Relatório sobre o Algoritmo Genético para o Problema de Cobertura de Conjuntos(SCP).

Departamento de Informática - UEM - Maringá - PR - Brasil
Modelagem e Otimização Algorítmica - 6903/01
Marcos Vinicius de Oliveira - Ra:124408,
Guilherme Frare Clemente - Ra:124349.

● Introdução

O *Set Covering Problem*(Problema de Cobertura de Conjuntos) é um problema clássico de otimização combinatória, com diversas aplicações em logística, design de redes, alocação de recursos, entre outros. O objetivo é selecionar o menor número possível de conjuntos de como que a união desses conjuntos cubra todos os elementos de um conjunto universo.

Neste relatório, será apresentado um algoritmo genético que será explicado detalhadamente conforme o relatório é avançado, combinado com uma busca local já implementada na linguagem de programação Python para o desenvolvimento do problema, além de que o algoritmo utiliza uma abordagem construtiva gulosa (funções de custo) que serão explicadas também. Além disso, também será feito uma análise dos resultados obtidos com a execução do algoritmo genético quando comparado com as soluções obtidas pela literatura, para auxiliar no funcionamento do algoritmo.

● 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 $C \subseteq S$ tal que a união de todos os conjuntos em C seja igual a U, e C seja mínimo.

Formulação Matemática:

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

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

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

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:

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

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

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;

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

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

- # Descrição do Algoritmo

Algoritmo Genético com busca local

O algoritmo genético proposto para a resolução do *Problema de Cobertura de Conjunto* utiliza as seguintes estruturas:

- **Seleção das soluções para cruzamento:** Utiliza o método da roleta viciada, onde a probabilidade de seleção de cada solução é proporcional à sua aptidão. Isso significa que soluções com maior custo terão uma probabilidade maior de serem selecionadas para reprodução.
- **Cruzamento das soluções:** É realizado o cruzamento de dois pais selecionados, utilizando o método de crossover de um ponto. Um ponto de corte é escolhido aleatoriamente e as partes das soluções dos pais antes e depois desse ponto são trocadas, criando dois novos filhos.
- **Mutação:** pós o crossover, é aplicada a mutação por inversão. Nesse método, são escolhidos aleatoriamente dois pontos de corte e a ordem dos genes entre esses pontos é invertida em um dos filhos. Isso introduz uma diversificação na população, permitindo explorar novas regiões do espaço de busca.
- **Procedimento de busca local:** Após a etapa de mutação, é realizado um procedimento de busca local para melhorar as soluções individuais. Esse procedimento busca otimizar a solução localmente, ajustando a configuração das colunas selecionadas para melhor cobrir as linhas do problema.
- **Critério de Parada:** O critério de parada é definido pelo número máximo de gerações especificado como parâmetro. Uma vez atingido esse número, o algoritmo encerra sua execução e retorna a melhor solução encontrada até o momento.

Em resumo, o algoritmo genético com busca local combina a exploração global do espaço de busca através da seleção, crossover e mutação com a exploração local através da busca local. Isso permite encontrar soluções de alta qualidade para o problema do Set Covering, buscando um equilíbrio entre diversificação e intensificação na busca.

Algoritmo Construtivo

O algoritmo abaixo mostra a construção gulosa para resolução do problema, onde dentro desse algoritmo possui uma classe **Coluna** que representa: índice da coluna, custo da coluna, linhas que essa coluna cobre. E também a classe **Dados** que representa: número de linhas, número de colunas, e uma lista de **Coluna.**

- **Algoritmo Construtivo:** Definido no algoritmo como(**construtivo()**)

    - Inicializa uma solução vazia e um conjunto R contendo todas as linhas do universo inicialmente não cobertas.
    - Interativamente, seleciona a coluna que tem o menor custo relativo ao número de linhas que ainda não estão cobertas por outras colunas já selecionadas na solução. Isso é feito por meio de uma função de custo gulosa, que é escolhida de forma aleatória no algoritmo (7 funções foram fornecidas).
    - Adiciona a coluna selecionada à solução e atualiza o conjunto R removendo as linhas cobertas por essa coluna.
    - Após a seleção das colunas, o algoritmo remove colunas redundantes da solução. Colunas redundantes são aquelas que não são necessárias para cobrir todas as linhas, pois outras colunas já selecionadas as cobrem.
    - Verifica se a solução obtida é válida, ou seja, se todas as linhas são cobertas pela união dos subconjuntos selecionados

Este algoritmo oferece uma solução inicial para o problema, mas pode não ser ótimo. Em uma parte dos passos é indicado 7 funções, abaixo é mostrado como é representado essas funções:

***kj* = número de linhas (ainda não cobertas) que podem ser cobertas com a coluna j;**

cj = custo da coluna j;

Funções:

1- cj

2- $cj \ / \ kj$

3- $cj \ / \ log_2 kj$

4- $cj \ / kj \ log_2 kj$

5- $cj \ / \ kj \ ln \ kj$

6- $cj \ / \ kj^2$

7- $cj^{1/2} / \ kj^2$

Onde a função que obteve um melhor desempenho foi a função 2, obteve um melhor custo em relação às outras funções.

Algoritmo de busca local

Algoritmo de melhoramento *(Busca local)* proposto por *Jacob & Brusco:* Relembrando que neste algoritmo foi utilizado a técnica de ***Best Improvement***, ou seja, ele continua procurando colunas que cobrem mais linhas, até encontrar a melhor solução possível. Ele tenta remover colunas da solução e adicionar outras, mantendo a melhor solução encontrada até o momento.

- Após a aplicação de um dos algoritmos construtivos, é realizada uma etapa de aprimoramento da solução encontrada.
- O algoritmo de busca local busca otimizar a solução inicial encontrada, removendo e adicionando colunas de forma a reduzir o custo total da solução e garantir a cobertura de todas as linhas.
- O processo de melhoramento é iterativo e utiliza parâmetros aleatórios para determinar as operações a serem realizadas, visando explorar diferentes configurações de solução.
- O algoritmo possui os seguintes passos:
  - Determinação de parâmetros aleatórios D e E que definem a quantidade de colunas a serem removidas da solução inicial e o limite de custo para a adição de novas colunas.
  - Remoção de um número aleatório de colunas da solução inicial até que o número de colunas seja igual a D.
  - Identificação das linhas não cobertas pela solução atual e seleção de novas colunas que podem ser adicionadas para cobrir essas linhas.
  - Adição de colunas que cobrem as linhas não cobertas e que têm custo inferior ou igual a E.
  - Verificação e possível remoção de colunas redundantes da solução final.
- O processo de melhoramento continua até que não seja mais possível melhorar a solução ou até atingir um número máximo de iterações definido pelo usuário.

Sua natureza iterativa e a utilização de parâmetros aleatórios permitem explorar diferentes configurações de solução e encontrar soluções potencialmente melhores em relação à solução inicial.

- ## Resultados

Para a análise dos resultados, o algoritmo foi testado para 8 entradas distintas: teste1.dat (matriz 50x300), teste2.dat(matriz 50x500), teste3.dat(matriz 50x700), teste4.dat(matriz 100x500), wren1.dat(matriz 200x539), wren2.dat(matriz 222x5522), wren3.dat(matriz 219x4990) e wren4.dat(matriz 798x4569). O código foi executado para dois casos:

- 100 de população, 100 de gerações, probabilidade de mutação de 0.9 e probabilidade de elitismo 0.9;
- 100 de população, 500 de gerações, probabilidade de mutação de 0.9 e probabilidade de elitismo 0.9.

As colunas obtidas para os respectivos custos foram(Para o primeiro caso):

Teste_01.dat: [10, 278, 44, 185, 129, 133, 242, 77, 58, 109]

Teste_02.dat: [152, 253, 471, 488, 296, 458, 302, 110, 370]

Teste_03.dat: [569, 190, 341, 10, 402, 538, 39, 4, 421]

Teste_04.dat: [194, 72, 244, 56, 296, 197, 176, 253, 461, 211, 41, 326, 359, 226, 165, 484, 468, 44, 433]

Wren_01.dat: [488, 146, 250, 75, 240, 249, 303, 338, 208, 1, 241, 126, 345, 2, 324, 323]

Wren_02.dat: [1540, 5363, 5071, 5460, 5409, 5399, 5081, 2770, 2399, 5477, 5408, 4979, 5426, 4990, 826, 2791, 5147, 5482, 5074, 5427, 5522, 5481, 5459, 5503, 5040, 5050, 5484, 5389, 2999, 436, 5516, 5502, 5097]

Wren_03.dat: [4380, 2245, 1164, 1267, 4869, 4634, 4210, 3981, 4964, 4969, 4790, 4901, 4862, 4881, 4629, 80, 4839, 4937, 4936, 4551, 4686, 4695, 1959, 4503, 4654, 4919, 646, 4882, 2453, 1687, 4900, 4575]

Wren_04.dat: [1252, 1007, 2309, 4515, 1022, 2924, 78, 3261, 3111, 1167, 2862, 2439, 2646, 1084, 1098, 2390, 1895, 516, 1882, 230, 1711, 686, 3883, 2716, 1413, 4255, 2518, 825, 2760, 2741, 1073, 51, 3365, 1632, 2483, 2631, 1780, 3458, 3356, 4416, 2706, 3249, 2679, 973, 533, 2734, 3858, 3979, 2558, 1788, 1324, 756, 132, 90, 735, 647, 3876, 763, 2966, 426, 182, 2437, 248, 2246, 4374, 939, 1802, 558, 2815, 2057, 1773, 915, 2765, 1732, 2787, 2047, 2731, 1475, 1572, 2802, 4016, 310, 3122, 289, 3719, 1936, 142, 3482, 4193, 1272, 1228, 2775, 1377, 1168, 1393, 133, 1857,

599, 172, 2077, 715, 937, 2624, 616, 4377, 3385, 1815, 2883, 1300, 140, 1847, 2627, 2575, 263, 2893, 211, 3934, 654, 631, 2766, 4488, 4444, 3027, 282]

Abaixo é mostrado o custo da melhor solução encontrada para cada entrada para o primeiro caso(com busca local), juntamente com o tempo de execução de cada entrada.

Arquivo de Entrada	Custo da Melhor Solução	Tempo de Execução (segundos)
teste_01.dat	594.76	8.5166
teste_02.dat	544.6	12.3048
teste_03.dat	517.52	17.2175
teste_04.dat	1157.08	24.2668
wren_01.dat	8142.0	41.7902
wren_02.dat	13866.0	532.7040
wren_03.dat	13749.0	477.5441
wren_04.dat	61286.0	1882.6180

As colunas obtidas para os respectivos custos foram(Para o segundo caso):

Teste_01.dat: [194, 190, 222, 58, 179, 103, 10, 22, 99]

Teste_02.dat: [471, 189, 434, 386, 325, 62, 61, 251, 7]

Teste_03.dat: [4, 341, 402, 663, 569, 127, 39, 538, 10]

Teste_04.dat: [194, 211, 468, 359, 326, 253, 197, 72, 56, 461, 433, 296, 226, 44, 484, 176, 165, 244, 41]

Wren_01.dat: [303, 250, 252, 75, 208, 2, 345, 146, 324, 1, 249, 488, 323, 240, 338, 126]

Wren_02.dat: [5050, 2095, 4948, 5147, 2399, 5408, 5522, 5328, 5074, 5484, 5399, 5409, 961, 4979, 5460, 5449, 5481, 5477, 5040, 5142, 5482, 1164, 1540, 5363, 5071, 5427, 5459, 5389, 2999, 4990, 2779, 2068]

Wren_03.dat: [4634, 4919, 4937, 4568, 1688, 4764, 4869, 2556, 1164, 4839, 2294, 644, 386, 4881, 4920, 4969, 4551, 4650, 4862, 4965, 4696, 4882, 3990, 4990, 4781, 4964, 4936, 2224, 2739, 4210, 4104, 4689]

Wren_04.dat: [2735, 2728, 1869, 1097, 4407, 1895, 3423, 2626, 278, 248, 142, 2966, 1736, 4017, 2883, 513, 763, 3121, 647, 1080, 992, 2087, 1022, 1379, 2862, 2385, 4095, 3367, 310, 1721, 218, 1475, 1780, 607, 93, 1539, 559, 2331, 3835, 2816, 34, 930, 2483, 1167, 2678, 1231, 759, 1435, 2757, 4510, 839, 2801, 2441, 2625, 3111, 2736, 525, 3718, 3358, 1275, 3331, 2663, 133, 599, 4250, 1400, 3809, 1802, 2598, 903, 1599, 2802, 2763, 182, 3264, 939, 753, 2499, 209, 685, 4536, 885, 2000, 1169, 1499, 330, 915, 441, 713, 4374, 3494, 166, 2794, 2631, 2428, 967, 737, 533, 3026, 1248, 193, 1293, 2568, 1083, 1815, 2924, 3389, 3456, 3273, 2538, 4439, 668, 265, 2246, 2430, 152, 1364, 4444, 78, 1152]

Abaixo é mostrado o custo da melhor solução encontrada para cada entrada para o segundo caso(com busca local), juntamente com o tempo de execução de cada entrada.

Arquivo de Entrada	Custo da Melhor Solução	Tempo de Execução
teste_01.dat	557.8599999999999	37.2424932
teste_02.dat	537.8899999999999	61.0320063
teste_03.dat	517.5799999999999	89.1179799
teste_04.dat	1157.0799999999995	108.6609417
wren_01.dat	8142.0	211.0229094
wren_02.dat	13678.0	2527.1519349
wren_03.dat	13710.0	2135.178894
wren_04.dat	58755.0	12448.6128057

Abaixo é mostrado uma tabela de comparação dos resultados obtidos para uma população e gerações com valor 100:

Caso	MS	Alg	GAP%	Alg_bl	GAP%
Teste_01	557.44	656.82	17.82%	594.76	6.69%
Teste_02	537.89	610.68	13.53%	544.6	1.24%
Teste_03	517.58	590.31	14.05%	517.52	-0.01%
Teste_04	1162.08	1320.71	13.65%	1157.08	-0.43%
Wren_01	7856	9032.0	14.96%	8142.0	3.6%
Wren_02	13908	15820.0	13.74%	13866.0	-0.3%
Wren_03	13780	16471.0	19.52%	13749.0	-0.22%
Wren_04	58161	65866.0	13.24%	61286.0	5.37%

A partir desses dados fornecidos acima, conseguimos observar como a aplicação da busca local no algoritmo genético influencia no resultado final obtido pelo custo, diminuindo assim os Gaps de cada arquivo de entrada, ou seja, a busca local tem um impacto relativamente grande no algoritmo genético do SCP.

Abaixo é mostrado uma tabela de comparação dos resultados obtidos para uma população de 100 e gerações de 500:

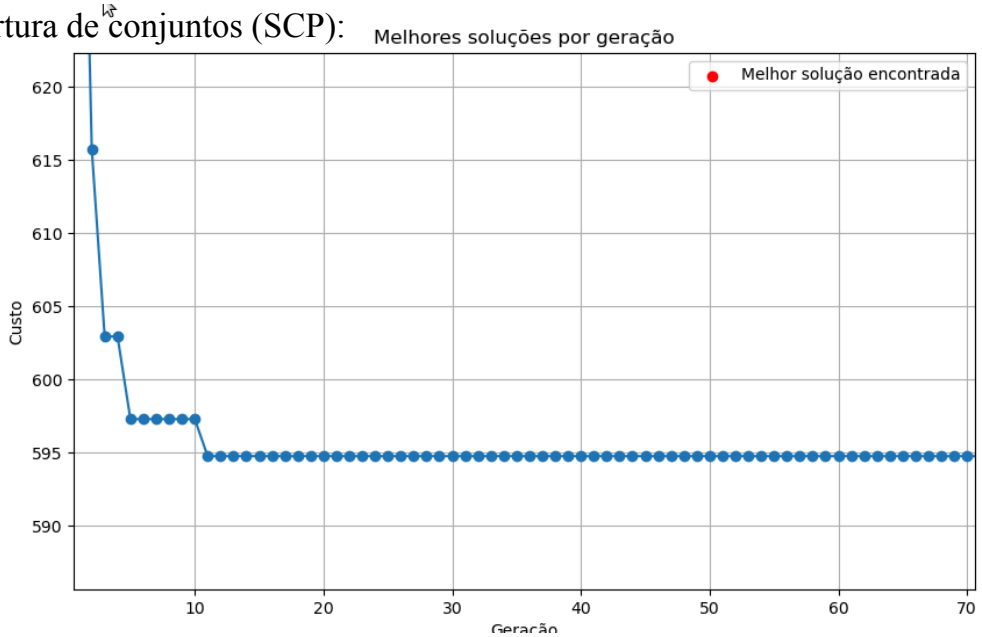
Caso	MS	Alg	GAP%	Alg_bl	GAP%
Teste_01	557.44	600.83	7.78%	557.86	0.07%
Teste_02	537.89	635.25	18.10%	537.89	0.0%
Teste_03	517.58	579.96	12.05%	517.58	0.0%
Teste_04	1162.08	1286.51	10.70%	1157.08	-0.43%
Wren_01	7856	9465.0	20.48%	8142.0	3.64%
Wren_02	13908	16126.0	15.94%	13678.0	-1.65%
Wren_03	13780	16420.0	19.15%	13710.0	-0.50%
Wren_04	58161	64919.0	11.61%	58755.0	1.02%

Da mesma forma apresentada na tabela anterior, o impacto da busca local no algoritmo genético SCP influencia muito para o resultado final obtido, ou seja, o custo das soluções obtidas. Conseguimos observar também que à medida que aumentamos o número de gerações, o algoritmo conseguiu atingir um custo melhor sem a busca local e também com a busca local aplicada, quando comparada com a primeira tabela ou o número de gerações era 100.

Para uma próxima análise, será mostrado abaixo um gráfico do arquivo de entrada Teste_01.dat que representa o desvio padrão da população com base na sua aptidão, ou seja, com base no seu custo. Só foi apresentado para o Teste_01.dat pois para todos as outras entradas, o gráfico obteve a mesma resposta, pois, no momento que o algoritmo não consegue melhorar mais, ele continua dando a mesma solução, ou seja, uma reta constante, e por conta disso, a maioria das entradas ficaram com um aspecto parecido com o gráfico abaixo:

Por fim, é mostrado abaixo um gráfico do custo obtido pela entrada Teste_03.dat, em relação ao número de gerações, para conseguir ter um entendimento melhor sobre a busca local. Pode-se observar que, com a aplicação da busca local para cada entrada, os valores dos custos tendem a diminuir de uma forma compreensível, chegando a custos bem próximos dos fornecidos pela literatura. Observando isso, é

possível concluir que a aplicação da busca local modifica a solução de uma maneira boa para alcançar resultados melhores e válidos para todas as entradas quando comparadas sem a aplicação da busca local no algoritmo genético para o problema de cobertura de conjuntos (SCP):



Com base no gráfico acima, conseguimos observar que conforme for avançando o número de gerações, o custo da solução(aptidão/fitness) começa a melhores até alcançar próximo a solução ótima obtida pelo algoritmo genético implementado, e isso conclui que conforme o número de gerações aumenta, a aptidão vai se aproximando da ótima até um valor que não é possível mais melhorar para aquela entrada em relação ao algoritmo genético.

- ## Conclusões

O presente estudo abordou a aplicação de um algoritmo genético combinado com busca local para resolver o Problema de Cobertura de Conjuntos(SCP). O SCP é um problema complexo com ambas aplicações em diversas áreas, desde logística até otimização combinatória.

Os resultados obtidos demonstraram que a combinação do algoritmo genético com a busca local resultou em soluções de alta qualidade para uma variedade de entradas do problema. Observou-se uma redução significativa nos custos das soluções encontradas em comparação com os resultados iniciais, evidenciando o impacto positivo da busca local na melhoria das soluções geradas pelo algoritmo genético.

Além disso, foi possível observar que o aumento no número de gerações contribuiu para a melhoria das soluções, tanto com quanto sem a aplicação da busca local. No entanto, a busca local desempenhou um papel crucial na obtenção de soluções mais próximas dos resultados ótimos conhecidos na literatura, especialmente para casos complexos.

A análise dos resultados também revelou que a escolha adequada de parâmetros, como o número de gerações e a taxa de mutação, pode influenciar significativamente o desempenho do algoritmo genético com busca local.

Portanto, conclui-se que a abordagem proposta neste estudo é eficaz e promissora para resolver o SCP, oferecendo soluções de alta qualidade em um tempo razoável. Futuros trabalhos podem explorar outras técnicas de otimização e ajustar os parâmetros do algoritmo para obter resultados ainda melhores em diferentes instâncias do problema.

Referências

Glover, F., & Kochenberger, G. A. (2006). Handbook of Metaheuristics. Springer Science & Business Media.

Dorigo, M., & Stützle, T. (2004). Ant colony optimization. MIT press.

Johnson, D. S. (1973). Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences, 9(3), 256-278.

Beasley, J. E. (1990). OR-library: Distributing test problems by electronic mail. Journal of the Operational Research Society, 41(11), 1069-1072.

Brusco, M. J., & Jacobs Jr, H. J. (2003). A hybrid genetic algorithm for the set covering problem. INFORMS Journal on Computing, 15(4), 347-362.

Johnson, D. S. (1974). Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences, 9(3), 256-278.

Hochbaum, D. S., & Pathria, A. (1996). Approximation algorithms for the set covering and vertex cover problems. SIAM Journal on Computing, 25(6), 1333-1357.

Stinson, D. R. (2004). Combinatorial designs: constructions and analysis. Springer Science & Business Media.