Manual de Computação Evolutiva e Heurística

Instruções para Formatação

Sumário

Capítulo 1

Busca Tabu

CAPÍTULO 1

Busca Tabu

1. Introdução

A Busca Tabu (BT), conhecida na literatura inglesa como *Tabu Search*, é uma metaheurística de busca local originada nos trabalhos independentes de Fred Glover (Glover; 1986) e Pierre Hansen (Hansen; 1986). Gendreau (2003) e Glover and Laguna (1997) apontam, no entanto, que muitos dos componentes presentes nesta primeira proposta da Busca Tabu, bem como outros que foram incorporados posteriormente, já haviam sido introduzidas em Glover (1977).

A BT surgiu como uma técnica para guiar uma heurística de busca local tradicional na exploração do espaço de soluções além da otimalidade local, usando para isso basicamente estruturas de memória. Ela é uma das metaheurísticas mais usadas e seu sucesso decorre de sua eficiência em produzir soluções de alta qualidade para vários problemas combinatórios, entre os quais: programação de horários (Santos et al.; 2005; Souza et al.; 2004), roteirização (Archetti et al.; 2006; Cordeau et al.; 2002; Gendreau et al.; 1999, 1996) e sequenciamento (Allahverdi et al.; 2008).

Sendo uma técnica de busca local, a BT parte de uma solução inicial e se move no espaço de soluções de uma solução para outra que esteja em sua vizinhança. Além da necessidade de especificar como gerar uma solução inicial,

definir os movimentos para explorar o espaço de soluções e estabelecer o critério de parada, que são componentes típicos de técnicas de busca local, para projetar um algoritmo BT é necessário, também, especificar os seguintes componentes básicos: 1) Critério de escolha da próxima solução vizinha; 2) Seleção dos atributos do movimento; 3) Memória de curto prazo para armazenar as regras de proibição (Lista Tabu); 4) Número de iterações que um atributo selecionado permanecerá proibido (Tamanho da Lista) e 5) Critério de aspiração.

Um algoritmo BT com esses componentes básicos é o mais difundido, sendo suficiente para resolver satisfatoriamente muitos problemas combinatórios (Glover and Laguna; 1997). Entretanto, a Busca Tabu não se resume apenas a isso. Algoritmos BT mais sofisticados incluem, também, o uso de memórias de longo prazo para a aplicação de estratégias de intensificação e/ou diversificação, bem como de Reconexão por Caminhos e Oscilação Estratégica, o que proporciona à técnica mais recursos para explorar melhor o espaço de soluções.

Um grande número de publicações descrevendo a Busca Tabu é encontrado na literatura. Entre essas, destacamos: Gendreau (2003, 2002); Glover and Laguna (1997); Glover et al. (1993); Glover and Laguna (1993); Hertz and de Werra (1990); Glover (1990, 1989); de Werra and Hertz (1989).

O restante deste capítulo está organizado como segue. Na Seção 2 ilustra-se o funcionamento de um algoritmo básico BT considerando um problema simples, mas de muita aplicabilidade, o da Mochila 0-1. Nessa seção, os componentes típicos da BT são progressivamente incluídos e justificados. Na Seção 3 é apresentado o pseudocódigo de um algoritmo básico BT. Na Seção 4 são apresentados exemplos de regras de proibição para alguns problemas combinatórios. Na Seção 5 são mostradas estratégias para evitar a análise da vizinhança completa de uma solução. Na Seção 6 discute-se sobre a implementação eficiente da Lista Tabu, enquanto na Seção 7 relata-se sobre o tamanho ideal dessa lista. Na Seção 8 são listados os critérios de aspiração normalmente adotados nas implementações baseadas em BT. Na Seção 9 são detalhadas as memórias de longo prazo baseadas em frequência de transição e residência. Na Seção 10 apresenta-se a técnica Oscilação Estratégica, enquanto na última Seção são listadas as referências usadas neste capítulo. A estratégia Reconexão por Caminhos, apesar de normalmente fazer parte de uma implementação baseada em Busca Tabu, não será apresentada, uma vez que está detalhada no capítulo deste livro sobre a técnica GRASP.

2. Funcionamento de um algoritmo BT

Para ilustrar o funcionamento de um algoritmo BT, consideraremos uma instância do PM01 (Problema da Mochila 0-1). Nesse problema, há uma mochila de capacidade b, um conjunto de n itens j, cada qual com um peso w_j e um valor de retorno p_j caso o item j seja alocado à mochila. O objetivo é escolher os itens que devem ser alocados à mochila de forma que o valor de retorno total seja o maior possível, respeitando-se a capacidade da mochila.

Matematicamente, o PM01 pode ser representado pelo modelo de programação linear inteira definido pelas equações (1.1)-(1.3).

$$\max f(s) = \sum_{j=1}^{n} p_j s_j \tag{1.1}$$

$$\sum_{j=1}^{n} w_j s_j \leq b \tag{1.2}$$

$$s_j \in \{0,1\}\ \forall j = 1, \cdots, n \tag{1.3}$$

em que s_j é uma variável binária que assume valor 1 se o item j for alocado à mochila e 0, caso contrário.

A Eq. (1.1) representa a função objetivo, a qual visa à maximização do valor de retorno dos itens alocados à mochila. A restrição (1.2) impede que a capacidade da mochila seja superada, enquanto as restrições (1.3) definem o tipo das variáveis de decisão.

Como exemplo do PM01, seja uma mochila de capacidade $b = 32$ e os dados da Tabela 1.1.

Tabela 1.1: Instância do Problema da Mochila 0-1

Item j	1	2	3	4	5	6	7	8
Peso w_j	4	15	7	9	8	10	9	11
Retorno p_j	2	2	3	4	6	5	8	7

Como é uma técnica de busca local, a BT explora o espaço de soluções S indo de uma solução $s \in S$ a outra s' que esteja em sua vizinhança $V \subseteq N(s)$. A solução $s' \in N(s)$ é dita vizinha de s se ela for obtida pela aplicação de um movimento m a s, operação esta representada por $s' \leftarrow s \oplus m$.

Em muitos problemas, a solução vizinha é alcançada por meio de vários tipos de movimentos. Nesse caso, $N(s) = \bigcup_{i \in M} N^i(s)$, sendo $M = \bigcup m_i$ o conjunto dos diferentes movimentos m_i.

No Problema da Mochila 0-1, uma solução s pode ser representada por um vetor binário n-dimensional $s = (s_1, s_2, \cdots, s_n)$, em que cada componente $s_j \in \{0,1\}$ assume valor 1 se o item j for alocado à mochila e 0, caso contrário. Para explorar o espaço de soluções deste problema, um movimento m natural é considerar a inversão do valor de um bit. Desta maneira, a vizinhança $N(s)$ de uma solução s do PM01 consiste no conjunto dos vizinhos s' que diferem de s pelo valor de um único bit. Consideraremos, nesta definição de vizinhança, que *todas* as soluções vizinhas serão analisadas, isto é, que $V = N(s)$. Na Seção 5 é apresentada uma estratégia para evitar a análise da vizinhança completa.

É necessário, agora, definir uma função de avaliação para guiar o procedimento de busca no espaço de soluções do problema. Considerando que se deseja maximizar o valor de retorno trazido pela utilização dos itens alocados à mochila, há duas possibilidades para a escolha dessa função.

A primeira é considerar a exploração apenas no espaço de soluções factíveis. Neste caso, a função de avaliação coincide com a própria função objetivo do problema, dada pela Eq. (1.1), e a solução s satisfaz à condição de que a capacidade da mochila seja respeitada, isto é, que a restrição (1.2) não seja violada.

Outra possibilidade é permitir a geração de soluções infactíveis. Neste caso, ao invés de utilizar a função f da Eq. (1.1), avaliaríamos uma solução s pela função auxiliar g, baseada em penalidade, dada pela Eq. (1.4):

$$g(s) = \sum_{j=1}^{n} p_j s_j - \rho \times \max\{0, \sum_{j=1}^{n} w_j s_j - b\} \tag{1.4}$$

sendo ρ uma penalidade.

A primeira parcela dessa função de avaliação g é a função objetivo f propriamente dita do PM01, enquanto a segunda é uma penalização que tem como objetivo desestimular a colocação na mochila de itens que ultrapassam sua capacidade. Como a função g deve ser maximizada, o sinal desta segunda parcela é negativo de forma a não incentivar a realização de movimentos que gerem soluções infactíveis. O valor de ρ deve ser suficientemente grande para atender a esse objetivo. Para os dados da Tabela 1.1, pode-se tomar, por exemplo, $\rho = \sum_{j=1}^{n} p_j = 15$.

Para o que se segue, será permitida apenas a geração de soluções factíveis. No entanto, pode ser conveniente alternar a exploração do espaço de soluções entre soluções factíveis e infactíveis e, nesse caso, a função g dada pela Eq. (1.4) deve ser usada. Considerações acerca dessa estratégia são feitas na Seção 10.

Consideremos um exemplo do PM01 cuja solução inicial seja $s^0 = (1, 0, 0, 1, 0, 1, 1, 0)$, obtida de forma aleatória. A essa solução estão associados os valores $f(s^0) = 19$ e $peso(s^0) = 32$, correspondentes à aplicação da função objetivo f dada pela Eq. (1.1) e da função $peso(s) = \sum_{j=1}^{n} w_j s_j$, respectivamente.

Na Tabela 1.2 estão representados todos os vizinhos segundo a definição de vizinhança adotada. Na primeira coluna desta tabela representa-se o número do vizinho; na segunda, o vizinho; na terceira, seu peso e na quarta, o valor de sua função de avaliação. O sinal "–" indica que a respectiva solução é infactível. Números em negrito indicam o bit modificado.

Tabela 1.2: Vizinhança da solução s^0

viz.	$s' \in N(s^0)$	$peso(s')$	$f(s')$
1	$(\mathbf{0}, 0, 0, 1, 0, 1, 1, 0)$	28	17
2	$(1, \mathbf{1}, 0, 1, 0, 1, 1, 0)$	47	–
3	$(1, 0, \mathbf{1}, 1, 0, 1, 1, 0)$	39	–
4	$(1, 0, 0, \mathbf{0}, 0, 1, 1, 0)$	23	15
5	$(1, 0, 0, 1, \mathbf{1}, 1, 1, 0)$	40	–
6	$(1, 0, 0, 1, 0, \mathbf{0}, 1, 0)$	22	14
7	$(1, 0, 0, 1, 0, 1, \mathbf{0}, 0)$	23	11
8	$(1, 0, 0, 1, 0, 1, 1, \mathbf{1})$	43	–

A solução s^0 representa um ótimo local em relação à vizinhança considerada, uma vez que todos os seus vizinhos têm avaliação pior. Uma heurística de busca local convencional pararia nesse ponto. No entanto, a BT aceita soluções de não-melhora como estratégia para ir além desse ótimo. No exemplo considerado,

escolheremos o melhor membro $s' \in N(s^0)$ como *critério de escolha do próximo vizinho*, no caso, o primeiro. Desta forma, ao final da primeira iteração, a nova solução corrente passa a ser $s^1 = (0,0,0,1,0,1,1,0)$, com $f(s^1) = 17$, sendo o valor da melhor solução dada por $f(s^\star) = 19$.

Vejamos o que aconteceria caso prosseguíssemos com a mesma estratégia para a vizinhança da solução corrente. Seus vizinhos estão explicitados na Tabela 1.3.

Tabela 1.3: Vizinhança da solução s^1

viz.	$s' \in N(s^1)$	$peso(s')$	$f(s')$
1	$(\mathbf{1},0,0,1,0,1,1,0)$	32	19
2	$(0,\mathbf{1},0,1,0,1,1,0)$	43	–
3	$(0,0,\mathbf{1},1,0,1,1,0)$	35	–
4	$(0,0,0,\mathbf{0},0,1,1,0)$	19	13
5	$(0,0,0,1,\mathbf{1},1,1,0)$	36	–
6	$(0,0,0,1,0,\mathbf{0},1,0)$	18	12
7	$(0,0,0,1,0,1,\mathbf{0},0)$	19	9
8	$(0,0,0,1,0,1,1,\mathbf{1})$	39	–

Pela Tabela 1.3, a solução $s' = (1,0,0,1,0,1,1,0)$, com $f(s') = 19$, é o melhor vizinho de s^1. Se movêssemos para essa solução, obtendo $s^2 \leftarrow s'$, voltaríamos à solução inicial s^0 e, posteriormente, prosseguiríamos novamente em direção à s^1, caso repetíssimos a mesma estratégia anterior. Isto é, o algoritmo ciclaria usando-se unicamente a estratégia de mover-se para o melhor vizinho.

Para evitar a ciclagem, ou seja, a geração de uma mesma sequência de soluções já visitadas anteriormente, um algoritmo BT usa uma estrutura de memória, a chamada Lista Tabu, assim chamada a lista de soluções "proibidas". Sempre que uma solução é gerada, ela é colocada na Lista Tabu T. Assim, à medida que uma nova solução é gerada, a lista é aumentada. No exemplo dado, ao final da primeira iteração teríamos $T^1 = \{s^0\}$, indicando que s^0 não seria candidato ao melhor vizinho de s^1, por já ter sido gerado anteriormente. A nova solução agora deve ser $s^2 = (0,0,0,0,0,1,1,0)$, uma vez que este é o vizinho não tabu de melhor avaliação. Com esta estratégia de memória, o algoritmo BT pode ir além do ótimo local e acessar outras regiões do espaço de soluções.

Uma Lista Tabu clássica armazena $|T|$ soluções e funciona numa estrutura de fila de tamanho fixo, isto é, quando a lista está cheia e uma nova solução entra, a mais antiga sai. Essa estratégia está fundamentada no fato de que na exploração do espaço de soluções, as soluções geradas há mais tempo possivelmente estão "distantes" da região do espaço sob análise e, como tal, não têm influência na escolha da próxima solução vizinha naquela região. Desta forma, armazendo-se apenas as soluções mais "próximas" da solução atual, a ciclagem estará evitada nessa região. Por armazenar apenas as soluções mais recentes, essa Lista Tabu é referenciada como uma memória de curto prazo (*short-term memory*). Uma lista de tamanho $|T|$, no entanto, somente impede ciclos de até $|T|$ soluções. Isso significa que, se na k-ésima iteração a lista tabu é $T = \{s^{k-r}, \cdots, s^{k-2}, s^{k-1}\}$, então, quando $r = k$, a ciclagem estará completamente evitada. Entretanto, na iteração seguinte, s^{k-r} sai da lista, podendo retornar caso seja novamente selecionada pelo critério de escolha do próximo vizinho. A definição do tamanho

da lista é, pois, um parâmetro crítico da BT. A dimensão da lista não pode ser tão pequena, sob pena de haver ciclagem; nem tão grande, para armazenar desnecessariamente soluções que não estejam ligadas à história recente da busca. Uma ideia do tamanho da lista tabu será discutida mais adiante, na Seção 7.

Em muitas aplicações reais, no entanto, além de requerer muito espaço em memória, é muito dispendioso computacionalmente avaliar se uma solução está ou não presente na Lista Tabu. Por exemplo, em um problema de programação de tripulações (*crew scheduling*), uma solução é uma escala de trabalho, isto é, um conjunto de jornadas, cada qual representada por um conjunto de viagens a serem realizadas pelos tripulantes. Verificar se uma dada escala está ou não em uma Lista Tabu de soluções envolveria comparar as jornadas que a compõem com aquelas das escalas armazenadas em T. Essa operação poderia ser simplificada, comparando-se primeiramente o valor da função de avaliação da escala dada com o das escalas da lista. Sendo diferente, então a referida escala não seria tabu; caso contrário, o segundo passo da comparação consistiria em analisar o custo das jornadas. Igualmente, havendo algum custo diferente, concluiríamos que a escala corrente não é tabu. No entanto, no caso de todos os custos serem iguais, então as jornadas seriam comparadas viagem por viagem. Mesmo essa alternativa de comparação é ainda muito custosa computacionalmente.

Por outro lado, quando se move de uma solução para outra que esteja em sua vizinhança, na realidade, a solução vizinha difere muito pouco da solução corrente, mais precisamente, apenas do movimento realizado. Em vista disso e das considerações anteriores, ao invés de armazenar toda uma solução na lista, guarda-se nela apenas alguma regra de proibição associada a um atributo (característica) da solução ou movimento realizado para impedir o retorno a uma solução já gerada anteriormente. O atributo selecionado é dito tabu-ativo e uma solução que contém atributos tabu-ativos torna-se tabu. Um movimento é dito tabu se ele der origem a uma solução tabu. Referenciar-nos-emos a tal lista como Lista Tabu baseada em atributos. Na Seção 4 são mostradas algumas regras de proibição para problemas combinatórios que usam movimentos de inserção e troca.

Voltando ao PM01, quando se muda para um novo vizinho, este difere da solução corrente apenas no valor do bit modificado. Por exemplo, da solução s^0 para a solução s^1, o único bit alterado é o primeiro. Então, nada mais natural do que considerar como atributo a posição do bit na solução e como regra de proibição, a inversão do valor do bit dessa posição. Desta forma, a posição 1 torna-se tabu ativa e o movimento que altera o valor do bit desta posição é dito tabu pois gera uma solução tabu (a solução s^0).

Considerando uma lista tabu de tamanho 2, isto é, $|T| = 2$, na primeira iteração (Tabela 1.2) proibiríamos a inversão do primeiro bit, representando esta operação por $T = \{\langle 1\rangle\}$. Com isso, todos os vizinhos obtidos de s^1 pela alteração do primeiro bit estarão proibidos. Desta forma, o retorno à s^0 fica impedido. Na segunda iteração (Tabela 1.3), o melhor vizinho não tabu de s^1 é obtido pela alteração do quarto bit, então $T = \{\langle 1\rangle, \langle 4\rangle\}$ ao final desta iteração. Agora, está proibida a geração de vizinhos de s^2 que diferem desta solução pela alteração no valor do primeiro ou quarto bit.

Passemos, agora, à terceira iteração, procurando a próxima solução vizinha de s^2. A Tabela 1.4 relaciona os dados da vizinhança completa desta solução.

Tabela 1.4: Vizinhança da solução s^2

viz.	$s' \in N(s^2)$	$peso(s')$	$f(s')$
1^t	$(\mathbf{1},0,0,0,0,1,1,0)$	23	15
2	$(0,\mathbf{1},0,0,0,1,1,0)$	34	–
3	$(0,0,\mathbf{1},0,0,1,1,0)$	26	16
4^t	$(0,0,0,\mathbf{1},0,1,1,0)$	28	17
5	$(0,0,0,0,\mathbf{1},1,1,0)$	27	19
6	$(0,0,0,0,0,\mathbf{0},1,0)$	9	8
7	$(0,0,0,0,0,1,\mathbf{0},0)$	10	5
$8^\star$	$(0,0,0,0,0,1,1,\mathbf{1})$	30	20

Na Tabela 1.4, os vizinhos 1 e 4 (assinalados com um t na primeira coluna) estão proibidos de serem acessados, uma vez que o vizinho 1 é alcançado alterando-se o primeiro bit e o vizinho 4, o quarto bit, e ambos pertencem à Lista Tabu T. Como se observa, o melhor vizinho não tabu (assinalado com um asterisco) é o último, dado por $s' = (0,0,0,0,0,1,1,1)$, o qual passa a ser a nova solução corrente s^3. Essa solução é melhor que a melhor solução gerada até então, pois $f(s^3) = 20 > f(s^\star) = 19$. Assim, a melhor solução deve ser atualizada. A estratégia de mover para o melhor vizinho, ainda que esse seja de pior avaliação, associada ao uso de memória, proporcionou à busca a possibilidade de prosseguir além do ótimo local s^0 e encontrar melhores soluções, sem a ocorrência de ciclagem. Como definimos $|T| = 2$ e a lista tabu já tem dois atributos tabu-ativos, então o atributo tabu mais antigo, $\langle 1\rangle$, sai para a entrada de $\langle 8\rangle$, resultando em $T = \{\langle 4\rangle, \langle 8\rangle\}$ ao final desta iteração.

A lista tabu baseada em atributos se, por um lado, visa à eliminação de ciclagem (uma vez que ela garante o não retorno, por $|T|$ iterações, a uma solução já visitada anteriormente); por outro, pode ser restritiva (de Werra and Hertz; 1989). De fato, aplicando-se o movimento tabu $\langle 4\rangle$ à solução s^3 do PM01, obtém-se a solução $s' = (0,0,0,1,0,1,1,1)$ que ainda não foi gerada e, dessa forma, não deveria estar proibida. Em outras palavras, uma lista tabu baseada em atributos pode não somente impedir o retorno a uma solução já gerada anteriormente, como também impedir que algumas soluções do espaço de busca sejam alcançadas.

De forma a contornar essa situação, aplica-se o chamado *critério de aspiração*. Um critério de aspiração é um mecanismo que retira, sob certas circunstâncias, a classificação tabu de um movimento. Um exemplo simples de aplicação dessa ideia é executar um movimento tabu somente se ele conduzir a um vizinho melhor que $s^\star$. Esse é o chamado critério de aspiração por objetivo global. Ele se fundamenta no fato de que se um movimento tabu conduz a uma solução com valor melhor que o da melhor solução encontrada até então, é sinal de que ela ainda não foi gerada. Outros critérios de aspiração são discutidos na Seção 8.

Nas Tabelas 1.5 a 1.10 são registradas as características das soluções geradas pela Busca Tabu aplicada ao PM01 durante as próximas 6 iterações. Em cada

tabela são mostrados os dados dos vizinhos, o valor da solução corrente, o valor da melhor solução encontrada até então, bem como a lista tabu ao final da iteração. Considerou-se como critério de parada 3 iterações sem melhora.

Tabela 1.5: Vizinhança de s^3

viz.	$s' \in N(s^3)$	$peso(s')$	$f(s')$
1	$(\mathbf{1},0,0,0,0,1,1,1)$	34	–
2	$(0,\mathbf{1},0,0,0,1,1,1)$	45	–
3	$(0,0,\mathbf{1},0,0,1,1,1)$	37	–
4^t	$(0,0,0,\mathbf{1},0,1,1,1)$	39	–
5	$(0,0,0,0,\mathbf{1},1,1,1)$	38	–
$6^\star$	$(0,0,0,0,0,\mathbf{0},1,1)$	20	15
7	$(0,0,0,0,0,1,\mathbf{0},1)$	21	12
8^t	$(0,0,0,0,0,1,1,\mathbf{0})$	19	13

$T = \{\langle 8\rangle, \langle 6\rangle\}$; $f(s^4) = 15$; $f(s^\star) = 20$

Tabela 1.6: Vizinhança de s^4

viz.	$s' \in N(s^4)$	$peso(s')$	$f(s')$
1	$(\mathbf{1},0,0,0,0,0,1,1)$	24	17
2	$(0,\mathbf{1},0,0,0,0,1,1)$	35	–
3	$(0,0,\mathbf{1},0,0,0,1,1)$	27	18
4	$(0,0,0,\mathbf{1},0,0,1,1)$	29	19
$5^\star$	$(0,0,0,0,\mathbf{1},0,1,1)$	28	21
6^t	$(0,0,0,0,0,\mathbf{1},1,1)$	30	20
7	$(0,0,0,0,0,0,\mathbf{0},1)$	11	7
8^t	$(0,0,0,0,0,0,1,\mathbf{0})$	9	8

$T = \{\langle 6\rangle, \langle 5\rangle\}$; $f(s^5) = 21$; $f(s^\star) = 21$

Tabela 1.7: Vizinhança de s^5

viz.	$s' \in N(s^5)$	$peso(s')$	$f(s')$
$1^\star$	$(\mathbf{1},0,0,0,1,0,1,1)$	32	23
2	$(0,\mathbf{1},0,0,1,0,1,1)$	43	–
3	$(0,0,\mathbf{1},0,1,0,1,1)$	35	–
4	$(0,0,0,\mathbf{1},1,0,1,1)$	37	–
5^t	$(0,0,0,0,\mathbf{0},0,1,1)$	20	15
6^t	$(0,0,0,0,1,\mathbf{1},1,1)$	38	–
7	$(0,0,0,0,1,0,\mathbf{0},1)$	19	13
8	$(0,0,0,0,1,0,1,\mathbf{0})$	17	14

$T = \{\langle 5\rangle, \langle 1\rangle\}$; $f(s^6) = 23$; $f(s^\star) = 23$

Tabela 1.8: Vizinhança de s^6

viz.	$s' \in N(s^6)$	$peso(s')$	$f(s')$
1^t	$(\mathbf{0},0,0,0,1,0,1,1)$	28	21
2	$(1,\mathbf{1},0,0,1,0,1,1)$	47	–
3	$(1,0,\mathbf{1},0,1,0,1,1)$	39	–
4	$(1,0,0,\mathbf{1},1,0,1,1)$	41	–
5^t	$(1,0,0,0,\mathbf{0},0,1,1)$	24	17
6	$(1,0,0,0,1,\mathbf{1},1,1)$	42	–
7	$(1,0,0,0,1,0,\mathbf{0},1)$	23	15
$8^\star$	$(1,0,0,0,1,0,1,\mathbf{0})$	21	16

$T = \{\langle 1\rangle, \langle 8\rangle\}$; $f(s^7) = 16$; $f(s^\star) = 23$

Tabela 1.9: Vizinhança de s^7

viz.	$s' \in N(s^7)$	$peso(s')$	$f(s')$
1^t	$(\mathbf{0},0,0,0,1,0,1,0)$	17	14
2	$(1,\mathbf{1},0,0,1,0,1,0)$	36	–
3	$(1,0,\mathbf{1},0,1,0,1,0)$	28	19
4	$(1,0,0,\mathbf{1},1,0,1,0)$	30	20
5	$(1,0,0,0,\mathbf{0},0,1,0)$	13	10
$6^\star$	$(1,0,0,0,1,\mathbf{1},1,0)$	31	21
7	$(1,0,0,0,1,0,\mathbf{0},0)$	12	8
8^t	$(1,0,0,0,1,0,1,\mathbf{1})$	32	23

$T = \{\langle 8\rangle, \langle 6\rangle\}$; $f(s^8) = 21$; $f(s^\star) = 23$

Tabela 1.10: Vizinhança de s^8

viz.	$s' \in N(s^8)$	$peso(s')$	$f(s')$
$1^\star$	$(\mathbf{0},0,0,0,1,1,1,0)$	27	19
2	$(1,\mathbf{1},0,0,1,1,1,0)$	46	–
3	$(1,0,\mathbf{1},0,1,1,1,0)$	38	–
4	$(1,0,0,\mathbf{1},1,1,1,0)$	40	–
5	$(1,0,0,0,\mathbf{0},1,1,0)$	23	15
6^t	$(1,0,0,0,1,\mathbf{0},1,0)$	21	16
7	$(1,0,0,0,1,1,\mathbf{0},0)$	22	13
8^t	$(1,0,0,0,1,1,1,\mathbf{1})$	42	–

$T = \{\langle 6\rangle, \langle 1\rangle\}$; $f(s^9) = 19$; $f(s^\star) = 23$

A melhor solução obtida ao final das nove primeiras iterações da Busca Tabu foi $s^\star = s^6 = (1,0,0,0,1,0,1,1)$, com valor $f(s^\star) = 23$, sendo ela alcançada na sexta iteração.

Ilustra-se, pela Figura 1.1, a evolução do valor da função objetivo do PM01 ao longo das iterações realizadas. A Figura 1.2, por sua vez, mostra uma situação de ciclagem em Busca Tabu. Observa-se que, a partir da quinta iteração, aparece uma sequência de valores para a função objetivo que se repete de 6 em 6 iterações.

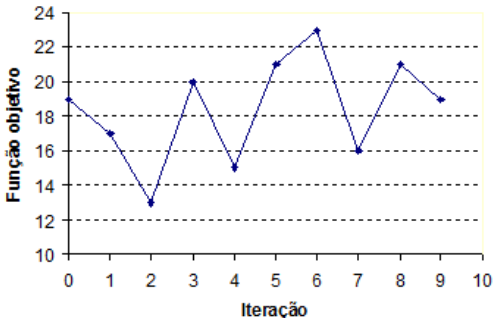

Figura 1.1: Evolução da função objetivo

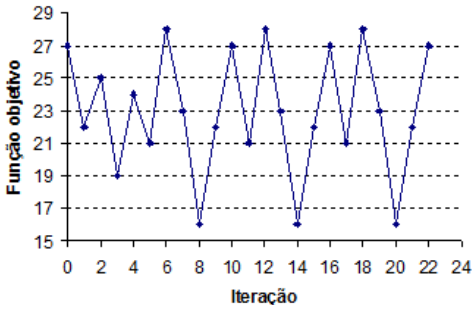

Figura 1.2: Exemplo de ciclagem

3. O Algoritmo Busca Tabu

O Algoritmo 1 resume as ideias básicas apresentadas da Busca Tabu para um problema de minimização. Os parâmetros principais de controle do algoritmo são a cardinalidade $|T|$ da lista tabu, o critério de aspiração, a cardinalidade do conjunto V de soluções vizinhas testadas em cada iteração e a regra de parada.

Algorithm 1 Busca Tabu

1: **Entrada**: $f(.), N(.), |T|, |V|, s$
2: **Saída**: $s^\star$
3: $s^\star \leftarrow s$ {Melhor solução encontrada até então}
4: $Iter \leftarrow 0$ {Contador do número de iterações}
5: $MelhorIter \leftarrow 0$ {Iteração mais recente que forneceu $s^\star$}
6: $T \leftarrow \emptyset$ {Lista Tabu}
7: **while** Critério de parada não satisfeito **do**
8: $Iter \leftarrow Iter + 1$
9: Seja $s' \leftarrow s \oplus m$ o melhor elemento de $V \subseteq N(s)$ tal que o movimento m não seja tabu ($m \notin T$) ou s' atenda a um critério de aspiração
10: Atualize a lista tabu T
11: $s \leftarrow s'$
12: **if** $f(s) < f(s^\star)$ **then**
13: $s^\star \leftarrow s$
14: $MelhorIter \leftarrow Iter$
15: **end if**
16: **end while**
17: Retorne $s^\star$

As regras de parada comumente utilizadas são: (a) o número de iterações sem melhora na solução global ($iter$ - $MelhorIter$) atinge um valor máximo $BTmax$ e (b) o tempo de processamento atinge um tempo máximo permitido.

4. Exemplos de regras de proibição

Ilustraremos, nesta seção, algumas regras de proibição comumente aplicadas a problemas que envolvem permutação de elementos de uma solução.

Seja uma permutação (solução) s de n elementos. Em um movimento de troca, um elemento i da posição s_i é permutado com o elemento j da posição s_j, enquanto em um movimento de inserção, um elemento i da posição s_i é inserido na posição s_j.

Exemplo: Dada a solução $s = (2, 6, 1, 5, 4, 3)$, a solução $s' = (2, 6, 1, \mathbf{3}, 4, \mathbf{5})$ é obtida de s pela permutação do elemento 5, da quarta posição, com o elemento 3, da sexta posição. Já a solução $s' = (2, 6, 1, 4, 3, \mathbf{5})$ é obtida de s inserindo-se o elemento 5, da quarta posição, na sexta posição.

Um atributo pode ser definido pelo par ⟨elemento, posição do elemento⟩. Considerando a troca do elemento i da posição s_i com o elemento j da posição s_j ou que o elemento i é inserido na posição s_j, podemos ter as seguintes regras de proibição para movimentos de troca e inserção (França; 2009):

1. Impedir movimentos que resultem em uma permutação em que i ocupe a posição s_i e j ocupe a posição s_j. Para o exemplo considerado, supondo a troca de $i = 5$ com $j = 3$, então está proibido que o elemento 5, da quarta posição, ocupe a sexta posição, assim como o elemento 3, da sexta posição, ocupe a quarta posição.

2. Impedir movimentos que resultem em uma permutação em que i ocupe a posição s_i ou j ocupe a posição s_j. Para o exemplo considerado, supondo a troca de $i = 5$ com $j = 3$, então está proibido que o elemento 5, da quarta posição, ocupe a sexta posição ou, então, que o elemento 3, da sexta posição ocupe a quarta posição.

3. Impedir que o elemento i retorne à posição s_i. No exemplo dado, está proibido que o elemento 5 retorne à quarta posição.

4. Impedir que o elemento i mova para posições $k \leq s_i$. No exemplo considerado, ficaria proibido que o elemento 5 movesse para qualquer posição menor ou igual à quarta.

5. Impedir que o elemento i se mova para posições $k \leq s_j$. No exemplo referenciado, estaria impedida a mudança do elemento 5 para posições inferiores ou igual à sexta. Em outras palavras, neste caso, esta regra impediria qualquer movimentação do elemento 5, uma vez que a sexta posição é a última.

6. Impedir que o elemento i se mova. Para o exemplo dado, o elemento 5 estaria proibido de se mover.

7. Impedir que os elementos i e j troquem de posição com quaisquer outros elementos. No exemplo considerado, ficaria proibida a troca de posição do elemento 5 com qualquer outro elemento, bem como a troca do elemento 3 com qualquer outro.

8. Impedir que os elementos i e j se movam. No exemplo dado, isto significa que o elemento 5 está proibido de se mover para qualquer outra posição, assim como o elemento 3.

As regras anteriores estão listadas em ordem crescente de restritividade, isto é, as primeiras regras impedem a geração de um conjunto menor de soluções, enquanto as últimas proíbem um número bem maior de soluções.

O próximo exemplo considera um problema de alocação de aulas a salas (*Classroom Assignment Problem*). Nesse problema, há um conjunto $H = \{1, 2, 3, 4\}$ de horários para a realização das aulas, um conjunto $S = \{1, 2, 3\}$ de salas nas quais as aulas são ministradas e um conjunto $M = \{A, B, C\}$ de aulas a serem ministradas, cujos horários são previamente fixados. A Tabela 1.11 mostra uma solução s, enquanto a Tabela 1.12 apresenta uma solução s' obtida de s por meio de um movimento de troca de sala envolvendo duas aulas distintas.

Tabela 1.11: solução s

	Salas		
Horário	1	2	3
1	A		
2			D
3		C	D
4	B	C	

Tabela 1.12: solução s'

	Salas		
Horário	1	2	3
1	A		
2		D	
3		D	C
4	B		C

Nas tabelas 1.11 e 1.12, cada célula s_{ij} representa a aula ministrada no horário i e sala j. Por exemplo, na solução s, a aula C é ministrada na sala 2 durante os horários 3 e 4. Como atributo do movimento de troca, podemos considerar o par $\langle i, j \rangle$, indicando uma aula (definida por seu horário i de início) e sua sala j de realização. As seguintes regras de proibição são possíveis, dentre outras: (a) Impedir a troca das aulas envolvendo as salas j_1 e j_2, com início no horário i, sendo $i = \min\{i_1, i_2\}$, i_1 o horário de início da aula da sala j_1 e i_2 o da sala j_2; (b) Impedir que a aula da sala j_1, iniciada no horário i_1, seja mudada para a sala j_2; (c) Impedir que a aula da sala j_1, com início no horário i_1, seja mudada; (d) Aplicar uma das regras anteriores, associadas ao valor da função objetivo antes da execução do movimento.

Para uma classe de problemas de sequenciamento em uma máquina com penalidades por antecipação e atraso da produção, Wan and Yen (2002) desenvolveram um algoritmo baseado em Busca Tabu que faz uso de movimentos de troca envolvendo apenas tarefas adjacentes na ordem de produção. Os autores estabeleceram como atributo tabu a tripla $\langle i, j, fo \rangle$, em que i e j são as tarefas envolvidas na troca e fo é o valor da função objetivo da solução antes do movimento realizado.

Exemplos de regras de proibição para os problemas de Programação de Horários, Generalizado de Atribuição e n-rainhas são apresentados na Seção 9.

5. Lista de Candidatos

Para muitos problemas combinatórios, a exploração da vizinhança completa é muito custosa computacionalmente. Para esses casos, é essencial utilizar uma Lista de Candidatos.

Em uma Lista de Candidatos, a análise da vizinhança N da solução corrente é restringida a um subconjunto V dessa (linha 9 do Algoritmo 1). Tipicamente, são analisadas apenas as soluções "promissoras", assim referenciadas as soluções que satisfazem a um determinado critério.

Para exemplificar tal estratégia, consideremos o problema de sequenciamento em uma máquina tendo como objetivo a minimização do atraso total. Neste problema, são dados um conjunto de n tarefas, o tempo de processamento p_j da tarefa j e a data de entrega d_j da tarefa j. Como variáveis de decisão tem-se C_j, representando o instante de término da tarefa j, e o atraso T_j dessa tarefa, definido por $T_j = \max\{0, C_j - d_j\}$. O problema consiste, pois, em determinar uma sequência s que minimize a Eq. (1.5):

$$T = \sum_{j=1}^{n} \max\{0, C_j - d_j\} \tag{1.5}$$

Utilizando-se apenas movimentos de troca, tem-se uma vizinhança de tamanho $n(n-1)/2$, ou seja, a análise da vizinhança inteira tem complexidade $O(n^2)$. Seja $\Delta = f(s') - f(s)$ o valor de um movimento e uma instância deste problema, extraída de Glover and Laguna (1997), com os seguintes dados: $p = (6, 4, 8, 2, 10, 3)$ e $d = (9, 12, 15, 8, 20, 22)$. Dada a sequência $s = (1, 2, 3, 4, 5, 6)$, tem-se que o atraso total é $T = 36$.

A Tabela 1.13 mostra a análise da vizinhança completa, a qual envolve 15 soluções vizinhas. Nesta tabela, mostra-se, também, o valor Δ do movimento, o atraso total T da solução vizinha, bem como o valor da diferença entre as datas de entrega das tarefas i e j envolvidas na troca.

Tabela 1.13: Vizinhança de um problema de sequenciamento de tarefas

Tarefas		Δ	T	$\lvert d_i - d_j \rvert$
1	2	1	37	**3**
1	3	6	42	6
1	4	-4	32	**1**
1	5	21	57	11
1	6	4	40	13
2	3	3	39	**3**
2	4	-6	30	4
2	5	20	56	8
2	6	7	43	10
3	4	-6	30	7
3	5	4	40	5
3	6	-6	30	7
4	5	8	44	12
4	6	3	39	14
5	6	-7	29	**2**

Caso fosse criada uma Lista de Candidatos pela regra $|d_i - d_j| \leq 3$, teríamos analisado apenas 4 soluções vizinhas (os movimentos que dão origem a essas soluções estão destacados em negrito na Tabela 1.13). Dentre essas, a melhor

solução (obtida a partir da troca entre as tarefas 5 e 6), estaria incluída. Em outras palavras, com uma regra simples, a de que uma solução vizinha só é analisada se a diferença entre as datas de entrega das tarefas envolvidas na troca for menor ou igual a 3 unidades, a vizinhança é substancialmente reduzida. O valor 3 estabelecido para a distância máxima entre as datas de entrega é, obviamente, um parâmetro a ser considerado.

Ao invés de restringir a análise da vizinhança às soluções "promissoras", poder-se-ia, simplesmente, reduzir a vizinhança. Por exemplo, em problemas de alocação de aulas a salas (*classroom assignment problem*), que envolve um conjunto de salas, um conjunto de turmas e um conjunto de dias da semana para a realização das aulas, uma Lista de Candidatos pode ser um dia da semana. Nesta situação, pode-se aplicar uma estratégia do tipo: na primeira iteração da BT utilizam-se como vizinhança soluções obtidas a partir de movimentos envolvendo apenas as aulas da segunda-feira; na segunda iteração, aulas da terça-feira e assim por diante.

Em problemas de programação de horários em escolas (*school timetabling problem*), poder-se-ia, de forma análoga, considerar na primeira iteração da BT uma vizinhança envolvendo apenas as alocações dos primeiros k professores; na segunda, dos próximos k professores e assim por diante, até atingir os k últimos professores, situação na qual se voltaria para a análise da vizinhança envolvendo os primeiros k professores.

6. Implementação eficiente da Lista Tabu

Na prática, normalmente não se implementa uma Lista Tabu na estrutura de uma lista tal como apresentado anteriormente, pois, quase sempre, é possível utilizar uma estrutura de dados mais eficiente. Para mostrar isso, relembremos que no PM01 consideramos como movimento tabu a inversão do valor do bit relativo a um dado item. Isso significa que a cada iteração da Busca Tabu demandamos um máximo de $O(|T|)$ operações para verificar se um movimento é tabu ou não, sendo $|T|$ a cardinalidade da lista. A complexidade dessa verificação pode ser reduzida a $O(1)$ se utilizarmos uma estrutura especial, que armazene apenas o tempo em que o movimento tabu permanecerá na lista. Esse tempo de permanência é conhecido como duração tabu (*tabu tenure*).

Assim, no problema da mochila mencionado, ao invés de armazenarmos uma lista com os movimentos tabus, poderíamos utilizar um vetor T de n posições, com cada célula $T(j)$ representando a iteração até a qual a inversão do bit relativo ao item j estará proibida. Inicialmente, toma-se $T(j) = 0\ \forall j = 1, \cdots, n$. Considerando *iter* a iteração atual e supondo a proibição durante *TempoTabu* iterações, então $T(j) \leftarrow iter + TempoTabu$.

Exemplificando, ao final da primeira iteração do PM01 considerado na Seção 2, tem-se $j = 1$, $iter$=1 e $TempoTabu = 2$, então $T = (3, 0, 0, 0, 0, 0, 0, 0)$, indicando que qualquer modificação relativa ao item 1 (primeira posição do vetor T) estará proibida até a 3^a iteração da Busca Tabu. Dito de outra maneira, a inversão do bit da primeira posição é um movimento tabu enquanto a iteração

corrente $iter \leq T(1) = 3$. Na segunda iteração ($iter$=2), a inversão do bit da quarta posição foi considerada tabu. Assim, aplicando-se também a mesma duração tabu, tem-se a "lista" $T = (3, 0, 0, 4, 0, 0, 0, 0)$, indicando que a alteração dos valores dos bits relativos aos itens 1 e 4 está proibida até a terceira e quarta iterações, respectivamente. Observe que não é necessário atualizar os demais componentes da lista.

Osman (1993) trata um Problema de Roteamento de Veículos com um algoritmo BT que usa uma estrutura de dados na forma de matriz para armazenar os movimentos tabus. Essa matriz, de dimensões $(n+1) \times v$, consiste de v colunas (uma para cada conjunto R_p de clientes da rota p) e $n+1$ linhas, sendo uma para cada um dos n consumidores e outra para um consumidor fictício envolvido nas operações de inserção. Um movimento consiste de dois pares (i, R_p) e (j, R_q), os quais identificam que o consumidor i do conjunto R_p de clientes da rota p foi trocado com o cliente j do conjunto R_q de consumidores da rota q. Os atributos $\langle i, R_p \rangle$ e $\langle j, R_q \rangle$ especificam as restrições tabu que proíbem um movimento de ser realizado. Um movimento é considerado tabu se o consumidor i retorna a R_p e j retorna a R_q simultaneamente. Cada elemento $T(i, p)$ da matriz registra o número da iteração na qual o consumidor i foi removido do conjunto R_p. Inicialmente, a matriz T é inicializada com valores negativos altos para evitar a identificação equivocada de consumidores como tabu durante as iterações iniciais. Um movimento é considerado tabu em uma iteração k se nem i retorna a R_p, nem j retorna a R_q durante as próximas *TempoTabu* iterações, isto é, se forem satisfeitas simultaneamente as duas condições a seguir: $k - T(i, p) < TempoTabu$ e $k - T(j, q) < TempoTabu$.

Duas estratégias para escolha da próxima solução vizinha são adotadas em Osman (1993), cada qual dando origem a uma variante diferente do algoritmo Busca Tabu. A primeira, denominada *Best-admissible selection strategy* – BA, consiste em escolher o vizinho de melhor avaliação dentre todos da vizinhança da solução corrente. A segunda, chamada de *First-best-admissible strategy* – FBA, consiste em escolher o primeiro vizinho que melhorar a solução corrente; no caso de não haver soluções de melhora, então o melhor vizinho de não melhora é selecionado. Para aplicar a estratégia BA são usadas duas matrizes, BSTM e RECM, de dimensões $v \times v$ e $[v(v-1)/2] \times 2$, respectivamente. A parte triangular superior da matriz BSTM(p, q), com $1 \leq p < q \leq v$, é usada para armazenar a variação Δ_{ij} no valor da função objetivo associada ao melhor movimento obtido, de troca do consumidor $i \in R_p$ com $j \in R_q$, ou um valor arbitrariamente alto se tal movimento não é permitido. A parte triangular inferior BSTM(q, p) é usada para armazenar o índice da posição l associada com o par R_p e R_q no conjunto de possíveis pares de combinações $\{1, 2, \cdots, v(v-1)/2\}$. Tal índice indica a posição em que os atributos do melhor movimento estão armazenados. Por exemplo, RECM$(l, 1) = i$, RECM$(l, 2) = j$.

Em Glover and Laguna (1997) considera-se o Problema da k-Árvore Mínima. Esse problema consiste em procurar uma árvore[1] de k arestas em um grafo $G = (V, E)$, com $E = \{(i, j), i, j \in V\}$, tal que a soma dos pesos dessas arestas seja mínima. Um movimento consiste em eliminar uma aresta (i, j) da árvore e substituí-la por outra do grafo, que não pertence à árvore, sujeita à condição de que o grafo resultante também seja uma árvore. Há dois tipos de trocas possíveis. O primeiro consiste em manter os mesmos vértices da árvore corrente, mudando-se apenas uma aresta. No segundo tipo, um novo vértice entra em lugar de outro da árvore. Para qualquer das situações (adição ou eliminação de arestas da árvore corrente), os autores sugerem a utilização de uma única matriz tabu T, de dimensões $n \times n$, em que $n = |V|$, para armazenar a iteração até a qual os movimentos de adição e eliminação das arestas permanecem tabu. Importante ressaltar que não há confusão nos dois estados possíveis de uma mesma aresta, já que em uma dada iteração uma aresta pertence ou não à árvore. Uma aresta (i, j) fica tabu-ativo, isto é, proibida de ser manipulada (adicionada, caso ela não pertença à árvore corrente; ou eliminada, caso contrário) na iteração *iter*, se $iter \leq T(i, j)$, sendo $T(i, j) = k + TempoTabu$, k a iteração na qual a aresta (i, j) tornou-se tabu-ativo e *TempoTabu* a duração da regra de proibição. Os autores chamam a atenção para o fato de que é possível atribuir uma duração tabu para a adição de uma aresta e outra, diferente, para a eliminação de uma aresta.

7. Tamanho da Lista Tabu

Em uma Lista Tabu, seja ela de soluções ou baseada em atributos de movimentos/soluções, o tamanho da lista indica a quantidade máxima de iterações em que cada solução ou atributo deve lá permanecer para cumprir seu papel de impedir o retorno a uma solução já visitada anteriormente. Por esse motivo, o tamanho da lista tabu é comumente referenciado como a duração das regras de proibição ou duração tabu. Discutiremos, nesta Seção, sobre a duração tabu ideal.

Uma duração pequena permite a exploração de soluções "próximas" à solução corrente, uma vez que poucos movimentos são proibidos; enquanto durações longas fazem com que a busca se "afaste" dessa. O controle da duração tabu permite, portanto, fazer um balanço entre examinar uma região mais profundamente (o que induz a uma estratégia de intensificação) e mover-se para outras regiões do espaço de busca (o que induz a uma estratégia de diversificação). A duração tabu deve ser longa o suficiente para prevenir a ocorrência de ciclagem e pequena o suficiente para não proibir muitos movimentos e parar o procedimento de busca muito prematuramente.

A duração de uma regra de proibição pode ser fixa durante toda a busca ou ser dinâmica, isto é, pode variar ao longo das iterações.

Nos anos 1980, muitos trabalhos científicos adotavam como duração tabu típica, o número "mágico" 7, sendo esse um valor ainda muito usado, como

[1] Uma árvore é um grafo conexo (isto é, existe um caminho entre dois quaisquer de seus vértices) e sem ciclos

nos trabalhos de Ronconi and Armentano (2009) e Wan and Yen (2002) sobre sequenciamento. Outros autores preferem dimensionar empiricamente esse valor. Por exemplo, Gendreau et al. (1993) trataram o Problema da Clique Máxima[2] por três estratégias de Busca Tabu, combinando lista tabu de soluções com lista tabu de atributos de movimentos. Para explorar o espaço de soluções, os autores usaram dois tipos de movimentos, consistindo um deles em inserir um vértice v na clique Q corrente e outro, em eliminar um vértice v de Q. Na primeira estratégia, consideraram uma única lista tabu contendo as últimas 100 soluções geradas; na segunda, usaram duas listas, sendo uma com as últimas 100 soluções geradas e outra, baseada em atributos, com os 5 últimos vértices eliminados. Na terceira estratégia, de forma semelhante à segunda, também foram usadas duas listas, mudando-se apenas o tamanho da lista de soluções para 150.

Boa parte dos pesquisadores considera a duração como dependente das características do problema e do tamanho da instância considerada. Em Buscher and Shen (2009), por exemplo, os autores definem uma duração tabu dada por $|T| = 0,8 \times (n \times m \times s)^{0,5}$, em que n é o número de tarefas, m é o número de máquinas e s, o número de sublotes. Já em Osman (1993), são adotadas duas durações, uma para cada um dos dois algoritmos baseados em Busca Tabu desenvolvidos. Em uma delas, considera-se $|T| = 8 + (0,078 - 0,067 \times \rho) \times n \times v$, sendo n o número de consumidores, v o número de veículos e ρ, um parâmetro que representa a razão entre a soma das demandas dos consumidores e a soma das capacidades dos veículos usados nas melhores soluções da literatura até então. Em outra, $|T| = \max\{7, -40 + 0,6 \times ln(n \times v)\}$. Escolhe-se como duração tabu um dos três valores: $t_{\min} = 0,9 \times |T|$, $|T|$ ou $t_{\max} = 1,1 \times |T|$. Após isso, o valor t escolhido é mantido fixo por $2 \times t$ iterações, ao final das quais outro valor é escolhido aleatoriamente. Para resolver a mesma classe de problemas, Gendreau et al. (1994) escolheram uma duração tabu que varia dinamicamente no intervalo $[t_{\min}, t_{\max}]$, sendo $t_{min} = 0,9 \times n$ e $t_{max} = 1,1 \times n$. A seguir, essa duração é mantida fixa por $2 \times t_{\max}$ iterações.

A variação no valor da duração tabu ao longo da busca tem a vantagem de corrigir o erro porventura existente no dimensionamento empírico desse tempo. De fato, se houver ciclagem utilizando-se uma duração $|T|$, então variando-a, haverá alteração na quantidade de movimentos tabus e, assim, diferentes soluções poderão ser geradas. Com essa possibilidade de mudança na trajetória da busca no espaço de soluções, a ocorrência de ciclagem fica reduzida.

A lista tabu dinâmica, por sua vez, pode ser classificada em dois tipos: aleatória ou sistemática. Na aleatória, a duração tabu é selecionada aleatoriamente no intervalo $[t_{\min}, t_{\max}]$ e mantida fixa por $\alpha \times t_{\max}$ iterações, ao final das quais nova duração é sorteada. Outra possibilidade é selecionar uma duração tabu diferente a cada iteração. Na sistemática, a duração segue uma determinada regra previamente estabelecida. Por exemplo, em Ronconi and Armentano (2009), os autores escolhem a duração tabu aleatoriamente em um intervalo

[2] Dado um grafo não orientado $G = (V, E)$, uma clique (ou subgrafo completo) é um grafo cujos vértices são todos adjacentes entre si. Em outras palavras, Q é uma clique de G se $Q \subseteq V(G)$ e para todo $u, v \in Q$ então $(u, v) \in E(G)$. No Problema da Clique Máxima o objetivo é determinar o tamanho $\omega(G)$ da maior clique de G.

$[t_{\min}, t_{\max}]$ e aplicam essa duração como referência para as próximas 20 iterações, ao final das quais nova duração é sorteada. A seguir, essa duração base é ajustada na iteração corrente de acordo com o valor da solução vizinha. Se o valor da solução vizinha é maior que o da solução corrente (aquela a partir da qual foi aplicado o movimento), então a duração base é aumentada em uma unidade. Caso, no entanto, esse valor seja menor, então a duração é diminuída de uma unidade. Outra regra, apontada em França (2009), é definir uma sequência de números inteiros no intervalo $[t_{\min}, t_{\max}]$ em que a duração aumente e diminua de forma alternada. Exemplo: Dado o intervalo [5, 10], ter-se-ia uma sequência $S = \{5, 8, 6, 9, 7, 10\}$, indicando que na primeira iteração da aplicação dessa regra, a duração seria 5, depois aumentaria para 8, a seguir diminuiria para 6 e assim por diante.

Em Ronconi and Armentano (2009), os autores chamam a atenção para o fato de que a duração tabu depende também da regra de proibição e do atributo do movimento considerado. Em um problema de sequenciamento de n tarefas, uma regra que proíbe uma tarefa i de ser movida em um movimento de troca reduz para $n-1$ o número de tarefas que podem ser trocadas após sua primeira aplicação. Considerando que se proíbe uma tarefa a cada iteração, e que para a realização do movimento de troca há necessidade de pelo menos duas tarefas liberadas, a última troca possível se realizará na iteração $n-1$. Dessa forma, $n-1$ é o limitante superior para a duração dessa regra de proibição. Uma alternativa para corrigir o erro de um limitante mal dimensionado é aplicar o critério de aspiração *default*, isto é, desativar a regra de proibição para o(s) atributo(s) tabu-ativo(s) mais antigo(s).

Os limitantes inferior e superior da duração tabu das regras de proibição relativas aos atributos dos movimentos considerados em Ronconi and Armentano (2009) são explicitados na Tabela 1.14.

Tabela 1.14: Limitantes das durações tabus de Ronconi and Armentano (2009)

Movimento	Atributo	Regra	Limitante inferior	Limitante superior
Inserção	i	i não pode ser movida	$n/4$	$n/2$
	i	i não pode ser escolhida para inserção	$n/2$	**n**
	$i, s(i)$	i não pode retornar à posição $s(i)$	$4n$	$5n$
Troca	i, j	i e j não podem ser movidas	$n/4$	$n/2$
	i	i não pode ser movida	$n/2$	**n-1**
	$i, s(i)$	i não pode ser movida para posições $k \leq s(i)$	$n/2$	$3n/2$

Os autores observam, pela Tabela 1.14, que os limitantes superiores destacados em negrito foram determinados após um estudo sobre o número de iterações necessárias para que não mais houvesse movimentos disponíveis (não tabu) para

serem escolhidos, considerando que, quando uma regra de proibição é ativada, ela permanece válida ao longo das iterações. Considera-se, além disso, que o critério de aspiração não está ativado. Os demais valores foram estimados por testes computacionais. Por esta tabela, observa-se que quanto mais restritiva é a regra de proibição, menor é o limitante superior.

Finalmente, ressalta-se que o tamanho da lista também pode depender do estágio da busca. Em problemas de programação de horários (*timetabling*), por exemplo, conforme relatado em Souza (2000), é recomendável aumentar o tamanho da lista quando a busca se encontra numa região com soluções de igual valor da função objetivo. Com essa estratégia, mais soluções dessa região do espaço ficam proibidas, dando oportunidade ao algoritmo de caminhar em outras direções. Ao sair dessa região, a lista volta ao seu tamanho normal. Em Buscher and Shen (2009), os autores aplicam uma estratégia baseada na proposta de Battiti and Tecchiolli (1994) para alterar o tamanho da lista quando a busca detecta a existência de soluções frequentemente visitadas. Basicamente, a duração tabu é progressivamente aumentada em *INC* unidades sempre que uma solução é visitada por mais do que *REP* vezes. Na ausência de repetições, a duração tabu é diminuída em *DEC* unidades. Detalhes dessa estratégia, que compõe a chamada *Busca Tabu Reativa*, são descritos em Battiti and Tecchiolli (1994) e Battiti (1996).

8. Critérios de aspiração

São apresentados, a seguir, alguns dos critérios de aspiração mais comuns.

Aspiração por objetivo global: Consiste em retirar o *status* tabu de um movimento se for produzida uma solução com a melhor avaliação global.

Aspiração por objetivo regional: Um movimento tabu perde seu *status* quando for gerada uma solução melhor que a melhor encontrada na região atual de busca. Uma forma de se delimitar a região atual de busca é registrar a melhor solução encontrada em um passado recente e utilizar o valor dessa solução como critério para aspiração. Por exemplo, considerando um problema de minimização e supondo $f(s^\star_R)$ a melhor solução encontrada nas últimas *ContIterRegiao* iterações, então um movimento tabu que guia a uma solução s' tal que $f(s') < f(s^\star_R)$ pode ser realizado. Nessa estratégia de aspiracão, *ContIterRegiao* é um parâmetro.

Aspiração *Default*: Se todos os movimentos possíveis são tabus e não é possível aplicar outro critério de aspiração, então o movimento mais antigo perde sua condição tabu. Em uma implementação baseada em tempo de permanência na lista, como o exemplificado para o Problema da Mochila, se $T(j) \geq iter\ \forall j$, em que *iter* representa a iteração atual, então a inversão do bit da posição k, tal que $T(k) = \min\{T(j),\ \forall j\}$, pode ser realizada, apesar de tabu.

Há outros critérios de aspiração mais sofisticados, como aqueles desenvolvidos em de Werra and Hertz (1989) e Hertz and de Werra (1990). No entanto, apesar de bem sucedidos em suas aplicações, eles raramente são usados. O critério mais usado, presente na quase totalidade das implementações baseadas em Busca Tabu, é o critério de aspiração por objetivo global.

Uma implementação BT precisa prever, sempre que possível, a aplicação do critério de aspiração *default*. Por exemplo, no PM01 considerado na Seção 2, trabalhamos apenas com soluções factíveis. Ainda que sejam levados em consideração os limitantes do tamanho da lista (como os mostrados na Tabela 1.14), haveria a possibilidade de todos os vizinhos factíveis de uma dada solução serem tabus. Nesse caso, a aplicação do critério de aspiração *default* é uma alternativa para a Busca Tabu prosseguir na exploração do espaço de soluções.

9. Memória de Longo Prazo

Uma memória de curto prazo não é suficiente para evitar que a busca fique presa em certas regiões do espaço de soluções. Assim, alguma estratégia de diversificação é necessária. Para cumprir esse objetivo, algoritmos Busca Tabu usam uma memória de longo prazo para encorajar o processo de busca a explorar regiões ainda não visitadas. A vantagem em se utilizar tal estratégia ao invés de promover uma simples reinicialização do algoritmo é que, com o uso de memória de longo prazo, diminui-se o risco de voltar a visitar uma mesma região do espaço de soluções, situação que poderia ocorrer no processo de reinicialização em vista da perda das informações anteriores.

A memória de longo prazo também pode visar à aplicação de uma estratégia de intensificação. Ao contrário da diversificação, o objetivo agora é estimular a busca a gerar soluções que contenham atributos encontrados nas soluções já visitadas que sejam historicamente bons.

A memória de longo prazo pode ser baseada em frequência de transição ou em frequência de residência. Esses tipos de memória são apresentados a seguir.

Memória baseada em frequência de transição

A ideia aqui é usar uma estrutura de memória para computar atributos que mudam de uma solução visitada para outra.

Em Santos et al. (2005) é desenvolvido um algoritmo BT para resolver o Problema de Programação de Horários em Escolas (PPHE). No problema considerado, há um conjunto $T = \{1, \cdots, t\}$ de professores, um conjunto $C = \{1, \cdots, c\}$ de turmas e um conjunto $P = \{1, \cdots, p\}$ de horários para a realização das aulas. O objetivo é alocar as aulas dos professores às turmas minimizando o número de horários ociosos na programação semanal do professor e o número de vezes em que cada professor vai à escola, bem como maximizar o atendimento ao número de aulas geminadas requeridas pelos professores para cada turma.

Um quadro de horários é representado por uma matriz $Q_{t\times p}$, em que cada linha representa a programação de aulas de um professor. Uma célula $q_{ik} \in \{0, 1, \cdots, c\}$ indica a alocação do professor i no horário k, sendo que $q_{ik} = j$,

com $j \in \{1, \cdots, c\}$, representa a turma para a qual o professor i está lecionando e $q_{ik} = 0$ significa que o professor i está disponível naquele horário.

Na Figura 1.3 mostra-se um fragmento de um quadro de horários envolvendo 5 professores e p horários. A letra "X" indica que o professor está indisponível no horário.

	Horários							
Professor	1	2	3	4	5	6	$\cdots$	p
1	1	0	0	2	2	X	$\cdots$	$\cdots$
2	0	X	X	0	1	2	$\cdots$	$\cdots$
3	X	X	1	0	3	1	$\cdots$	$\cdots$
4	0	1	0	1	0	3	$\cdots$	$\cdots$
5	0	0	2	3	X	X	$\cdots$	$\cdots$

Figura 1.3: Fragmento de um quadro de horários

Pela Figura 1.3, o professor 5 está disponível nos dois primeiros horários, dá aula para a turma 2 no terceiro horário e para a turma 3 no quarto horário e está indisponível no quinto e sexto horários.

Os autores exploram o espaço de soluções por meio de movimentos baseados na troca de duas alocações feitas na programação de horários de um professor. Um movimento envolvendo os horários p_1 e p_2 de um professor i é representado por um tripla $\langle i, p_1, p_2 \rangle$, em que $q_{i,p_1} \neq q_{i,p_2}$, $p_1 < p_2$ e $p_1, p_2 \in \{1, \cdots, p\}$. Como o movimento pode dar origem a um quadro de horários infactível, a função de avaliação adotada é a soma da função objetivo propriamente dita com uma parcela que mensura o nível de inviabilidade de uma solução.

Uma vez realizado o movimento, ele é mantido tabu pelas próximas *TempoTabu* iterações. Essa duração tabu não é fixa, mas sim variável, sendo escolhida aleatoriamente dentro de um intervalo $[t_{\min}, t_{\max}]$, em que os limitantes são definidos pelas equações (1.6) e (1.7):

$$t_{\min} = \lfloor \mathit{TempoTabuCentral} - \varphi \times \mathit{TempoTabuCentral} \rfloor \tag{1.6}$$

$$t_{\max} = \lceil \mathit{TempoTabuCentral} + \varphi \times \mathit{TempoTabuCentral} \rceil \tag{1.7}$$

No cálculo dessas durações mínima e máxima, *TempoTabuCentral* é um parâmetro de entrada que estima a duração tabu de um movimento e $\varphi \in [0, 1]$, outro parâmetro de entrada que define a variação permitida nessa duração.

Os autores armazenam em uma matriz $Z_{t \times c}$ o número z_{ij} de movimentos feitos envolvendo um professor i e uma turma j. Usando esses valores e considerando $\bar{z} = \max\{z_{ij}, i \in T, j \in C\}$, é definida uma medida de frequência relativa de transição ϵ_{ij}, computada conforme a Eq. (1.8):

$$\epsilon_{ij} = z_{ij}/\bar{z} \tag{1.8}$$

Como um movimento $\langle i, p_1, p_2 \rangle$ envolvendo os horários p_1 e p_2 da agenda do professor i pode ser relativo a dois horários de aula ou a um horário de aula e

um horário livre, então a penalidade ψ_{i,j_1,j_2} associada ao movimento é calculada com base na Eq. (1.9), a qual leva em consideração as três situações possíveis:

$$\psi_{i,j_1,j_2} = \begin{cases} \epsilon_{i,j_1} \times f(Q^\star) & \text{se } j_1 \neq 0 \text{ e } j_2 = 0 \\ \epsilon_{i,j_2} \times f(Q^\star) & \text{se } j_1 = 0 \text{ e } j_2 \neq 0 \\ (\epsilon_{i,j_1} + \epsilon_{i,j_2})/2 \times f(Q^\star) & \text{se } j_1 \neq 0 \text{ e } j_2 \neq 0 \end{cases} \tag{1.9}$$

Na Eq. (1.9), $f(Q^\star)$ é o custo da melhor solução encontrada até então, j_1 é a alocação do professor i no horário p_1 (isto é, $j_1 = q_{i,p_1}$) e j_2 é a alocação do professor i no horário p_2 (ou seja, $j_2 = q_{i,p_2}$).

O Algoritmo 2 mostra o pseudocódigo implementado. Nesse algoritmo, a atualização da lista tabu é feita atribuindo-se ao movimento realizado uma duração aleatória calculada no intervalo definido pelas equações (1.6) e (1.7). A penalização de um movimento (linha 10) é calculada com base na Eq. (1.9). Toda vez que é realizado um movimento, a lista tabu de longo prazo é atualizada (linha 21) e sempre que a melhor solução é modificada, ela é reinicializada (linha 26). A estratégia de diversificação é acionada após um número *IterAcionaDiv* de iterações sem melhora e fica ativa por *IterDivAtiva* iterações (linha 9). *IterDivAtiva* e *IterAcionaDiv* são parâmetros do algoritmo.

Memória baseada em frequência de residência

Em uma memória de longo prazo baseada em frequência de residência a ideia é computar os atributos que são frequentes nas soluções visitadas para usá-los como estratégia de intensificação (estimulando-os a comporem a solução corrente) e/ou como uma estratégia para diversificar a busca (neste caso, penalizando-os para desestimular seu uso).

Em Santos et al. (2005), os autores também desenvolveram um algoritmo BT explorando memória de longo prazo baseada em frequência de residência. A medida de residência é armazenada em uma matriz $Y_{t\times c\times u\times p}$, em que $u = \max\{r_{ij}, i \in T, j \in C\}$, t é o número de professores, c é o número de turmas, p é o número de horários de aula e r_{ij} o número de aulas do professor i para a turma j. Uma medida de frequência relativa de residência da m-ésima aula do professor i para a turma j no horário k é calculada com base na Eq. (1.10):

$$\eta_{ijmk} = \frac{y_{ijmk}}{\bar{y}} \tag{1.10}$$

sendo $\bar{y} = \max\{y_{ijmk}, i \in T, j \in C, m \in \{1, 2, \cdots, u\}, k \in P\}$.

A penalidade μ_{ijmk} por alocar a m-ésima aula do professor i para a turma j no horário k é calculada conforme a Eq. (1.11):

$$\mu_{ijmk} = y_{ijmk} \times f(Q) \tag{1.11}$$

O algoritmo BT desenvolvido é o mesmo da Seção anterior, diferenciando-se daquele apenas na forma de calcular a penalização do movimento (linha 10 do Algoritmo 2), a qual é feita considerando-se a Eq. (1.11).

Em Díaz and Fernández (2001), os autores trataram o Problema Generalizado de Atribuição (PGA) por meio de um algoritmo BT que faz uso de memórias

Algorithm 2 BT-PPHE

Entrada: Q, *IterAcionaDiv*, *IterDivAtiva*, *TempoTabuCentral*, φ
Saída: $Q^\star$
$Q^* \leftarrow Q$; *InicializeListaTabu*; *IterSemMelhora* $\leftarrow 0$; *Iter* $\leftarrow 0$;
InicializeMemoriaLongoPrazo();
repeat
  $\Delta \leftarrow \infty$; *Iter*++; *MelhorMovimento* $\leftarrow$ *MovimentoAleatorio*();
  for all movimento m tal que $(Q \oplus m) \in \mathcal{N}(Q)$ **do**
    Penalidade $\leftarrow 0$;
    if *IterSemMelhora* **mod** *IterAcionaDiv* $<$ *IterDivAtiva* **e** *iter* $\geq$ *IterAcionaDiv* **then**
      Penalidade $\leftarrow$ *CalculePenalidade*(m);
    end if
    $\Delta' \leftarrow f(Q) - f(Q \oplus m)$;
    if $(\Delta' +$ *Penalidade* $< \Delta$ **e** m não tabu) **ou** $(f(Q \oplus m) < f(Q^\star)$ **e** $\Delta' < \Delta)$ **then**
      MelhorMovimento $\leftarrow m$;
      $\Delta \leftarrow \Delta'$;
      if $f(Q \oplus m) \geq f(Q^\star)$ **then**
        $\Delta \leftarrow \Delta +$ *Penalidade*;
      end if
    end if
  end for
  AtualizeMemoriaLongoPrazo(*MelhorMovimento*, Q);
  $Q \leftarrow Q \oplus$ *MelhorMovimento*;
  AtualizeListaTabu(*MelhorMovimento*, *Iter*);
  if $f(Q) < f(Q^\star)$ **then**
    $Q^\star \leftarrow Q$; $IterSemMelhora \leftarrow 0$;
    InicializeMemoriaLongoPrazo();
  else
    IterSemMelhora ++;
  end if
until Critério de parada atingido;

de curto e longo prazos. No PGA há um conjunto $J = \{1, \cdots, n\}$ de tarefas que devem ser processadas por um conjunto $I = \{1, \cdots, m\}$ de agentes. Cada tarefa j consome a_{ij} unidades de recurso do agente i, o qual, por sua vez, tem capacidade de processar apenas b_i unidades do recurso. A designação de uma tarefa j a um agente i custa c_{ij} unidades e o objetivo é fazer a alocação de menor custo, de sorte que cada tarefa seja atribuída a um único agente e que a capacidade de cada agente seja respeitada.

O PGA pode ser formulado como o modelo de programação linear inteira, dado pelas equações (1.12) - (1.15). Nesse modelo, a variável x_{ij} assume valor 1 se a tarefa j é designada ao agente i e 0, caso contrário. A função objetivo, dada pela Eq. (1.12), computa o custo das designações. As restrições (1.13) asseguram

que cada tarefa é executada por um único agente. As restrições (1.14) garantem que a disponibilidade de recurso de cada agente seja respeitada.

$$\min \quad z = \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{1.12}$$

$$\text{s. a:} \quad \sum_{i \in I} x_{ij} = 1 \qquad \forall\, j \in J \tag{1.13}$$

$$\sum_{j \in J} a_{ij} x_{ij} \leq b_i \qquad \forall\, i \in I \tag{1.14}$$

$$x_{ij} \in \{0,1\} \quad \forall\, i \in I, j \in J \tag{1.15}$$

No procedimento proposto, o custo de designação de uma tarefa j a um agente i é calculado de forma relativa pela Eq. (1.16), em que se toma como referência o menor custo de atribuição da tarefa j.

$$\Delta_{ij} = c_{ij} - \min\{c_{ij}, i \in I\} \tag{1.16}$$

A função objetivo pode, então, ser reescrita como:

$$\min \quad z = \sum_{j \in J} \min\{c_{ij}, i \in I\} + \sum_{i \in I} \sum_{j \in J} \Delta_{ij} x_{ij} \tag{1.17}$$

Uma vez que a primeira parcela da Eq. (1.17) é constante, ela pode ser removida, resultando na função objetivo (1.18).

$$\min \quad z' = \sum_{i \in I} \sum_{j \in J} \Delta_{ij} x_{ij} \tag{1.18}$$

Para explorar o espaço de soluções, os autores relaxam as restrições de capacidade dos agentes (Eq. (1.14)), penalizando-as na função objetivo, a qual passa a ser expressa pela Eq. (1.19).

$$\min \quad z'' = \sum_{i \in I} \sum_{j \in J} \Delta_{ij} x_{ij} + \rho \times \sum_{i \in I} e_i \tag{1.19}$$

Na Eq. (1.19), a parcela $e_i = \max\{\sum_{j \in J} a_{ij} x_{ij} - b_i, 0\}$ mensura a quantidade de violação de capacidade do agente i e ρ é um fator de penalidade.

Essa relaxação é utilizada porque, segundo os autores, as restrições (1.14) são difíceis de serem atendidas, pois confinam a busca a uma região estreita do espaço de soluções factíveis. Desta forma, utilizando-se apenas movimentos simples há poucas soluções vizinhas factíveis, forçando a busca a terminar rapidamente, em geral com soluções de baixa qualidade. Por outro lado, permitindo-se alguma violação de viabilidade, há duas vantagens. Primeiro, permite-se a execução de movimentos menos complexos que aqueles que seriam necessários. Segundo, dá-se à busca um grau maior de flexibilidade e uma região mais ampla do espaço

de soluções pode ser então explorada. O desvio de viabilidade é controlado pelo fator de penalidade ρ na função objetivo. Essa mesma estratégia de penalização é adotada por Yagiura et al. (2006) para o PGA.

Uma solução do PGA é representada por um vetor s de n elementos, em que o k-ésimo componente é o agente designado para a execução da tarefa j, isto é:

$$s = (s_1, \cdots, s_n), \text{ em que } s_j \in I; s_j = i \Longleftrightarrow x_{ij} = 1$$

Exemplificando, seja $I = \{A, B, C\}$ e $J = \{1, 2, 3, 4, 5\}$. Uma solução s para o PGA desse exemplo pode ser $s = (C, A, C, B, A)$, o que significa que as tarefas 1 e 3 são executadas pelo agente C, as tarefas 2 e 5 pelo agente A e a tarefa 4, pelo agente B.

São usados dois tipos de movimento, sendo um baseado em troca e outro em substituição, para definir as vizinhanças $N^T(.)$ e $N^S(.)$, respectivamente. No primeiro tipo, dadas duas tarefas distintas j_1 e j_2, são permutados os agentes s_{j_1} e s_{j_2} que as executam, desde que eles não sejam os mesmos, isto é, $s_{j_1} \neq s_{j_2}$. No movimento de substituição, a atribuição de uma tarefa j é alterada de um agente i_1 para outro i_2, sendo $i_1 \neq i_2$. No exemplo considerado, $s'_1 = (A, C, C, B, A)$ é um vizinho obtido pelo movimento de troca, enquanto $s'_2 = (B, A, C, B, A)$, pelo de substituição. Como há $m - 1$ possibilidades para substituir o agente que executa uma dada tarefa e há n tarefas, então na vizinhança de substituição há $(m - 1)n$ vizinhos. Na vizinhança de troca, por sua vez, há um máximo de $n(n - 1)/2$ vizinhos.

Os autores observam que os movimentos de troca não alteram a quantidade de tarefas designadas para cada agente, isto é, se um agente executa, digamos, duas tarefas, continuará a executar duas tarefas e o que muda são as tarefas atribuídas. Por outro lado, movimentos de substituição são mais restritivos em termos de factibilidade. Assim, adotaram como vizinhança de uma solução s a união das vizinhanças de troca e substituição, ou seja, $N(s) = N^T(s) \cup N^S(s)$, por ser uma opção mais abrangente.

Ao se realizar um movimento, seja ele de troca ou substituição, considera-se como atributo o par $\langle i, j \rangle$, em que i é o agente designado para executar a tarefa j. Como memória de curto prazo, considera-se uma matriz tabu T que armazena na posição (i, j) a iteração até a qual tal atributo estará proibido. Exemplo: se durante uma dada iteração k, uma tarefa j do agente i_1 for realocada a outro agente i_2, então o par $\langle i_1, j \rangle$ permanecerá tabu nas próximas *TempoTabu* iterações. Isso é implementado fazendo-se $T_{i_1,j} = k +$ *TempoTabu*. No caso de movimentos de troca, em que se envolvem dois pares $\langle i_1, j \rangle$ e $\langle i_2, j \rangle$, com $i_1 \neq i_2$, somente um deles é registrado como atributo tabu-ativo, no caso, aquele associado ao maior valor de custo ($\Delta_{i_1,j}$ ou $\Delta_{i_2,j}$).

Os autores utilizam uma memória de longo prazo baseada em frequência de residência tanto para aplicar uma estratégia de intensificação quanto de diversificação. Durante a busca, armazena-se em uma matriz *fr* o número de vezes em que uma tarefa j ficou alocada ao agente i, tendo em vista o total de soluções visitadas até aquele ponto.

Para a aplicação da fase de intensificação, toma-se a melhor solução encon-

trada até então e verificam-se as designações de agentes a tarefas dessa solução. Se um agente ficou designado com muita frequência a uma tarefa, no caso 85%, no mínimo, essa atribuição é considerada fixa. Segundo os autores, as atribuições de alta frequência podem ser consideradas boas dado que: primeiro, elas estão presentes na melhor solução encontrada até então. Segundo, se elas aparecem na maioria das soluções geradas até aquela iteração, é porque elas foram selecionadas muitas vezes ou, após selecionadas, elas não foram trocadas por um longo período de tempo. Após a fixação de algumas tarefas a agentes, dá-se início à fase de intensificação, acionando-se a chamada a um procedimento que usa memória de curto prazo. Durante esse período, a matriz fr continua a ser atualizada. A vantagem dessa estratégia é que, com a fixacão de algumas designações, a fase de intensificação atua em um problema de menor dimensão, possibilitando, assim, realizar uma busca mais efetiva nesse espaço.

Para a aplicação da fase de diversificação, à matriz de custo Δ é adicionada uma parcela de penalidade fr que objetiva desestimular atribuições de tarefas a agentes realizadas com muita frequência. Nessa fase, todas as designações voltam a se tornar livres e a matriz de custo é alterada, inicialmente, para $\Delta + fr$. A seguir, aplica-se o procedimento de busca baseado em memória de curto prazo por um número reduzido de iterações. Finalizado esse procedimento, aciona-se novamente o procedimento de busca baseado em memória de curto prazo, mas desta vez usando-se a matriz de custo Δ original.

O Algoritmo 3 mostra o pseudocódigo do algoritmo de Busca Tabu desenvolvido por Díaz and Fernández (2001) para resolver o PGA. Nesse algoritmo, s_0 é a solução inicial, l é o número de iterações em que são aplicadas as fases de intensificação e diversificação e k indica o número de soluções geradas.

O pseudocódigo do procedimento *MemoriaCurtoPrazo*(k, s^k, $s^\star$, T, fr, Δ) é apresentado no Algoritmo 4. Nesse algoritmo, o custo de uma solução assume valor $\Delta + fr$ quando ele é acionado para executar a estratégia de diversificação (linha 17 do Algoritmo 3) e valor Δ nos demais casos.

Algorithm 3 `BT-PGA`

Entrada: s_0, l
Saída: $s^\star$
$T \leftarrow 0$; $fr; \leftarrow 0$; $s^* \leftarrow s_0$; $k \leftarrow 0$;
Gere solução inicial s_0;
Aplique *MemoriaCurtoPrazo*(k, s^k, $s^\star$, T, fr, Δ);
for all $iter = 1$ até l **do**
  { Fase de intensificação }
  $s^* \leftarrow s^k$;
  for all tarefa j de s^* **do**
    if $fr(s^\star_j, j) > 0,85 \times k$ **then**
      Fixe a tarefa j ao agente i, isto é, faça $x_{s^\star_j, j} \leftarrow 1$;
    end if
  end for
  Aplique *MemoriaCurtoPrazo*(k,s^k);
  { Fase de diversificação }
  Libere todas as atribuições;
  Aplique *MemoriaCurtoPrazo*(k, s^k, $s^\star$, T, fr, Δ) por poucas iterações usando $\Delta + fr$ como matriz de custo;
  Recupere a matriz de custo Δ original;
  Aplique *MemoriaCurtoPrazo*(k, s^k, $s^\star$, T, fr, Δ);
end for

Algorithm 4 `MemoriaCurtoPrazo`(k, s^k, $s^\star$, T, fr, Δ)

while Critério de parada não atingido **do**
  Selecione um vizinho $s^{k+1} \in N(s^k)$;
  if *custo*$(s^{k+1}) <$ *custo*$(s^\star)$ **then**
    $s^\star \leftarrow s^{k+1}$;
  end if
  Atualize a memória de curto prazo T;
  Atualize a memória de longo prazo fr;
  $k \leftarrow k + 1$;
end while

Em Laguna (1994) é apresentado um guia bastante didático para a implementação de um algoritmo baseado em Busca Tabu, usando como referência o Problema das n-Rainhas. Esse problema consiste em encontrar uma disposição de n rainhas do jogo de xadrez em um tabuleiro $n \times n$, de tal modo que nenhuma rainha ataque (colida com) as outras de acordo com as regras do jogo. Uma rainha ataca outra quando elas estão posicionadas na mesma linha ou mesma coluna ou mesma diagonal.

Uma solução do problema é representada por um vetor s de n posições, em que cada célula s_i representa a coluna j em que a rainha da linha i ocupa. Portanto, o par $(i, s_i) = (i, j)$ representa a posição de uma rainha.

A Figura 1.4 ilustra a disposição das rainhas em um problema de 6 rainhas,

com a configuração $s = (2, 3, 5, 1, 6, 4)$.

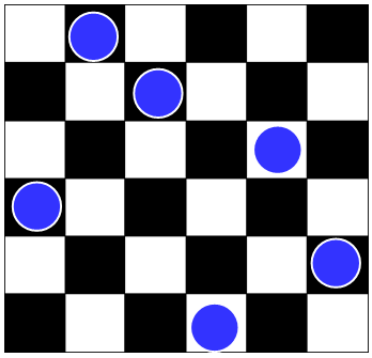

Figura 1.4: Representação de uma solução no Problema das n-Rainhas

Pela Figura 1.4, observa-se que a rainha da primeira linha está na segunda coluna, a da segunda linha está na terceira coluna e assim por diante. Com esta representação, não há colisão de rainhas em linhas ou colunas, apenas em diagonais. Assim, a função objetivo, isto é, o número de colisões, pode ser determinado calculando-se apenas as colisões nas diagonais.

Para calcular o número de colisões nas diagonais, inicialmente são criados dois vetores d_n e d_p, cada qual contendo $2n-1$ elementos, o primeiro para representar as diagonais negativas e o segundo, as diagonais positivas. Dada uma rainha na linha i e coluna s_i, o índice k de um elemento da diagonal negativa é obtido fazendo-se $k = i + s_i$, enquanto o índice k de um elemento da diagonal positiva é obtida fazendo-se $k = i - s_i$. As rainhas 2 e 4, por exemplo, estão na diagonal negativa de índice 5. De fato, $2 + s_2 = 2 + 3 = 4 + s_4 = 4 + 1 = 5$. De forma semelhante, as rainhas 1, 2 e 6 estão na diagonal positiva de índice -1 pois $1 - s_1 = 1 - 2 = 2 - s_2 = 2 - 3 = 5 - s_5 = 5 - 6 =$ -1. Desta forma, os índices das diagonais positivas variam no intervalo $[1 - n, n - 1]$, enquanto os índices das diagonais negativas pertencem ao intervalo $[2, 2n]$.

O número de rainhas em cada diagonal é, inicialmente, fixado em 0 para todas as diagonais positivas e negativas, conforme o código a seguir.

para $(i = 2, \cdots, 2n)$ faça
    $d_n(i) \leftarrow 0$;
    $d_p(i - n - 1) \leftarrow 0$;
fim-para

Em seguida, são contabilizados o número de rainhas em cada uma das diagonais (positiva e negativa):

para $(i = 1, \cdots, n)$ faça
    $d_n(i + s(i)) \leftarrow d_n(i + s(i)) + 1$;
    $d_p(i - s(i)) \leftarrow d_p(i - s(i)) + 1$;
fim-para

O número de colisões de uma solução s pode ser, então, determinado pela Eq. (1.20):

$$f(s) = \sum_{k=1-n}^{n-1} \max\{0, d_p(k) - 1\} + \sum_{k=2}^{2n} \max\{0, d_n(k) - 1\} \qquad (1.20)$$

Observa-se que o número de colisões em uma dada diagonal é o número de rainhas nessa diagonal menos uma unidade.

A exploração do espaço de soluções do problema é feita utilizando-se movimentos de troca de coluna entre duas rainhas posicionadas em linhas distintas i e j. Adota-se como atributo tabu o par $\langle i, j \rangle$. Exemplo: dada a configuração $s = (2, 4, 6, 5, 7, 3, 1)$ e o movimento de troca das rainhas das linhas 3 e 5, gera-se a solução $s' = (2, 4, 7, 5, 6, 3, 1)$, sendo tabu o movimento $\langle 3, 5 \rangle$. Armazena-se na parte triangular superior de uma matriz T, na posição T_{ij} (considerando $i < j$), a iteração até a qual tal movimento permanece tabu. As informações contidas nesta matriz triangular superior são usadas como memória de curto prazo para a Busca Tabu.

Toda vez que é realizada uma troca de colunas envolvendo as rainhas das linhas i e j, é computada na parte triangular inferior dessa matriz T, na posição T_{ji} (considerando $i < j$), o número de trocas realizadas até então. Essas informações representam a memória de longo prazo baseada em frequência de transição e são usadas pelo autor apenas quando não há movimentos de melhora, servindo para diferenciar tais movimentos.

Na Tabela 1.15 mostram-se as memórias de curto e longo prazos, considerando que a iteração atual é $iter = 26$, que a duração tabu de um movimento é 3 e que a solução corrente é $s = (1, 3, 6, 2, 7, 5, 4)$. Para clareza de apresentação, estão representadas na matriz apenas as células que representam movimentos tabu-ativos. Na Tabela 1.16 mostram-se as cinco melhores trocas, o valor do movimento e esse valor penalizado, isto é, a soma do valor atual do movimento com a frequência de trocas armazenada na parte inferior da matriz T.

Tabela 1.15: Estrutura de dados n-Rainhas

	1	2	3	4	5	6	7
1						29	
2							
3	5					28	
4		3					27
5	2			4			
6	3				2		
7		4	3	1			

Tabela 1.16: Melhores candidatos

Troca		Valor	V. penalizado
1	6	0	1
1	3	1	6
1	**5**	1	**3**
2	7	1	5
3	7	1	4

Exemplificando, a troca $\langle 2, 7 \rangle$, que tem valor de movimento igual a 1, é agora penalizada em mais quatro unidades, uma vez que na parte inferior da matriz T, tem-se $T_{72} = 4$. Ou seja, como as rainhas 2 e 7 trocaram de posição 4 vezes durante o histórico da busca, então esse valor é usado para penalizar tal movimento.

No exemplo apontado, a memória baseada em frequência tem influência decisiva na diferenciação dos movimentos de mesmo valor. De fato, o movimento com melhor avaliação, o qual envolve a troca de coluna das rainhas das linhas 1 e 6 e tem valor 0, é tabu, pois $T_{16} = 29 \geq 26 = iter$. Com a penalização adotada, o melhor movimento passa a ser a troca de coluna das rainhas das linhas 1 e 5 (troca destacada em negrito na Tabela 1.16).

Como o custo da análise da vizinhança inteira é $O(n^2)$, o autor também propõe a utilização de uma lista de candidatos. A ideia proposta é criar uma lista envolvendo apenas as rainhas que colidem, e reconstruir essa lista por uma busca completa depois de um certo tempo. Exemplo: supondo que rainhas 1, 5 e 7 são as únicas a participar de colisões no problema das 7-rainhas, nas próximas 4 iterações, por exemplo, a lista de candidatos consistiria de trocas envolvendo apenas tais rainhas. Após isso, seria feita busca na vizinhança inteira para reconstruir a lista das próximas rainhas a participar de colisões.

O autor propõe, ainda, a utilização de uma memória de longo prazo baseada em frequência de residência como estratégia de intensificação. A ideia, no caso, seria criar uma segunda matriz, nomeada *FreqBoasSol*, e contabilizar nela apenas os atributos das melhores soluções encontradas, no caso, daquelas que efetivamente estiverem a uma certa distância do valor da melhor solução encontrada até então. Mais precisamente, após um movimento ser selecionado, se o custo da solução gerada (*custoVizinho*) estiver pouco distante (abaixo até $p\%$ do valor da melhor solução até então, denotado por *MelhorCusto*), então os atributos dessa solução gerada seriam contabilizados. O atributo considerado é o par $\langle i, s_i \rangle$, em que i indica a linha em que está uma rainha e s_i a sua coluna. Representa-se, a seguir, o trecho de código para atualizar essa memória de longo prazo.

```
se (custoVizinho < (1 + p) × MelhorCusto) então
    para (i = 1, ··· , n) faça
        FreqBoasSol(i, s(i)) ← FreqBoasSol(i, s(i)) + 1;
    fim-para
fim-se
```

Para exemplificar, seja a solução $s = (2, 4, 6, 5, 7, 3, 1)$ e o movimento de troca envolvendo as rainhas das linhas 3 e 5. Supondo que a solução vizinha $s' = (2, 4, 7, 5, 6, 3, 1)$, resultante desse movimento, satisfaça à condição anterior, então seriam aumentados em uma unidade os valores contidos nas seguintes células da matriz *FreqBoasSol*: (1, 2), (2, 4), (3, 7), (4, 5), (5, 6), (6, 3) e (7, 1).

A fase de intensificação seria acionada em certas ocasiões, como por exemplo, depois de um dado tempo sem melhora. Nessa situação, a matriz *FreqBoasSol* poderia ser usada para proibir trocas de rainhas localizadas em colunas com alta frequência. Dito de outra maneira, as rainhas que estivessem em colunas com elevada frequência permaneceriam fixas em suas posições, e as demais estariam livres para a realização de movimentos. Assim, a busca se restringiria a trocas que envolvessem apenas as rainhas que estivessem em colunas livres.

10. Oscilação estratégica

Para muitos problemas combinatórios, pode não ser eficaz explorar o espaço de soluções por meio apenas de movimentos que respeitem todas as restrições do problema. Isso acontece, por exemplo, em problemas de roteamento de veículos, nas situações em que os veículos estejam operando próximos à capacidade máxima, limitando, com isso, a movimentação de clientes entre as rotas (Gendreau; 2003). Nesses casos, a relaxação de restrições pode ser uma estratégia atrativa, uma vez que ela cria um espaço de soluções mais amplo, que pode ser explorado por movimentos mais simples. Para implementar essa estratégia, basta desconsiderar a satisfação das restrições selecionadas do espaço de busca e penalizá-las na função objetivo sempre que elas forem violadas.

Quando os pesos atribuídos às violações das restrições relaxadas são sistematicamente modificados ao longo da busca, isto é, ora aumentados, ora diminuídos, tem-se a chamada "oscilação estratégica" ou "relaxação adaptativa". Ao aumentar a penalização de movimentos que conduzem a soluções infactíveis, estimula-se o procedimento de busca a entrar na região de soluções factíveis. Ao contrário, quando tal penalização é reduzida ou mesmo anulada, estimula-se o procedimento a caminhar no espaço de soluções infactíveis. Portanto, o uso dessa estratégia permite à busca alternar entre soluções factíveis e infactíveis, daí o nome "oscilação". De acordo com Gendreau (2003), tal estratégia teve origem no trabalho de Glover (1977).

Em Gendreau et al. (1994), os autores tratam o Problema de Roteamento de Veículos penalizando o excesso de carga encontrado em uma solução por um peso ρ, o qual é multiplicado por um fator α que varia de acordo com o seguinte esquema:

1. No início da busca $\alpha \leftarrow 1$.
2. A cada k iterações sem melhora:
   - se todas as k soluções visitadas forem factíveis então $\alpha \leftarrow \alpha/\gamma$;
   - se todas as k soluções visitadas forem infactíveis então $\alpha \leftarrow \alpha \times \gamma$;
   - se algumas soluções forem factíveis e outras infactíveis, então α permanecerá inalterado.

O parâmetro γ é fixado em 2. Assim, quando somente soluções factíveis são geradas, o peso ρ é reduzido, estimulando a geração de soluções infactíveis. Ao contrário, quando somente soluções infactíveis são geradas, o peso ρ é aumentado, encorajando a busca a entrar novamente no espaço das soluções factíveis. O valor de α é limitado por duas constantes $\alpha_{\min}$ e $\alpha_{\max}$, para evitar que a estratégia de oscilação aumente (ou diminua) indefinidamente os pesos.

Estratégia semelhante pode ser aplicada ao Problema da Mochila 0-1 tratado na Seção 2. Nesse caso, usaríamos a função (1.4), apresentada à página 4, para avaliar uma solução. A penalidade ρ relativa à existência de uma solução infactível seria, assim como no exemplo anterior, multiplicada por um fator α.

Em Díaz and Fernández (2001), os autores variam o peso atribuído ao parâmetro ρ da Eq. (1.19) de acordo com a expressão (1.21).

$$\rho \leftarrow \rho \times \alpha^{(ninv/(niter-1)-1)} \tag{1.21}$$

em que *ninv* é o número de soluções infactíveis obtidas durante as últimas *niter* iterações e α é um parâmetro que varia conforme a seguir se descreve. Essa expressão busca adaptar o parâmetro ρ ao estágio da busca, incorporando uma memória de curto prazo (*ninv*) e uma memória de médio prazo (α).

Para um dado valor atribuído a α, o valor de ρ varia no intervalo $[\alpha^{-1} \times \rho, \alpha^{1/(niter-1)} \times \rho]$. Como $\alpha \geq 1$, então ρ somente aumenta de valor quando *todas* as soluções encontradas nas últimas *niter* iterações são infactíveis. Contudo, mesmo nesse caso, o limite superior do intervalo faz com que o acréscimo no valor de ρ seja bem pequeno.

Quando $\alpha = 1$ na expressão (1.21), ρ se torna fixo durante todo o processo e a história da busca não é levada em consideração. No caso em que $\alpha = 2$, a expressão obtida é bastante similar à de Gendreau et al. (1996). Observa-se que para α fixo durante a busca, apenas memória de curto prazo é usada.

Os autores observam que o valor atribuído a α influencia consideravelmente o progresso da busca. Quanto maior for este valor, maior é o intervalo de variação para ρ. Portanto, quando somente soluções infactíveis são geradas nas últimas iterações, a razão de incremento de ρ é maior para valores mais elevados de α. A variação no valor de α é, no entanto, pequena para evitar mudanças súbitas no procedimento de busca. Inicialmente, $\alpha = 1$ e $\rho = 1$. Após a estratégia de oscilação encontrar a primeira solução factível, α assume valor 2. Quando a melhor solução encontrada até então não é alterada nas últimas 100 iterações, o valor de α é aumentado em 0,005 a cada 10 iterações, ficando limitado ao seu valor máximo, 3. Se, no entanto, uma solução de melhor qualidade é encontrada, o valor de α é reinicializado em seu patamar mínimo, 2.

Referências Bibliográficas

Allahverdi, A., Ng, C. T., Cheng, T. C. E. and Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs, *European Journal of Operational Research* **187**: 985–1032.

Archetti, C., Speranza, M. G. and Hertz, A. (2006). A tabu search algorithm for the split delivery vehicle routing problem, *Transportation Science* **40**: 64–73.

Battiti, R. (1996). Reactive search: Toward self-tuning heuristics, *in* V. Rayward-Smith, I. Osman, C. Reeves and G. Smith (eds), *Modern Heuristic Search Methods*, John Wiley & Sons, New York, chapter 4, pp. 61–83.

Battiti, R. and Tecchiolli, G. (1994). The reactive tabu search, *ORSA Journal of Computing* **6**: 126–140.

Buscher, U. and Shen, L. (2009). An integrated tabu search algorithm for the lot streaming problem in job shops, *European Journal of Operational Research* **199**: 385–399.

Cordeau, J. F., Gendreau, M., Laporte, G., Potvin, J. Y. and Semet, F. (2002). A guide to vehicle routing problem, *Journal of the Operational Research Society* **53**: 512–522.

de Werra, D. and Hertz, A. (1989). Tabu search techniques: A tutorial and an application to neural networks, *OR Spektrum* **11**: 131–141.

Díaz, J. A. and Fernández (2001). A tabu search heuristic for the generalized assignment problem, *European Journal of Operational Research* **132**: 22–38.

França, P. M. (2009). Busca tabu. Disponível em http://www.densis.fee.unicamp.br/ franca/EA043/Transpa-Cap-4a.pdf.

Gendreau, M. (2002). Recent advances in tabu search, *in* C. C. Ribeiro and P. Hansen (eds), *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, pp. 369–377.

Gendreau, M. (2003). An introduction to tabu search, *in* F. Glover and G. A. Kochenberger (eds), *Handbook of Metaheuristics*, Kluwer Academic Publishers, chapter 2, pp. 37–54.

Gendreau, M., Guertin, F., Potvin, J.-Y. and Taillard, E. D. (1999). Parallel tabu search for real-time vehicle routing and dispatching, *Transportation Science* **33**: 381–390.

Gendreau, M., Hertz, A. and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem, *Management Science* **40**: 1276–1290.

Gendreau, M., Laporte, G. and Séguin, R. (1996). A tabu search heuristic for the vehicle routing problem with stochastic demands and customers, *Operations Research* **44**: 469–477.

Gendreau, M., Soriano, P. and Salvail, L. (1993). Solving the maximum clique problem using a tabu search approach, *in* F. Glover, M. Laguna and E. Taillard (eds), *Tabu Search*, Vol. 41 of *Annals of Operations Research*, J. C. Baltzer AG, pp. 385–403.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints, *Decision Sciences* **8**: 156–166.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research* **5**: 553–549.

Glover, F. (1989). Tabu search: Part i, *ORSA Journal of Computing* **1**: 190–206.

Glover, F. (1990). Tabu search: Part ii, *ORSA Journal of Computing* **2**: 4–32.

Glover, F. and Laguna, M. (1993). Tabu search, *in* C. R. Reeves (ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Advanced Topics in Computer Science Series, Blackwell Scientific Publications, London, chapter 3, pp. 70–150.

Glover, F. and Laguna, M. (1997). *Tabu Search*, Kluwer Academic Publishers, Boston.

Glover, F., Taillard, E. and de Werra, D. (1993). A user's guide to tabu search, *in* P. L. Hammer (ed.), *Tabu Search*, Vol. 41 of *Annals of Operations Research*, Baltzer Science Publishers, Amsterdan, pp. 3–28.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming, *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

Hertz, A. and de Werra, D. (1990). The tabu search metaheuristic: how we used it, *Annals of Mathematics and Artificial Intelligence* **1**: 111–121.

Laguna, M. (1994). A guide to implement tabu search, *Investigación Operativa* **4**: 5–25.

Osman, I. H. (1993). The vehicle routing problem, *in* F. Glover, M. Laguna and E. Taillard (eds), *Tabu Search*, Vol. 41 of *Annals of Operations Research*, J. C. Baltzer AG, pp. 421–451.

Ronconi, D. and Armentano, V. (2009). Busca tabu para a minimização do tempo total de atraso no problema de flowshop, *Pesquisa Operacional para o Desenvolvimento* **1**: 50–62.

Santos, H. G., Ochi, L. S. and Souza, M. J. F. (2005). A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem, *ACM Journal of Experimental Algorithmics* **10**: art–2.09. 15 p.

Souza, M. J. F. (2000). *Programação de horários em escolas: uma aproximação por metaheurísticas*, Tese de doutorado, Programa de Engenharia de Sistemas e Computação, COPPE, Universidade Federal do Rio de Janeiro.

Souza, M. J. F., Ochi, L. S. and Maculan, N. (2004). A grasp-tabu search algorithm for solving school timetabling problems, *in* M. G. C. Resende and J. P. Souza (eds), *Metaheuristics: Computer Decision-Making*, Vol. 1153, Kluwer Academic Publishers, pp. 659–672.

Wan, G. and Yen, B. P. C. (2002). Tabu search for single machine scheduling with distinct due windows and weighted earliness/tardiness penalties, *European Journal of Operational Research* **142**: 271–281.

Yagiura, M., Glover, F. and Ibaraki, T. (2006). A path relinking approach with ejection chains for the generalized assignment problem, *European Journal of Operational Research* **169**: 548–569.

Índice Remissivo