4
Combinando heurísticas com resolvedores MIP

Como discutido nas seções anteriores, de maneira geral existem duas formas de resolver um problema de otimização combinatória: utilizando métodos exatos ou métodos heurísticos. Ambos apresentam dificuldades: os métodos exatos tendem a ser lentos para instâncias grandes, enquanto os heurísticos exigem profundo conhecimento do problema, normalmente são específicos para o problema a ser resolvido (não facilmente reutilizáveis) e não proveem nenhuma informação sobre a qualidade global da solução.

Essas duas correntes independentes, vindas de comunidades científicas bastante diferentes [33], têm tido sucesso em resolver estes problemas. Está claro que ambas as estratégias têm vantagens e desvantagens complementares [10] - a idéia de combinar estas duas estratégias para gerar algoritmos mais poderosos é natural. Apesar disto, esta linha de pesquisa só veio a ser desenvolvida recentemente, nos últimos 5 a 10 anos.

A próxima seção relata uma taxonomia destas técnicas, depois será dedicada uma seção a cada uma das técnicas mais famosas. A última seção reapresenta e extende o conceito de vizinhanças elipsoidais.

4.1
Taxonomia

Um excelente *survey* de técnicas que combinam heurísticas (e metaeurísticas) com resolvedores MIP é o trabalho de Puchinger e Raidl [33]. Neste trabalho, além de relatar diversos trabalhos que unem as duas técnicas, eles propõem uma classificação das técnicas em combinações colaborativas e combinações integrativas.

4.1.1
Combinações colaborativas

Combinações colaborativas são combinações de alto nível onde nenhum algoritmo está contido no outro. Podem ser subdivididas em técnicas de execução sequencial ou de execução paralela, dependendo de como os algoritmos são executados.

Como exemplo de técnicas colaborativas sequenciais, pode-se citar vários trabalhos, como o de Applegate et al. [2], o de de Plateau et al. [32] e o de Feltl e Raidl [11]. No trabalho de Applegate et al. [2] para o PCV assimétrico, heurísticas são utilizadas para encontrar vários ótimos locais, os conjuntos de arestas dessas soluções são então usados para gerar um grafo e um resolvedor MIP é usado para resolver o PCV neste subgrafo. O algoritmo de Plateau et al. [32] começa com uma técnica de ponto interior e utiliza vários métodos de arredondamento para gerar várias soluções viáveis que são então utilizadas em uma metaeurística *scatter search*. O trabalho de Feltl e Raidl [11] propõe um algoritmo genético onde as soluções iniciais são geradas a partir de soluções para a relaxação linear do problema - este algoritmo será detalhado na seção 6.3.3. Existem poucas combinações colaborativas paralelas, um sinal de que as pesquisas nessa direção ainda não amadureceram o suficiente. O *framework* de times assíncronos (A-Teams) proposto por Talukdar et al. [36] sugere o uso de agentes distribuídos (com memória compartilhada), onde cada agente é responsável pela resolução do problema ou de uma relaxação do problema ou um subproblema.

4.1.2 Combinações integrativas

Tratam-se de combinações em que uma técnica é componente subordinado da outra. Podem ser subdivididas em técnicas que incorporam métodos exatos em metaeurísticas ou vice-versa.

Técnicas que incorporam métodos exatos em metaeurísticas normalmente utilizam os resolvedores para fazer buscas locais em grandes vizinhanças (*Very Large Neighborhood Search*, VLNS), como o algoritmo Dynasearch [5], onde a vizinhança a ser buscada é o conjunto (de tamanho exponencial) de todas as combinações de pequenas buscas. Outras técnicas utilizam métodos exatos para explorar sub-espaços formados pela união dos atributos de várias soluções, como os algoritmos genéticos de Cotta e Troya [6] e de Marino et al. [26]. No capítulo 5, uma nova maneira de agregar as características de multiplas soluções (batizada de vizinhança elipsoidal) será proposta e formalizada.

Entre as técnicas que incorporam metaeurísticas em métodos exatos, Puchinger e Raidl listam algoritmos que "aplicam o espírito das metaeurísticas" [33]. As próximas seções irão detalhar duas dessas técnicas, que têm apresentado excelentes resultados e, consequentemente, recebido bastante atenção dos pesquisadores da área.

4.2 Local Branching (LB)

O algoritmo *Local Branching* (LB) de Fischetti e Lodi [12] é sem dúvida uma das mais famosas integrações entre resolvedores MIP (neste caso, o CPLEX7) e técnicas normalmente utilizadas em metaeurísticas, principalmente por ter sido pioneiro na área e ter conseguido excelentes resultados em seu artigo original. A inovação do LB veio a partir de duas constatações: ao mesmo tempo em que os resolvedores MIP estão se tornando ferramentas sofisticadas e com bons resultados, eles ainda não conseguem resultados satisfatórios para instâncias especialmente difíceis ou grandes. Para essa categoria de instância, o importante não é encontrar (e provar) a solução ótima, mas sim conseguir soluções viáveis de boa qualidade em um espaço de tempo razoável - em outras palavras, como não é possível encontrar a solução ótima, é importante conseguir encontrar boas soluções rapidamente.

Partindo do modelo geral MIP definido na equação (3-1), o LB lida, sem perda de generalidade, somente com o conjunto de variáveis binárias $\mathcal{B}$ do modelo. Dada uma solução viável $\bar{x}$ do modelo, digamos que N_1 seja o conjunto dos índices das variáveis binárias cujo valor é 1 em $\bar{x}$: $N_1(\bar{x}) := \{j \in \mathcal{B} : \bar{x}_j = 1\}$. Dado um parâmetro inteiro e positivo k, pode-se definir a vizinhança k-*OPT* $\mathcal{N}(\bar{x}, k)$ de $\bar{x}$ como o conjunto de soluções viáveis do modelo que satisfazem a seguinte restrição, chamada de restrição de *local branching*:

$$\Delta(x, \bar{x}) := \sum_{j \in N_1(\bar{x})} (1 - x_j) + \sum_{j \in (\mathcal{B} \setminus N_1(\bar{x}))} (x_j) \leq k \tag{4-1}$$

onde os dois termos do lado esquerdo da inequação contam o número de variáveis binárias que mudam de valor em relação a $\bar{x}$ de 0 para 1 e vice-versa. Essa definição de distância é também conhecida como distância de Hamming [19]. Nos problemas em que o tamanho de N_1 é constante, essa restrição pode ser re-escrita da seguinte maneira:

$$\Delta(x, \bar{x}) := \sum_{j \in N_1(\bar{x})} (1 - x_j) \leq k'(= k/2) \tag{4-2}$$

onde esta nova medida de distância mede o número de variáveis que possuem valor 1 em ambas as soluções.

É importante notar que a definição de vizinhança k-OPT acima é consistente com as vizinhanças k-OPT normalmente utilizadas em heurísticas de busca local. Por exemplo, no problema do caixeiro viajante, a restrição (4-2) indica que a solução resultante x pode ser formada trocando-se até k' arestas da solução base $\bar{x}$.

A partir da definição das restrições de *local branching*, a idéia do algo-

ritmo em si é simples: cada nó da árvore de busca pode ser particionado em $\Delta(x, \bar{x}) \leq k$ (sub-árvore esquerda) e $\Delta(x, \bar{x}) \geq k + 1$ (sub-árvore direita). A idéia é encontrar um valor de k que permita que a sub-árvore esquerda possa ser resolvida em tempo razoável, gerando assim uma solução viável para o problema e um *bound* para o restante da busca. O valor ideal de k é grande o suficiente para que a solução x encontrada seja de boa qualidade (melhor que $\bar{x}$), mas não grande o suficiente para que a busca seja muito demorada.

Duas outras características do algoritmo LB são:

- **Limite de tempo na sub-árvore esquerda**: como a solução da sub-árvore esquerda pode consumir muito tempo, é imposto um limite de tempo. Caso o limite de tempo seja atingido e tenha sido encontrada uma nova solução $\bar{x}^2$, repete-se a busca na sub-árvore esquerda usando $\bar{x}^2$ como solução base. Se o limite de tempo for atingido sem que nenhuma solução melhor que $\bar{x}$ tenha sido gerada, o tamanho da vizinhança k é reduzido para $\lceil k/2 \rceil$ e a busca é repetida;
- **Diversificação**: novamente pegando emprestado técnicas utilizadas em metaeurísticas, o LB se utiliza de dois tipos de diversificação. O primeiro, chamado de diversificação "suave", aumenta o tamanho da vizinhança em $\lceil k/2 \rceil$ quando não é encontrada na sub-árvore esquerda nenhuma solução melhor que $\bar{x}$. Se o primeiro tipo de diversificação falha e novamente nenhuma solução melhor que $\bar{x}$ é encontrada, aplica-se uma diversificação "pesada": busca-se a primeira solução cuja distância de $\bar{x}$ seja pequena, sem impor nenhum limite superior nesta busca. Dessa maneira, encontra-se geralmente uma nova solução base, geralmente pior que a solução atual.

4.3 Relaxation Induced Neighborhood Search (RINS)

Relaxation Induced Neighborhood Search (RINS), desenvolvido por Danna et al. [7], são vizinhanças que se utilizam da solução da relaxação linear do problema para encontrar soluções durante o *branch and cut*. É uma técnica bastante simples mas que produz excelentes resultados, muitas vezes melhores e, em geral, mais rapidamente do que o LB [7]. O algoritmo parte do princípio de que, em cada nó da árvore de busca possui-se tanto a melhor solução encontrada até o momento (S, uma solução viável para o MIP) quanto a solução da relaxação linear daquele nó (T, que não é necessariamente uma solução viável para o MIP). As semelhanças entre estas duas soluções podem ser exploradas para melhorar a solução S. O RINS pode ser descrito em três passos, a serem executados em um nó da árvore de busca:

1. Fixar as variáveis que possuem os mesmos valores na solução viável S e na solução da relaxação T;

2. Usar um limite superior baseado no valor da solução viável S;

3. Resolver o "sub-MIP"nas variáveis restantes.

Qualquer solução encontrada pelo RINS é uma solução válida não só para o nó atual, mas para todo o modelo - dessa forma, se for encontrada alguma solução viável para o sub-MIP, ela vai substituir a solução S. A única consequência da execução do RINS é a descoberta desta nova solução viável e, consequentemente, de um melhor limite superior para a busca.

Como o sub-MIP a ser resolvido pelo RINS pode ser bastante complexo, principalmente se existirem poucas semelhanças entre S e T, o algoritmo utiliza um parâmetro de tempo máximo nl para cada execução. Existe também um parâmetro f ($f \gg 1$) que controla a frequência com que a busca é efetuada, já que a busca para nós próximos tende a ser bastante semelhante.

Colocado no contexto de algoritmos de busca local, pode-se classificar o RINS como um algoritmo que explora estruturas de vizinhança muito grandes (VLNS), onde a vizinhança é definida pela interseção entre a melhor solução até o momento S e a solução relaxada T. O RINS também se relaciona com a metaeurística *Path Relink* (detalhada na seção 2.2.2) já que, de certa forma, parte de duas soluções e varre o espaço de busca entre elas. Uma diferença importante em relação aos algoritmos *Path Relink* é que o RINS parte de duas soluções para problemas semelhantes, porém diferentes: o problema MIP em si (solução S) e sua relaxação linear (solução T). Além do atrativo de não depender de nenhum conhecimento do problema que está sendo resolvido, RINS também é simples e de fácil implementação.

4.3.1 Guided dives

O trabalho de Danna et al. [7] descreve outra técnica bastante simples e eficaz para melhorar a performance de algoritmos *branch and cut*. *Guided dives* é uma simples heurística a ser utilizada no momento em que já foi escolhida qual variável fixar para o próximo passo da busca e quando se precisa escolher em que ordem os possíveis valores de fixação vão ser explorados (e.g. quando já se decidiu fixar uma variável binária, mas se precisa decidir entre explorar primeiro a sub-árvore em que a fixação é em 1 ou primeiro a sub-árvore em que a fixação é em 0). A heurística é simples: fixar primeiro a variável em um valor igual ao valor presente na melhor solução atual (a solução S

mencionada acima). Essa simples heurística consegue melhorar sensivelmente a performance do *branch and cut* [7].

4.4 Variable Neighborhood Search (VNS)

A metaeurística VNS, proposta inicialmente por Hansen e Mladenovic [28], tem como idéia básica a troca sistemática de vizinhanças durante a busca local: a partir de uma solução inicial e uma lista de vizinhanças, escolhe-se uma solução qualquer na primeira vizinhança da solução e efetua-se busca local nesta. Se a busca não melhorar a solução inicial, repete-se o processo com a próxima vizinhança. Caso a solução melhore, volta-se à primeira vizinhança e repete-se o processo com a solução melhorada. O VNS caracteriza-se por ser uma metaeurística de fácil implementação, embora implementações mais complexas sejam necessárias para problemas mais difíceis e instâncias maiores [20].

O trabalho de Hansen, Maldenovic e Urosevic [21] propõe um algoritmo VNS para resolver problemas MIP. O algoritmo segue a linha proposta pelo algoritmo *Local Branching* (LB) de Fischetti e Lodi [12], descrito na seção 4.2: definindo vizinhanças utilizando restrições no modelo MIP e utilizando o resolvedor CPLEX para resolver as buscas locais.

Algoritmo 3 VNS Branching (VNSBra)

Entrada: limite de tempo local e total, fator de diversificação *step*, resolvedor MIP *solver*
Saída: solução x_opt

```
1:  raio = step
2:  while tempo utilizado < limite de tempo total do
3:     x_cur =VND( restante de tempo total, limite de tempo local, x_cur)
4:     if x_cur for melhor que x_opt then
5:        x_opt = x_cur
6:        raio = step
7:     else
8:        raio+ = step
9:     end if
10:    x_cur =VNS-Disk( restante de tempo total, x_cur, step, raio )
11: end while
12: return x_opt
```

O algoritmo VNS (chamado daqui em diante de VNSBra) está descrito no algoritmo 3. Começando com uma solução viável qualquer encontrada pelo resolvedor, o algoritmo consiste basicamente na repetição de três passos: um passo de intensificação (VND, na linha 3), possível atualização da melhor solução (linhas 4 a 9) e diversificação (VNS-Disk, linha 10). Dependendo da

atualização ou não da melhor solução, o passo de diversificação será mais ou menos enfatizado - esse controle é feito pela variável *raio*.

Algoritmo 4 Variable Neighborhood Descent (VND)

Entrada: limite de tempo local e total, solução atual x_cur
Saída: solução x_cur

1: $rhs = 1$
2: **while** tempo utilizado $<$ limite de tempo total **do**
3: adicionar restrição $\Delta(x, x\_cur) \leq rhs$
4: $x =$ resolver MIP (com limite de tempo local, limite superior $= x\_cur$)
5: **if** encontrada solução ótima **then**
6: reverter restrição para $\Delta(x, x\_cur) \geq rhs + 1$, $rhs = 1$, $x\_cur = x$
7: **else if** encontrada solução viável **then**
8: reverter restrição para $\Delta(x, x\_cur) \geq 1$, $rhs = 1$, $x\_cur = x$
9: **else if** MIP é provado inviável **then**
10: remove restrição, $rhs = rhs + 1$
11: **else if** nenhuma solução viável encontrada **then**
12: sair do loop
13: **end if**
14: **end while**
15: remover todas restrições adicionadas
16: **return** x_cur

A rotina de intensificação, chamada de VND e descrita no algoritmo 4, consiste em uma busca em vizinhança variada clássica. As vizinhanças são vizinhanças k-OPT, exatamente como definidas pelo LB na inequação (4-2). As vizinhanças k-OPT (partindo de 1-OPT) são examinadas de maneira sequencial e a melhor solução é mantida em x_cur. Na linha 3, adiciona-se ao modelo a restrição k-OPT (com $k = rhs$). Na linha 4, o resolvedor MIP (neste caso, o CPLEX), é usado para encontrar uma solução, com um tempo limite fixo e com um limite superior igual ao valor de x_cur (ou seja, o interesse está somente em soluções melhores que x_cur). Nas linhas 5-14 a resposta do resolvedor é tratada: se foi encontrada uma solução ótima ou viável, reverte-se a restrição e atualiza-se x_cur; se o MIP é inviável, passa-se para a próxima vizinhança; se não foi encontrada nenhuma solução viável, encerra-se a busca. Ao final da busca, na linha 15, retiram-se todas as restrições que haviam sido adicionadas.

A rotina de diversificação, chamada de VNS-Disk (descrita no algoritmo 5), simplesmente busca por uma solução viável em uma vizinhança em formato de "disco", a vizinhança formada por $v \leq \Delta(x, x\_cur) \leq v + step$. Se alguma solução é encontrada, ela é retornada. Caso contrário, o disco é movido e a busca é feita novamente. Como não há limite superior definido nesta busca, soluções piores que a solução inicial x_cur serão encontradas.

Algoritmo 5 VNS Diversification (VNS-Disk)

Entrada: limite de tempo total, solução atual x_cur, fator de diversificação $step$, raio inicial r_ini

Saída: solução diversificada x_diver

```
1: while tempo utilizado < limite de tempo total do
2:    adicionar as restrições r_ini ≤ Δ(x, x_cur) ≤ r_ini + step
3:    x = resolver MIP (escolher a primeira solução viável)
4:    remover as duas restrições adicionadas
5:    if não foi encontrada solução viável then
6:       r_ini = r_ini + step
7:    else
8:       return x_diver = x
9:    end if
10: end while
11: return x_diver
```

Devido a seus ótimos resultados em instâncias difíceis [21], o algoritmo VNSBra foi escolhido para ser implementado e testado neste trabalho ao invés do algoritmo LB.