Uma modificação do algoritmo GRASP para o problema de cobertura de conjunto de custo único

Guilherme Rolim e Souza*

Universidade Federal Fluminense
Junho de 2014

Resumo

O problema de cobertura de conjunto (SCP - *Set Covering Problem*) é um problema bastante conhecido na área de otimização. Este artigo realizou modificações na área de busca local de um algoritmo GRASP, já conhecido na literatura, para o SCP de custo único. As eficiências de ambos os algoritmos são testadas em variadas instâncias e comparadas.

1 Introdução

O problema de cobertura de conjunto pode ser descrito da seguinte forma: dado um conjunto M, onde $|M| = m$ e n subconjuntos $S_j \subseteq M$, onde $j \in N = 1, ..., n$ cada um contendo um custo não negativo c_j, o objetivo é minimizar o custo de subconjuntos S_j de tal forma que cada elemento de M pertença a pelo menos um destes subconjuntos. Uma formulação matemática deste problema pode ser descrita como:

$$min \sum_{j=1}^{n} c_j.x_j$$

Sujeito a:

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

$$x_j \in 0, 1, j \in N. \tag{2}$$

A variável x_j é igual a 1 se o subconjunto S_j pertence a solução e 0 caso contrário. O coeficiente a_{ij} é 1 se o elemento i pertence a S_j e 0 caso contrário. A matriz $A = (a_{ij})$, $i = 1, ..., m$, $j = 1, ..., n$ é chamada de matriz de cobertura. Na matriz de cobertura cada linha

*grolim@outlook.com.br

representa um elemento a ser coberto e cada coluna um subconjunto. A restrição (1) garante que todos os elementos do conjunto M serão cobertos por pelo menos um subconjunto.

A função de minimização consta com a variável c_j que indica o custo de escolha de um determinado subconjunto. Este trabalho aborda apenas a variante do SCP de custo único, introduzida por (TOREGAS et al., 1971). Por este motivo, a variável c_j pode ser ignorada, visto que, o custo de escolha de cada subconjunto é idêntico e não influencia na função de minimização.

O restante deste trabalho está organizado da seguinte forma, a seção 2 explicará o algoritmo GRASP proposto por (BAUTISTA; PEREIRA, 2007), que serviu de base para este trabalho, e detalhes sobre como foi implementado. A seção 3 explica as mudanças realizadas sobre o algoritmo implementado. A seção 4 mostra os resultados dos experimentos computacionais, comparando ambos os algoritmos. Por fim, a seção 5 conclui este trabalho.

2 O algoritmo

2.1 A metaheurística GRASP

A metaheurística GRASP (FEO; RESENDE, 1995), tipicamente, consiste de uma série de iterações feitas a partir de sucessivas construções gulosas aleatórias que, por fim, são melhoradas por uma busca local. O algoritmo GRASP é dividido em duas fases, a fase de construção e a fase de melhoria. A fase de construção retorna uma solução válida baseando-se em uma lista de candidatos restritos (RCL - Restricted Candidate List) montada a cada interação a partir dos melhores candidatos. É através desta lista que o GRASP se diferencia de uma heurística gulosa convencional. Enquanto a heurística gulosa sempre busca o melhor candidato, a metaheurística GRASP escolhe um bom candidato aleatoriamente, introduzindo diversidade ao procedimento. A fase de melhoria atua sobre a solução construída na fase de construção, permitindo que o procedimento encontre soluções ainda melhores. O algoritmo representado a seguir exibe de forma mais clara o funcionamento do GRASP.

```
 Algoritmo 1:  Bloco principal da metaheurística GRASP
1.MelhorSolução <- Inicialização()
2.Para todo i até Numero_de_iterações
   a.  Solução <- Fase_de_Construção()
   b.  Solução <- Fase_de_Melhoria(Solução)
   c.  Atualização (Solução, MelhorSolução)
3.Fim
```

A fase de construção e de melhoria serão melhor explicadas a seguir. A atualização ocorre ao final de cada iteração, atualizando a `MelhorSolução` caso a solução resultante seja melhor.

2.2 Fase de Construção

A fase de construção inicia a partir de uma solução trivial e inválida, isto é, todos os valores da solução são 0, indicando que nenhum subconjunto faz parte da solução. A qualidade de cada

subconjunto baseia-se no número de elementos, ainda não cobertos, que este cobrirá se fizer parte da solução. A cada iteração uma lista de candidatos restritos (RCL) é criada a baseada na qualidade do melhor candidato e um parâmetro de deteriorização α. O procedimento 1 ilustra a fase de construção.

```
 Procedimento 1:  Fase de construção
1.Inicialização()
2.Enquanto houver elementos descobertos
   a.  MaiorQualidade = A qualidade do melhor candidato
   b.  Determinar o limite L
      L = α.MaiorQualidade
   c.  Montar lista de candidatos restritos
      RCL = { candidatos cuja qualidade é maior ou igual a L}
   d.  Escolher um candidato da RCL aleatoriamente
      Escolhido = Random(RCL)
   e.  Adicionar o candidato Escolhido à solução
3.Fim
```

A fase de construção sempre retornará uma solução válida, ou seja, todos os elementos são cobertos. A fase de melhoria, no entanto, é capaz de retornar uma solução inválida. Isto acontece pois durante a fase de melhoria é introduzido um método para escapar de ótimos locais, conforme será mostrado na subseção seguinte.

2.3 Fase de melhoria

As soluções geradas pela fase de construção nem sempre são ótimas. Por este motivo, um método de busca local se faz necessário. A busca local implementada é capaz de melhorar a solução gerada pela fase de construção além de permitir que o algoritmo escape de ótimos locais. O procedimento 2 mostra a fase de melhoria implementada. A função BestFlip realiza o melhor Flip na solução. Um Flip é a troca de um valor da solução de 1 para 0 ou de 0 para 1. A função RandomFlip realiza um Flip em uma posição aleatória da solução. Através do RandomFlip este procedimento é capaz de escapar de ótimos locais. O BestFlip é realizado com probabilidade p e o RandomFlip com probabilidade 1-p.

```
 Procedimento 2:  Fase de melhoria
1.MelhorSolução <- Solução
2.Para todo i até MAXFLIPS
   a.  Se Random[0,1] < p
      Solução <- BestFlip(Solução)
   b.  Senão
      Solução <- RandomFlip(Solução)
   c.  Fim Se
   d.  Atualização (Solução, MelhorSolução)
3.Fim Para
```

A solução resultante desta fase pode ser inválida, devido à perturbação gerada pelo método RandomFlip. Foi determinado que soluções inválidas são descartadas, de forma a manter válida a restrição que garante que todos elementos serão cobertos por pelo menos um subconjunto.

3 Modificações no algoritmo

Foram criadas 3 novos métodos para a fase de melhoria, são eles: BestDualFlip, RandomDualFlip, RandomTripleFlip. O método BestDualFlip é bem parecido com o BestFlip do algoritmo original, no entanto, ao invés de realizar apenas o melhor Flip na solução, o BestDualFlip realiza os dois melhores Flips possíveis. O objetivo do BestDualFlip é melhorar a solução de forma mais eficaz embora a quantidade de tentativas a ser analisada seja maior. Os métodos RandomDualFlip e RandomTripleFlip, realizam 2 e 3 Flips aleatoriamente, o objetivo destes métodos é causar uma maior perturbação na solução. Para um primeiro teste, o RandomFlip na fase de melhoria do algoritmo original foi substituído pelo RandomDualFlip e pelo RandomTripleFlip, no entanto, a perturbação causada por estes métodos pioraram muito a solução, impedindo que o BestFlip pudesse melhorá-las. O algoritmo modificado final substitui o BestFlip pelo BestDualFlip, conforme mostra o procedimento 3.

```
 Procedimento 3:  Fase de melhoria modificada
1.MelhorSolução <- Solução
2.Para todo i até MAXFLIPS
   a.  Se Random[0,1] < p
      Solução <- BestDualFlip(Solução)
   b.  Senão
      Solução <- RandomFlip(Solução)
   c.  Fim Se
   d.  Atualização (Solução, MelhorSolução)
3.Fim Para
```

4 Resultados Computacionais

O algoritmo foi programado em C++ e compilado usando o VC++ 2012. Todos os testes foram realizados em uma única *thread* em um Intel Core i5 de 3 GHz com 8 Gb de memória RAM no sistema operacional Windows 8.1. Para o algoritmo original, as configurações especificadas foram: Número_de_Iterações = 200, $\alpha = 0.9$, MAXFLIPS = 10*n e p = 0.75. Para o algoritmo modificado, o número de MAXFLIPS foi reduzido para n/10, pois a complexidade do método BestDualFlip é muito superior ao BestFlip.

As instâncias utilizadas neste teste foram retirados da OR-Library (BEASLEY, 1990). Detalhes destas instâncias podem ser vistos na Tabela 1.

Tanto o algoritmo original quanto o modificado foram executados 5 vezes para cada uma destas instâncias. Os resultados obtidos podem ser vistos na Tabela 2. A sigla M.R representa o melhor resultado e M.R* a média dos melhores resultados. T.M.R o tempo para encontrar o melhor resultado, T.M.R* o tempo médio para encontrar o melhor resultado e T.T.M o tempo

Instância	Linhas(m)	Colunas(n)	Densidade(%)	Número máximo de 1's por linha	Tipo de problema
4.1	200	1000	2	36	Random
5.1	200	2000	2	60	Random
6.1	200	1000	5	71	Random
A.1	300	3000	2	81	Random
E.1	50	500	20	3	Random
CLR.10-4	511	210	12.3	10-126	Combinatorial
CYC.7	672	448	0.9	4-4	Logical

Tabela 1 – Dados das instâncias utilizadas.

Instância	M.R-O	M.R*-O	M.R-M	M.R*-M
4.1	40	40	60	62.2
5.1	37	37.2	79	82.4
6.1	21	22	40	43
A.1	42	42	106	110.6
E.1	6	6	12	12.4
CLR.10-4	25	25.6	32	32.6
CYC.7	154	155.2	172	173.2

Tabela 2 – Resultados obtidos pelo algoritmo original e o algoritmo modificado.

Instância	T.M.R-O(s)	T.M.R*-O(s)	T.T.M-O(s)	T.M.R-M(s)	T.M.R*-M(s)	T.T.M-M(s)
4.1	16	42.2	64.8	54	79.4	132.8
5.1	3	42.6	227.4	401	341.8	948.4
6.1	32	53.2	121.8	128	146.4	386.4
A.1	107	175	673	2367	1637	4298.6
E.1	0	3	28	22	19.4	53.8
CLR.10-4	4	8.2	90.6	1	9	53
CYC.7	48	23.4	85.8	46	37.8	84.2

Tabela 3 – Comparação de tempo entre os algoritmos.

total médio de execução do algoritmo. As identificações -O e -M indicam que os dados são referentes ao algoritmo original e modificado respectivamente.

5 Conclusão

Embora o método BestDualFlip analise uma quantidade maior de modificações na solução do que o BestFlip, os experimentos computacionais mostram que o algoritmo modificado não conseguiu obter resultados melhores que o original e que seu tempo de execução foi muito superior, principalmente nas instâncias grandes. A possível causa da má performance do BestDualFlip é porque a fase de construção já retorna uma boa solução e a realização de dois Flips altera excessivamente esta solução de forma a perturbá-la e não melhorá-la. Para trabalhos futuros, uma junção de ambos os algoritmos pode ser testada, de forma a reduzir o número de execuções do BestDualFlip que é muito custoso. Os algoritmos implementados estão disponibilizados em `https://github.com/loxorolim/GraspPO/`.

Referências

BAUTISTA, J.; PEREIRA, J. A grasp algorithm to solve the unicost set covering problem. *Computers & Operations Research*, Elsevier, v. 34, n. 10, p. 3162–3173, 2007. 2

BEASLEY, J. E. Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, JSTOR, p. 1069–1072, 1990. 4

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995. 2

TOREGAS, C. et al. The location of emergency service facilities. *Operations Research*, INFORMS, v. 19, n. 6, p. 1363–1373, 1971. 2