Análise de Algoritmos Heurísticos de Construção e Melhoramento para o problema de Cobertura de Conjunto (Set Covering)

Universidade Estadual de Maringá
Ciência da Computação - Departamento de Informática
6903/01 - Modelagem e Otimização de Algoritmos
Prof. Ademir Aparecido Constantino
Acadêmico: Marcos Vinicius de Oliveira RA: 124408

1. Introdução

Este documento tem como objetivo a análise da experimentação dos resultados da utilização de algoritmos heurísticos para solucionar o problema de Cobertura de Conjuntos, utilizando o algoritmo guloso proposto por Vasco and Wilson, um algoritmo guloso de seleção de melhor coluna imediata e o algoritmo de melhoramento proposto por Jacobs and Brusco (1995).

2. Descrição do problema

Dado um conjunto finito *U* de elementos, chamado de "universo", e uma coleção *S* de subconjuntos de *U* (chamados de "conjuntos"), o objetivo é encontrar o menor número de conjuntos cuja união seja igual a *U*. Em outras palavras, deseja-se encontrar um subconjunto mínimo de conjuntos que cubra todos os elementos do universo.

Formalização:

Seja *U*={1,2,...,n} o universo de **n** elementos e *S*={*S1,S2,...,Sm*} a coleção de **m** conjuntos, onde cada *Si* é um subconjunto de *U*.

A tarefa é encontrar um conjunto C⊆S tal que a união de todos os conjuntos em C seja igual a UU, e CC seja mínimo.

Formulação Matemática:

$$\text{minimizar} \sum_{S \in \mathcal{S}} x_S \qquad \text{(minimizar o número de conjuntos)}$$

$$\text{sujeito a} \sum_{S : e \in S} x_S \geqslant 1 \text{ para todos } e \in \mathcal{U} \qquad \text{(cubra todos os elementos do universo)}$$

$$x_S \in \{0,1\} \text{ para todos } S \in \mathcal{S} \text{ . (cada conjunto está na capa ou não)}$$

Exemplo Ilustrativo:

Considere o universo U={1,2,3,4,5} e a coleção de conjuntos S={{1,2,3},{2,4},{3,4},{4,5}}

Uma solução ótima seria escolher os conjuntos {1,2,3} e {4,5}, pois sua união cobre todos os elementos de *U* com 2 conjuntos. Uma solução de cobertura seria {1,2,3} e {2, 4} e {4, 5}, porém essa solução utiliza 3 conjuntos para cobrir *U* logo não é uma solução ótima.

Complexidade Computacional:

O problema de cobertura de conjuntos é conhecido por ser NP-difícil, o que significa que não existe um algoritmo polinomial conhecido para resolvê-lo em tempo polinomial. No entanto, há algoritmos de aproximação eficientes que podem fornecer soluções próximas à ótima em tempo polinomial. Além disso, o problema de decisão associado (decidir se existe uma cobertura de tamanho **k**) é NP-completo.

Este problema tem amplas aplicações em teoria da computação, teoria dos grafos, otimização combinatória, entre outros campos. É um exemplo clássico de um problema difícil, mas com soluções aproximadas.

Problema específico analisado:

Há de levar em consideração que na descrição acima, o número de colunas selecionadas é o parâmetro de custo utilizado. Na instância utilizada nos algoritmos aqui descritos, cada coluna (conjunto) a ser selecionado possui um custo próprio, logo a tarefa é de completar o conjunto das linhas resultando no menor custo possível, e não no menor número de colunas possíveis.

Com isso, a definição utilizada daqui em diante será a seguinte:

$$\text{Minimizar} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{Sugeito a} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \text{para i=1, 2, ..., m,}$$

Define-se ainda:
Matriz mxn $A = [a_{ij}]$;
$M = \{1, 2, ..., m\}$ o conjunto das linhas;
$N = \{1, 2, ..., n\}$ o conjunto das colunas

onde

c_j =custo da coluna j;

$$a_{ij} = \begin{cases} 1 \text{ se linha } i \text{ é coberta pela coluna } j; \\ 0 \text{ caso contrário} \end{cases}$$

$$x_j = \begin{cases} 1 \text{ se a coluna } j \text{ está na solução;} \\ 0 \text{ caso contrário.} \end{cases} .$$

3. Descrição dos Algoritmos

Na implementação feita, foram utilizados 2 algoritmos heurísticos construtivos, algoritmos esses que a partir de uma entrada, geram uma solução viável, porém sem garantia de ser uma solução ótima, e um algoritmo de melhoramento também heurístico, este diferente dos construtivos, recebe uma solução e, a partir dessa solução, faz manipulações entre a solução e os elementos que não participam da solução e procurar melhorar a solução dada como entrada, novamente sem garantia de resultar em uma solução ótima.

O objetivo deste documento não é descrever o passo a passo da execução do código em baixo nível, com isso em mente, abaixo se encontra uma descrição em alto nível do funcionamento dos algoritmos mencionados:

Construção 1 (`construction1`):

O algoritmo de construção 1 seleciona colunas com base na melhor combinação de custo e cobertura de linhas não cobertas. Ele itera sobre as colunas, avaliando diferentes funções de custo para cada uma e escolhendo a coluna que minimiza a função de custo. A função de custo é escolhida aleatoriamente para cada coluna. O processo continua até que todas as linhas do conjunto solução sejam cobertas.

Construção 2 (`construction2`):

Este algoritmo guloso seleciona colunas simplesmente com base na quantidade de linhas não cobertas que cada coluna pode cobrir, independente do custo da coluna. A ideia é escolher a coluna que cobre a maior quantidade de linhas não cobertas em cada iteração. O processo continua até que todas as linhas do conjunto solução sejam cobertas.

Algoritmo de Melhoria Local (`localSearchAlgorithm`):

O algoritmo de melhoria local é aplicado a uma solução inicial (obtida por um dos algoritmos de construção) para tentar melhorar a solução. Este algoritmo opera removendo aleatoriamente um número D de colunas da solução e substituindo-as por colunas que cobrem as linhas descobertas pela remoção das D colunas no conjunto da solução anterior. O processo é repetido até que todas as linhas sejam cobertas. Em seguida, são removidas colunas que, se removidas, não deixam nenhuma linha não descoberta. Este processo é repetido até que não seja possível remover colunas sem descobrir linhas.

A escolha de quais colunas são inseridas após a remoção das D colunas podem ser feitas de 2 formas na implementação feita, são elas:

Método de seleção Best Improvement

Este método tem como princípio escolher a coluna que, ao ser adicionada ao conjunto solução, irá cobrir o maior número de linhas dentre as colunas candidatas. É necessário ser feita a verificação dos custos de todos os candidatos, o que adiciona tempo de processamento na seleção.

Método de seleção First Improvement

Este método faz a seleção da coluna a ser adicionada à solução sem critérios, logo a primeira coluna a ser candidata a ser adicionada a solução é adicionada diretamente e o algoritmo segue sem verificar as demais candidatas. É um método mais performático em questão de tempo quando comparado ao Best Improvement pois não é necessário verificar mais candidatos

4. Resultados

Com o intuito de verificar as diferenças de performance de cada algoritmo, testes foram executados em 8 arquivos de entrada distintos, onde todos os testes são os mesmos para todos. O fluxo do teste é feito da seguinte forma:

1. Um algoritmo construtivo é executado para gerar uma solução base (o algoritmo escolhido para todos os testes foi o construtivo 1).
2. O algoritmo melhorativo de busca local é executado com a solução inicial dada pelo algoritmo construtivo pelo passo 1.
3. Após a execução do passo 2, o resultado é comparado com um resultado global de execução, e caso o resultado do passo 2 seja melhor que o resultado global, o resultado global recebe o valor do passo 2.
4. Após a execução do passo 3, retorne ao passo 1, com o resultado global atualizado.

No código é possível definir o número de iterações que devem ser feitas do ciclo acima, onde o número de iterações para o teste foram 100 e 10.000 para cada execução do algoritmo melhorativo (Best Improvement e First Improvement) para cada arquivo. As 10.000 iterações são divididas em 2 partes iguais que tem como valor a raiz quadrada da entrada dada, no caso sendo sqrt(10000) = 100. No teste isso significa que o algoritmo construtivo será chamado 100 vezes, e para cada chamada do algoritmo construtivo, o algoritmo de melhoramento será utilizado 100 vezes, resultando em 10.000 iterações no total, e o mesmo é feito para os testes com 100 iterações

Essa abordagem foi utilizada ao invés de apenas executar o algoritmo de melhoramento múltiplas vezes em apenas uma solução inicial pois como o algoritmo de melhoramento retira colunas do fim da lista de solução para efetuar o melhoramento, é possível que valores que estão no início da lista nunca sejam removidos, e esse valores podem não fazer parte de uma solução menos custosa, logo, ao alterar a solução de entrada periodicamente pode evitar com que soluções em ciclos aconteçam.

Para a análise dos resultados, uma das colunas será o GAP dos resultados, o GAP é dado pela função:

GAP = ((Solução Encontrada - Melhor Solução Base) / (Melhor Solução Base)) * 100

O GAP é a diferença entre o resultado encontrado pelo algoritmo e o melhor resultado esperado para aquela entrada em porcentagem, onde valores positivos mostram o quão “Pior” a solução encontrada é em comparação com a melhor solução encontrada até então e valores negativos significa o quão “Melhor” a solução encontrada é em comparação a solução melhor.

Primeiramente, vamos verificar os resultados ao executar o algoritmo construtivo:

Instância	Melhor Solução	Solução construtivo 1	GAP (%) Construtivo 1	Solução construtivo 2	GAP (%) Construtivo 2
test1.dat	557,44	612,64	9,90	616,03	10,51
test2.dat	537,89	629,75	17,08	639,2	18,83
test3.dat	517,58	562,93	8,76	678,82	31,15
test4.dat	1162,8	1274,73	9,63	1364,79	17,37
wren1.dat	7856	10689	36,06	10188	29,68
wren2.dat	13908	15486	11,35	17268	24,16
wren3.dat	13780	15698	13,92	18554	34,64
wren4.dat	58161	66621	14,55	69108	18,82

Tabela 1- Soluções dos algoritmos construtivos

Pela análise da tabela, primeiramente pode-se inferir que o algoritmo construtivo 1 tem um desempenho melhor quando comparado ao construtivo 2, e isso se deve ao fato que mesmo que as colunas encontradas pelo construtivo 2 sejam as colunas que cobrem mais linhas, o fato do peso das colunas não ser necessariamente proporcional ao número de linhas cobertas faz com que colunas mais custosas sejam escolhidas.

Em contrapartida, por conta do construtivo 1 utilizar levar em consideração o custo da coluna e não apenas a quantidade de linhas cobertas faz com que a escolha das colunas seja mais benéfica no aspecto geral.

Isso não significa que o construtivo 2 seja dispensável, pois além dos resultados em algumas instâncias serem próximos, como na entrada test1.dat, o construtivo 2 é executado em um tempo menor por não precisar efetuar os cálculos da função de custo como no construtivo 1, como pode ser analisado na tabela seguinte:

Instância	Tempo de execução do algoritmo construtivo 1 (segundos)	Tempo de execução do algoritmo construtivo 2 (segundos)
test1.dat	0.0002316	6.36e-05
test2.dat	0.0001758	6.38e-05
test3.dat	0.0001778	6.54e-05
test4.dat	0.0001602	6.81e-05
wren1.dat	0.0001916	6.63e-05
wren2.dat	0.0002715	1.225e-04
wren3.dat	0.0002105	8.18e-05
wren4.dat	0.0003542	0.0001686

Tabela 2 - Tempo de execução dos algoritmos construtivos

Apesar de serem diferentes em uma escala que para muitos casos seria um tempo ignorável, em situações onde o tempo é um fator crítico pode fazer com que um algoritmo seja tomado ao invés de outro, além do tempo ser proporcional ao tamanho da entrada, o que torna esse um fator importante para a escolha.

Por conta dos resultados menos custosos do algoritmo de construção 1, ele será usado para a construção das soluções iniciais do algoritmo de melhoramento, os quais os resultados junto com os tempos de execução se encontram a seguir:

Testes com 100 iterações:

Instância	Melhor Solução	Solução First Improvement	GAP (%) First Improvement	Solução Best Improvement	GAP (%) Best Improvement
test1.dat	557,44	597,29	7,1487	597,29	7,1487
test2.dat	537,89	585,96	8,9367	626,7	16,5108
test3.dat	517,58	562,3	8,6402	562,3	8,6402
test4.dat	1162,8	1175,16	1,0629	1161,97	-0,0713
wren1.dat	7856	8715	10,9343	8845	12,5891
wren2.dat	13908	14323	2,9838	14135	1,6321
wren3.dat	13780	14347	4,1146	14448	4,8476
wren4.dat	58161	63469	9,1263	63297	8,8306

Tabela 3 - Soluções dos algoritmos melhorativos com 100 iterações

Arquivo analisado	First Improvement (segundos)	Best Improvement (segundos)
test1.dat	0.1121	0.1122
test2.dat	0.1868	0.1886
test3.dat	0.2643	0.2617
test4.dat	0.4991	0.4972
wren1.dat	1.417	1.3933
wren2.dat	24.5628	20.7284
wren3.dat	28.8033	17.4769
wren4.dat	223.3541	244.7552

Tabela 4 - Tempos de execução dos algoritmos de melhoramento com 100 iterações

Mesmo com 100 iterações, já se percebe uma melhora nos custos finais, tendo até mesmo o encontro de uma solução melhor do que a solução antes definida como a melhor na entrada test4.dat.

Um ponto a ser destacado é o tempo de execução, mesmo que 100 iterações tenham sido utilizadas, ao considerar apenas uma iteração e considerando que cada iteração teve o mesmo tempo de execução (O que não acontece de fato), tem-se que o

tempo de execução do algoritmo de melhoramento é bem mais custoso em questão de tempo quando comparado aos algoritmos construtivos.

Testes com 10.000 iterações:

Instância	Melhor Solução	Solução First Improvement	GAP (%) First Improvement	Solução Best Improvement	GAP (%) Best Improvement
test1.dat	557,44	557,86	0,08	557,86	0,08
test2.dat	537,89	544,6	1,25	543,84	1,11
test3.dat	517,58	515,3	-0,44	515,3	-0,44
test4.dat	1162,8	1157,08	-0,49	1157,08	-0,49
wren1.dat	7856	8142	3,64	8142	3,64
wren2.dat	13908	13826	-0,59	14001	0,67
wren3.dat	13780	13697	-0,60	13847	0,49
wren4.dat	58161	60810	4,55	61526	5,79

Tabela 5 - Soluções dos algoritmos melhorativos com 10.000 iterações

Arquivo analisado	First Improvement (segundos)	Best Improvement (segundos)
test1.dat	9,1916	9,3758
test2.dat	15,2927	15,4789
test3.dat	21,4714	27,4674
test4.dat	49,8571	43,6393
wren1.dat	124,8112	123,5832
wren2.dat	1645,7145	1646,0408
wren3.dat	1394,9527	1397,9346
wren4.dat	12927,6116	13085,8281

Tabela 6 - Tempos de execução dos algoritmos de melhoramento com 10.000 iterações

Um ponto a ser destacado é que algumas soluções acabaram sendo melhores do que as soluções previamente tidas como as melhores, no caso da First Improvement, temos nas entradas test3.dat, test4.dat, wren2.dat e wren3.dat, enquanto para a Best Improvement temos as test3.dat e test4.dat.

Partindo para a análise dos custos, percebe-se que ambos os algoritmos chegaram a um resultado pelo menos próximo à melhor solução, sendo o maior GAP encontrado em wren4.dat pelo método First Improvement. Apesar dos resultados serem próximos, no aspecto geral o método First Improvement se sair melhor em relação ao Best Improvement, e isso se deve ao fato que apesar do Best Improvement escolher as colunas que cobrem o maior número de linhas, os custos não são proporcionais e logo, não necessariamente as colunas que cobrem o maior número de linhas são as colunas que fazem parte da solução menos custosa.

Analisando agora a tabela de tempo de execução, o First novamente toma vantagem em relação ao Best, pois no método Best se faz necessário a verificação de todas as colunas candidatas a serem adicionadas a solução em cada iteração, diferentemente do First, que escolhe a primeira coluna candidata e em frente com a execução, tornando-a mais performática em quesito de tempo.

5. Conclusão

Algumas conclusões específicas já se encontram no tópico de resultados, aqui se encontram as considerações gerais em torno dos resultados obtidos.

Algoritmos Construtivos

Apesar dos resultados serem aproximados entre si, há uma clara vantagem ao se utilizar o método construtivo 1. Nas instâncias utilizadas, a escolha das colunas levando em consideração o custo das colunas faz com que as escolhas sejam menos imprecisas, além de que outras funções de cálculo de custo podem ser utilizadas que não sejam cj / kj (cj -> custo da coluna j e
kj -> número de linhas cobertas pela coluna j). O algoritmo de construção 2, entretanto, possui uma implementação mais simples além de ser mais rápido que o algoritmo 1.

Algoritmos melhorativos

Começando com 100 iterações já se demonstra um resultado aproximado em algumas entradas, enquanto outras há um desvio de mais de 15%, que a depender dos valores de custo das colunas é uma grande diferença no custo final, porém possuem tempos razoáveis de execução, sendo um número de iterações que interessa caso a precisão do custo final possa passar uma margem de erro de até cerca de 20%. A forma de escolher as colunas com o Best Improvement se tornou mais vantajosa em alguns casos, encontrando até mesmo resultados melhores dos que colocados como parâmetro para análise, porém no quesito geral o método First Improvement ainda toma vantagem.

Em vista do método melhorativo com 10.000 iterações, é evidente que os custos são melhores quando comparados com 100 iterações independentemente da entrada ou algoritmo de escolha de coluna utilizado, pois mais possibilidades de solução são analisadas e, consequentemente, maiores são as chances de se encontrar uma solução melhor. Em contrapartida, o tempo de execução escala muito com o aumento do número de

iterações, tendo no caso da entrada wren4.dat com o algoritmo de Best Improvement um aumento de cerca de 60 vezes do tempo de execução, o que é de se esperar visto que o número de iterações teve um aumento em 100 vezes.

Analisando as funções de escolha de colunas, o First no formato das entradas se saiu com um desempenho melhor, seja em tempo ou em custos finais. Entretanto isso não é um aviso para se deixar de utilizar o método de Best Improvement para outros casos, pois o desempenho de cada algoritmo varia de uma série de fatores, alguns como tamanho da entrada, relação entre os custos e o número de linhas cobertas, densidade da entrada entre outros.

Aleatoriedade dos algoritmos melhorativos

Na implementação feita, o número de colunas D removidas da solução para serem substituídas é randômico, o que em um aspecto é vantajoso pois aumenta o número de possibilidades de solução, porém tem a desvantagem dos resultados não serem previsíveis, já que não se sabe o número de colunas que serão substituídas. Isso significa que o código utilizado, mesmo com os parâmetros de entrada iguais, as saídas podem diferir, o que significa que os resultados avaliados neste documento podem diferir de execução para execução, por isso um grande número de iterações como 10.000 foi escolhido, pois cobre uma grande quantidade de soluções.

Aspecto geral

Dentre os dados apresentados e o ponto levantado da aleatoriedade, pode-se dizer que mesmo com custos ainda a serem melhorados, tem-se que os algoritmos servem o seu propósito e podem ser melhorados para a obtenção de melhores resultados, como no aumento no número de iterações, funções de cálculo de custo para o algoritmo construtivo, métodos distintos do First Improvement e Best Improvement para seleção de colunas, utilização de estruturas de dados mais performáticas, entre outros.

O propósito do documento não é fornecer uma solução final e ótima do problema de cobertura de conjuntos, ainda mais por se tratarem de algoritmos heurísticos, mas fornece uma base para a construção de algoritmos mais eficientes e levanta o ponto de que resultados anteriormente encontrados podem ser melhorados a depender do problema, como demonstrado nos GAPs negativos encontrados na análise das soluções.

Referências

- Jacobs, L. W. and Brusco, M. J. A Local-Search Heuristic for Large Set-Covering Problems. Naval Research Logistic.
- PROBLEMA DE COBERTURA DE CONJUNTOS. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2022. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Problema_de_cobertura_de_conjuntos&oldid=64270749>. Acesso em: 24 ago. 2022.