintas de mesmo tamanho, a quantidade de iterações necessárias para a convergência pode variar de uma ou duas ordens de magnitude. Em especial, os problemas emergem ao tentar resolver via geração de colunas instâncias muito degeneradas (Oukil et al., 2007).

Os valores das variáveis duais associadas podem variar bruscamente durante as primeiras iterações, ocasionando a geração de colunas que tem pouca chance de pertencer à solução final. Mais próximo das iterações finais, os valores duais aproximam-se dos seus valores finais e variam pouco, e neste ponto são geradas as colunas mais promissoras. Observe que isso ocorre por que cada coluna adicionada induz restrições sobre as variáveis duais, por isso é natural que ao final os valores duais variem menos. Baseado nestas observações, (Du Merle et al., 1999) propõem uma técnica de *estabilização dual*.

Esta técnica, assim como variações da mesma, tem sido aplicadas por diversos autores, como por exemplo (Pigatti et al., 2005, Oukil et al., 2007). Em geral, a estratégia consiste em adicionar certas variáveis à formulação para induzir restrições no problema dual, estabelecendo limites superiores $\overline{\pi}$ e inferiores $\underline{\pi}$ para o valor das variáveis duais π, ou estabelecendo valores de referência $\tilde{\pi}$ para as variáveis e associando um custo proporcional ao desvio do valor da variável em relação ao valor de referência.

Restrições duais para estabilização da Geração de Colunas

Partimos de uma formulação original com restrições como segue, associadas a variáveis duais π:

$$(\pi_i) \qquad a^i x = b_i$$

Para *estabilizar* as variáveis duais, em cada restrição incluímos variáveis $\alpha_i, \beta_i \geq 0$, com coeficientes como a seguir:

$$(\pi_i) \qquad a^i x - \alpha_i + \beta_i = b_i$$

Alteramos a função objetivo para considerarmos os custos de α e β. Estes custos são precisamente os valores que utilizaremos como limites inferiores e superiores para as variáveis duais.

$$\min \sum_{i \in \{1,\ldots,n\}} c_i x_i - \underline{\pi}_i \alpha_i + \overline{\pi}_i \beta_i$$

Alterando-se os limites superiores destas variáveis, induzimos diferentes restrições no problema dual. Se não há limite superior, as restrições induzidas são limites rígidos sobre as variáveis duais, ou seja:

$$\underline{\pi}_i \leq \pi_i \leq \overline{\pi}_i$$

Podemos estabelecer limites superiores para α e β, através de restrições na formulação primal, associadas às variáveis duais ω e ζ:

$$(\omega_i) \qquad \alpha_i \leq \epsilon$$

$$(\zeta_i) \qquad \beta_i \leq \epsilon$$

Por sua vez, as variáveis ω e ζ irão interferir nas restrições duais, relaxando os limites superiores e inferiores de π:

$$\underline{\pi}_i - \omega_i \leq \pi_i \leq \overline{\pi}_i + \zeta_i$$

O novo significado destas restrições é que os limites podem ser burlados, porém esta violação incorre em um custo, que é o custo associado às variáveis ω e ζ.

Quando utilizamos os limites flexíveis e fazemos $\underline{\pi} = \overline{\pi}$, isto é equivente a utilizarmos um valor de referência $\tilde{\pi}$.

Estratégia de resolução

Na abordagem utilizada por (Pigatti et al., 2005), é possível obter desde o início uma valoração de π próxima da valoração ótima. Sob estas condições, utilizamos esta valoração como referência para atribuir valores iniciais aos custos das variáveis α e β. Um valor inicial é estabelecido para ϵ, e resolve-se o problema resultante estabilizado até o seu ótimo, por geração de colunas. Como o problema é mais restrito, espera-se que a resolução seja mais rápida que a resolução do problema original.

O problema resolvido, no entanto, não é equivalente ao problema original, por isso diminuimos o valor de ϵ e repetimos todo o processo. Na última iteração, fazemos $\epsilon = 0$ e resolvemos o problema pela última vez: neste caso, as variáveis adicionais não podem assumir valor diferente de zero, logo o problema é equivalente ao original. Em (Pigatti et al., 2005), são executadas quatro iterações do processo, fazendo epsilon assumir os valores 0.1, 0.01 e 0.001 e finalmente 0 em cada iteração.