4

Busca Tabu Reativa para Problemas de Códigos de Cobertura

Neste capítulo descrevemos uma aplicação da metaeurística busca tabu reativa para os problemas de códigos de cobertura. É apresentado inicialmente o esquema básico da busca, que pode ser entendido como a busca tabu clássica para os problemas em questão. Em seguida apresentamos o mecanismo de reação que é incorporado ao esquema básico para formar a busca tabu reativa, mostrando como este mecanismo contribui para o esquema básico da busca tabu.

Carnielli et al. [6] e Östergård [35] usam metaeurísticas para encontrar limites superiores para R-coberturas clássicas. Nenhum dos trabalhos utilizam a variação reativa da busca tabu.

A busca tabu, Glover [14, 15], é um procedimento adaptativo que tem sido aplicado a uma vasta gama de problemas de otimização combinatória. Ela é um algoritmo de busca baseado em caminho, já que o conjunto de soluções visitadas pela busca formam um caminho no espaço de busca do problema. O principal objetivo da busca tabu é evitar ciclagem entre soluções e intensificar a busca em regiões promissoras do espaço. Para se mover da solução atual para uma outra é necessário definir a vizinhança de uma solução. O movimento nessa vizinhança é quase sempre um passo guloso. Entretanto só se é permitido mudar da solução atual para a vizinha se o movimento necessário para isto não esteja classificado como "tabu", isto é, o movimento não esteja proibido. A descrição dos detalhes do algoritmo desenvolvido para busca de conjuntos dominantes no grafo G(n,q,R) = (V,E) é feita mais adiante.

O esquema clássico da busca tabu mantém uma lista de movimentos classificados como tabu (lista tabu), tendo esta lista um tamanho fixo, o que permite ciclos na busca de tamanho maior que o tamanho da mesma. Embora este esquema possa levar a excelentes resultados, ainda é possível a busca ficar presa a regiões de mínimos locais do espaço de busca, como descrito em Battiti e Tecchiolli [3]. Uma idéia para tentar evitar este problema, também

apresentada em [3], é a utilização de um mecanismo de reação na busca tabu. A idéia propõe combinar os principais elementos da busca tabu clássica com uma memória de longa duração que mantém a história das soluções visitadas durante a busca. Esta memória é utilizada para controlar dinamicamente o tamanho da lista tabu, aumentando este tamanho quando a ocorrência de ciclos se torna frequente e o diminuindo quando uma nova região do espaço começa a ser explorada. Além do controle dinâmico do tamanho da lista tabu, a busca tabu reativa usa outro mecanismo para escapar de mínimos locais. Ele consiste na aplicação de sucessivas perturbações aleatórias na solução atual quando o número de soluções visitadas supera um determinado limiar. A busca tabu reativa tem apresentado sucesso em diversas aplicações como demonstrado nos trabalhos de Battiti e Tecchiolli [3] e Osman e Wassan [31].

Na sequência é descrito como a estrutura básica da busca tabu é utilizada e como o mecanismo de reação é incorporado à mesma.

4.1
O Esquema Básico da Busca Tabu

4.1.1
Espaço de Busca

O esquema da busca tabu se inicia pela definição do espaço de busca. No nosso caso consideramos o espaço como sendo o conjunto de todos $2^{(q^n)}$ subconjuntos de $\mathbb{F}_q^n$ (o conjunto de vértices de $G(n, q, R)$). Como consequência, nem todos os elementos deste espaço formam um conjunto dominante, isto é, um subconjunto de V não necessariamente é uma solução viável para o problema original. Esta escolha implica em trabalhar no que é, as vezes, chamado de espaço de busca estendido. A vantagem de trabalhar neste espaço é que nele se tem uma estrutura de vizinhança natural e a desvantagem é a necessidade de utilização de um mecanismo de penalidade de forma a explorar de forma frequente conjuntos dominantes, isto é, soluções viáveis.

4.1.2

Vizinhança e Fator de Penalidade

A vizinhança natural, mencionada anteriormente, é dada pela adição de um vértice que não esteja no conjunto de vértices da solução atual ou a remoção de um vértice deste. Isto leva a uma vizinhança de tamanho $|V| = q^n$, o que pode ser, em alguns casos, uma vizinhança grande. A função objetivo, que é o tamanho do conjunto dominante, no nosso caso também possui um componente de penalidade. Este componente é o produto de um fator de penalidade com o número de vértices não cobertos pela solução atual. Quanto maior é o fator de penalidade menores são as chances de não obtermos soluções viáveis (conjuntos dominantes) durante a busca. Porém se o fator de penalidade é muito alto a busca passa a ser muito restritiva se comportando como se apenas soluções viáveis fossem permitidas. Portanto, ajustar este fator de penalidade é uma importante tarefa na "calibração" da busca tabu, já que ele balanceia a liberdade da busca com a necessidade de obtermos soluções viáveis.

A estratégia adotada para trabalhar com o fator de penalidade (ALPHA) é iniciá-lo com um pequeno valor (MIN_ALPHA), aumentá-lo a cada iteração por um valor constante (STEP_ALPHA), até que o mesmo atinja um valor máximo (MAX_ALPHA), quando então seu valor é resetado para o valor inicial (MIN_ALPHA). Esta estratégia permite a busca caminhar livre no início de cada ciclo e então convergir gradualmente para uma solução viável. Após uma quantidade razoável de experimentos para encontrar os melhores valores para os parâmetros descritos, foram adotados os valores 0.1, 1.0 e 0.001 para MIN_ALPHA, MAX_ALPHA e STEP_ALPHA, respectivamente.

4.1.3

Lista Tabu e Critério de Aspiração

Outro elemento principal da busca tabu é sua lista tabu. A lista adotada guarda a última iteração onde um movimento (adição ou remoção de um vértice) foi tornado tabu, assim são guardados dois valores por vértice de V, a última iteração em que foi adicionado a solução e a última iteração em que foi removido da mesma, contemplando assim todos os possíveis movimentos. Assim, para verificarmos se um movimento m é tabu basta verificar se a diferença de seu valor na lista tabu e o número da iteração atual é maior que o tamanho da lista tabu, usualmente referenciado como *tabu tenure*. Este

valor do tabu tenure será objeto do mecanismo de reação, que determinará seu valor dinamicamente, como será descrito na próxima seção.

O último elemento principal da busca tabu é o chamado critério de aspiração, que determina os casos onde o critério tabu pode ser violado para um movimento. O critério de aspiração utilizado revoga a restrição tabu de um movimento quando o mesmo leva a uma solução cujo valor de função objetivo da mesma é o melhor já encontrado durante a busca. A seguir apresentamos o pseudo-código do esquema básico da busca tabu que foi adotado.

4.1.4

O algoritmo do Esquema Básico da Busca Tabu

Inicializações.

- Construir uma cobertura inicial U adicionando aleatoriamente palavras, que não estão em U, até que U se torne uma cobertura.
- $BestCode \leftarrow U$.
- $T[M] \leftarrow 0$, para todo movimento possível M.
- $alpha \leftarrow MIN\_ALPHA$.
- Inicializa o mecanismo de reação.

Loop Principal. Repita NUMBER-OF-ITERATIONS vezes.

- Atualiza o Fator de Penalidade ($alpha$).
- Seleciona o melhor movimento M que não é tabu ou satisfaça o critério de aspiração.
- Realizar o movimento M em U.
- Setar o valor tabu de M em T, $T[M] \leftarrow iteracao - atual$.
- **Se** U é uma cobertura e $|U| < |BestCode|$
    - $BestCode \leftarrow U$.
- **Chama o mecanismo de reação**.

O algoritmo inicia construindo uma solução aleatória viável U(um conjunto dominante de $G(n, q, R)$). Em seguida, os elementos da busca tabu são inicializados. O loop principal é repetido um número suficientemente grande

de vezes usualmente dado pelo tempo disponível de CPU. Ele ajusta o fator de penalidade, testa o critério tabu dos movimentos e seu critério de aspiração, atualiza a melhor solução encontrada e ativa o mecanismo de reação, que é descrito na próxima seção.

4.2 O Mecanismo de Reação

4.2.1 Controle Dinâmico do Tabu Tenure

O controle do tabu tenure e a eventual realização de uma mudança aleatória na solução atual são os principais elementos do mecanismo de reação. A idéia principal é manter uma memória de longo prazo que mantém o conjunto das soluções visitadas, a iteração da última visita e o número de visitas às mesmas. O objetivo desta memória é detectar a ciclagem entre soluções. Cada vez que uma solução se repete na busca o intervalo entre as visitas é então calculado. A chamada reação rápida ocorre quando este intervalo é menor que um dado limiar ("threshold"). A reação consiste em atribuir ao tabu tenure um valor grande. A chamada reação lenta segue gradualmente reduzindo o valor do tabu tenure, já que se espera atingir uma nova região do espaço de busca ao atribuir ao tabu tenure um valor alto.

4.2.2 Mecanismo de Escape

Além deste controle dinâmico do valor do tabu tenure, existe outro mecanismo de escape de mínimos locais. Este mecanismo trabalha monitorando as soluções que são visitadas um número excessivo de vezes. Quando o número destas soluções excede um determinado limiar, um determinado número de movimentos aleatórios é realizado na solução atual, na esperança de que a busca atinja uma região do espaço ainda não explorada.

A combinação destes dois mecanismos torna a busca tabu reativa uma importante melhoria no busca tabu clássica, melhorando seus resultados em diversas aplicações da busca tabu.

4.2.3

O algoritmo do Mecanismo de Reação

A seguir é apresentado o pseudo-código do mecanismo de reação. Para implementá-lo algumas estruturas de dados e variáveis são necessárias. Elas são: (i) o conjunto $Visited$ que mantém as informações das soluções visitadas durante a busca; (ii) o conjunto $OftenRepeated$ que armazena as soluções visitadas um número excessivo de vezes, sendo sua cardinalidade mantida na variável *chaotic*; e (iii) variáveis $countLastSizeChange$, que mantém o número de iterações desde a última mudança no tabu tenure, e $movingAvg$ que armazena o tamanho médio dos ciclos completados durante a busca.

A inicialização do mecanismo de reação apenas cria e inicializa estas variáveis e conjuntos. O algoritmo correspondente segue abaixo.

Inicialização do Mecanismo de Reação

- Cria os conjuntos vazios $Visited$ e $OftenReapeated$.
- $chaotic \leftarrow 0$.
- $countLastSizeChange \leftarrow 0$.
- $movingAvg \leftarrow 0$.
- $tabuTenure \leftarrow MIN\_TENURE$.

Mecanismo de Reação

- $escape \leftarrow true$.
- $countLastSizeChange++$.
- **Se** U está em $Visited$.
  - $TamCycle \leftarrow$ iteração atual - iteração da última visita de U.
  - Incrementa o número de visitas a U e atualiza a iteração da última visita a U.
  - **Se** $TamCycle < CYCLE\_MIN$.
    - $movingAvg \leftarrow 0.9 * movingAvg + 0.1 * TamCycle$.
    - $tabuTenure \leftarrow$ *min(tabuTenure * INCREASE, MAX_TENURE)*.
    - $countLastSizeChange \leftarrow 0$.
  - **Se** número de visitas a $U > MAX\_VISITS$ e U não está em $OftenRepeated$

- Insere U em $OftenRepeated$ e incrementa *chaotic*.
- **Se** $chaotic > Chaos$.
  - $escape \leftarrow true$.

- **Senão**
  - Insere U em $Visited$ e seta o número de visitas a U para um e a última iteração de visita a U para a iteração atual.
  - **Se** $countLastSizeChange > movingAvg$.
    - $tabuTenure \leftarrow$ *max(tabuTenure * DECREASE, MIN_TENURE)*.
    - $countLastSizeChange \leftarrow 0$.

- **Se** *escape* é *true*.
  - Esvazia o conjunto $Visited$.
  - $chaotic \leftarrow 0$.
  - Realiza um determinado número de movimentos aleatórios em U.

O mecanismo de reação é invocado após cada movimento na solução atual. Ele primeiro testa se a solução atual já foi visitada anteriormente. Quando isto é verdade, o intervalo entre as visitas é então calculado e se for menor que CYCLE_MIN o mecanismo de reação rápido age incrementando o valor do tabu tenure através da multiplicação do mesmo pela constante INCREASE (INCREASE > 1), atualizando a variável $movingAvg$ e o conjunto $OftenRepeated$. Quando o número de visitas a esta solução excede a constante MAX_VISITS e a solução ainda não está no conjunto $OftenRepeated$ a mesma é então inserida neste conjunto. O último teste verifica se a cardinalidade deste conjunto, o valor da variável *chaotic*, se tornou maior que a constante Chaos, quando então o mecanismo de escape é então disparado para se mover aleatoriamente da solução atual.

Quando a solução atual é visitada pela primeira vez, então o mecanismo de reação lenta atua reduzindo o valor do tabu tenure no caso em que o valor da variável $movingAvg$ é menor que o da variável $countLastSizeChange$. Isto é feito através da multiplicação do tabu tenure por uma constante DECREASE ($0 <$ DECREASE < 1). Além disto, todas as operações com respeito a primeira visita a uma solução são então executadas, de forma que visitas subsequentes sejam detectadas. A implementação também usa as constantes MIN_TENURE e MAX_TENURE para limitar o valor do tabu tenure, evitando valores excessivamente pequenos ou grandes que são indesejados.

Analisando o algoritmo descrito anteriormente, fica claro que implementar o mecanismo de reação de forma estrita possivelmente irá demandar uma

grande quantidade de memória e/ou uma estrutura de dados muito poderosa que lide com compressão e tenha um tempo de acesso rápido para manter o conjunto de soluções visitadas. Portanto, ao invés de mantermos um conjunto real de soluções, é mantido uma hash table que tem como função de hash uma string parâmetro associada com a solução a ser armazenada. Esta string é obtida pela concatenação do número de vértices cobertos por exatamente i vértices da solução atual, tendo i variando de 1 a 6.

Para exemplificar a geração da string de hash utilizaremos como base as soluções para o $K_2(3,1)$ apresentadas na seção 2.3.2 nas figuras 2.3, 2.4 e 2.5, sendo as mesmas referenciadas por s_1, s_2 e s_3 respectivamente. Seja $s_1 = \{(000),(010),(111)\}$ então cada vértice do grafo formado a partir do $K_2(3,1)$ é coberto pelas palavras de s_1 da seguinte maneira (como pode ser verificado pela figura 2.3):

- 000 é coberto por 000 e 010.
- 001 é coberto por 000.
- 010 é coberto por 000 e 010.
- 011 é coberto por 010 e 111.
- 100 é coberto por 000.
- 101 é coberto por 111.
- 110 é coberto por 010 e 111.
- 111 é coberto por 111.

Assim temos que 000, 010, 011 e 110 são cobertos por exatamente duas palavras de s_1 enquanto 001, 100, 101 e 111 são cobertos por exatamente uma palavra de s_1. Como a string de hash é obtida pela concatenação do número de vértices cobertos por exatamente $1 \leq i \leq 6$ vértices da solução, então a string de hash gerada para s_1 será h_1 = "4 4 0 0 0 0". Repetindo o mesmo processo para $s_2 = \{(000),(010)\}$ e $s_3 = \{(001),(110)\}$ as strings de hash geradas para as mesmas serão h_2 = "4 2 0 0 0 0" e h_3 = "8 0 0 0 0 0". Como as três soluções e suas respectivas strings de hash geradas são diferentes podemos verificar que o esquema de geração da hash foi capaz de diferenciar corretamente as soluções.

No caso do $K_2(3,1)$ não observamos uma economia de memória ao utilizarmos as strings de hash ao invés da representação real das soluções, isto se deve ao fato desta instância ser trivial e portanto servir apenas para ilustrar a geração da string de hash. Fica evidente que para instâncias maiores

o esquema sugerido de geração de hash é menos custoso em termos de gasto de memória do que o armazenamento completo das soluções visitadas.

Embora este esquema tenha mostrado diferenciar de forma apropriada as soluções em nossos testes, ele certamente gera falsos positivos, ou seja, considerar duas soluções diferentes como iguais, o que irá disparar o mecanismo que trata de soluções já visitadas de forma indevida. O fato dos resultados obtidos terem sido de boa qualidade, como será mostrado no capítulo 6, nos evidencia que os possíveis falsos positivos ocorridos não prejudicaram de forma significativa a busca.