Universidade Estadual de Maringá
Centro de Tecnologia - Departamento de Informática
Ciência da Computação
6903 - Modelagem e Otimização Algoritmica
Professor Ademir Constantino

Terceira Avaliação

Aluno: Rafael Cortez Sanches
RA: 82357

Maringá, 29 de fevereiro de 2016

Conteúdo

1 Resumo

O objetivo desse trabalho é estudar a aplicação de algoritmos genéticos (meméticos) para a resolução do problema de cobertura de conjuntos com minimização de custo, um problema NP-Difícil conhecido na literatura.

Primeiramente, é introduzido o conceito da meta-heurística dos algoritmos genéticos, caracterizando sua estrutura geral. Em seguida, apresenta-se uma descrição do problema de cobertura de conjuntos (PCC), assim como a representação de suas soluções, da forma como foram tratadas nesse trabalho.

Também foi desenvolvido um algoritmo genético para solucionar instâncias do PCC, as quais foram fornecidas pelo professor. O algoritmo apresentado no trabalho foi implementado em C++, e teve seus parâmetros calibrados em seguida, por uma série de testes.

Uma vez calibrados os parâmetros, o programa os utilizou para solucionar 8 casos de teste diferentes, produzindo boas soluções através da meta-heurística.

2 Introdução

Algoritmos genéticos são meta-heurísticas que mimetizam genética e teoria da evolução. Eles geram um conjunto de soluções para um dado problema, o qual é chamado de população inicial. A partir dessa, seus indivíduos são combinados por cruzamentos e sofrem mutações, gerando novos indivíduos (soluções).

Aqueles que estiverem mais aptos (indivíduos que representam soluções melhores) têm maiores chances de serem selecionados para os cruzamentos, contribuindo para a geração de gerações aprimoradas. Entretanto, o algoritmo também cuida para não eliminar todas as soluções ruins, uma vez que elas podem contribuir com a variabilidade genética da população.

Um algoritmo genético pode ser dividido nas seguintes etapas:

1. **Geração da população inicial:** Soluções aleatórias para o problema são criadas e armazenadas. Geralmente, o tamanho dessa população se mantém constante ao longo do algoritmo.

2. **Escolha de um par para o cruzamento:** Duas soluções são selecionadas de acordo com uma função *fitness*. Soluções melhores têm maior chance de serem selecionadas em relação às piores soluções contidas na população.

3. **Crossover ou cruzamento:** As soluções escolhidas pela etapa anterior são combinadas em uma única solução. Caso essa apresente alguma redundância ou invalidade, uma função deve ser aplicada a ela para transformá-la em uma solução válida.

4. **Mutação:** A nova solução gerada pode ou não sofrer uma mutação. Essa ocorrência se apresenta de acordo com uma taxa de mutação pré-estabelecida. Caso sofra a mutação, a solução é sujeita a uma função que a modifica, dando a ela novas características além daquelas herdadas de seus pais.

A primeira etapa é executada somente uma vez, no início do algoritmo. As etapas seguintes são repetidas até que um certo critério de parada seja satisfeito. A finalização do processo geralmente se dá por um dos seguintes critérios:

- **Qualidade mínima da solução:** O algoritmo só encerra depois que encontrar uma solução com qualidade maior ou igual a um determinado valor fornecido pelo usuário.

- **Número máximo de gerações:** Cada etapa do algoritmo em que novas soluções são geradas caracteriza uma geração. Nesse critério de parada, o algoritmo cessa quando um número de gerações é alcançado.

- **Não aprimoramento da melhor solução:** O algoritmo para quando não se tem uma melhora da melhor solução após um certo número de gerações consecutivas.

- **Esgotamento de recursos:** Envolve limites de recursos disponíveis ao usuário, o que geralmente remete ao tempo de execução. Quando o recurso se esgota, o algoritmo cessa, retornando a melhor solução encontrada.

- **Parar manualmente:** O operador cessa a execução através de um comando, ficando com a melhor solução encontrada durante a execução.

Esses critérios podem ser combinados para compor outras condições de parada, por exemplo: tentar buscar uma solução de qualidade mínima, mas com tempo máximo de execução definido.

O algoritmo pode conseguir uma boa solução em tempo hábil e de acordo com os critérios fornecidos, mas vale lembrar que a solução encontrada por essa heurística não é necessariamente ótima.

3 O Problema de Cobertura de Conjuntos (PCC)

O problema de cobertura de conjuntos pode ser resumido da seguinte forma:

Tem-se uma série de k conjuntos A_j com elementos a_i distribuídos entre eles, tal que $\bigcup_{j=1}^{k} A_j = U$, onde U é o conjunto universo. Note que um mesmo elemento pode estar em mais de um conjunto A_j ao mesmo tempo. O PCC consiste em determinar qual o menor número de conjuntos entre a coleção A_j que cobre todos os elementos presentes no universo do problema.

Nesse trabalho, o PCC também considera um custo c_j associado a cada um dos conjuntos, de forma que a escolha de um conjunto A_j acresce em c_j o custo total da solução. O problema de otimização envolve a minimização desse custo.

3.1 Representação do Problema

A representação do problema pode ser apresentada como um modelo de programação linear (PL). Para representá-lo dessa forma, representamos os elementos a_i nas linhas i e os conjuntos A_j na colunas j. Além disso, estabelecem-se as seguintes variáveis:

- x_j: A presença da coluna j na solução. Pode assumir os valores 1 caso a coluna esteja presente, 0 caso contrário.
- y_{ij}: Se o elemento a_i está presente no conjunto A_j, essa variável assume valor 1. Caso contrário, assume valor 0.
- c_j: O custo associado a coluna j.

O problema consiste em minimizar o custo total:

$\sum_{j=1}^{k} c_j x_j$

Sujeito à restrição:

$\sum_{j=1}^{k} y_{ij} x_j \geq 1$ para toda linha i

3.2 Estrutura da Solução

Nesse trabalho, as soluções do PCC foram representadas por um tipo abstrato de dados contendo duas estruturas básicas de representação:

- Um vetor de vetores contendo todas as colunas do problema. O vetor mais externo armazena as colunas da solução, que são os vetores internos. Esses armazenam as linhas que aquela coluna cobre. Colunas que não fazem parte da solução têm um vetor de linhas de tamanho zero.

- Um vetor de vetores contendo todas as linhas do problema. O vetor mais externo armazena as linhas da solução, que são vetores de colunas. Esses armazenam as colunas que cobrem a linha em questão. Como o problema exige que todas as linhas sejam cobertas, os vetores internos nunca estarão vazios (restrição do modelo de PL).

Além das linhas e colunas, a solução também guarda a soma dos custos das colunas presentes nela. Ao atributo representado por essa soma é dado o nome de **aptidão**. Quanto menor for seu valor, melhor a solução.

Um exemplo de solução é apresentado a seguir, para um problema de 300 colunas e 50 linhas. As colunas não presentes na solução foram ocultadas para diminuir a extensão do exemplo.

```
Colunas presentes:
        Coluna 10: 7 9 11 27 41 42 49
        Coluna 44: 4 16 26 30 34 50
        Coluna 58: 12 16 21 23 34 39
        Coluna 77: 1 19 27 33 44
        Coluna 109: 13 14 31 40
        Coluna 129: 3 13 22 29 37 43 48
        Coluna 133: 5 15 20 24 28
        Coluna 185: 10 18 38 46 47 49
        Coluna 242: 2 6 14 25 35 38
        Coluna 278: 8 14 17 21 32 36 45
Cobertura de linhas:
        Linha 1: 77
        Linha 2: 242
        Linha 3: 129
        Linha 4: 44
        Linha 5: 133
        Linha 6: 242
        Linha 7: 10
        Linha 8: 278
        Linha 9: 10
        Linha 10: 185
        Linha 11: 10
        Linha 12: 58
        Linha 13: 109 129
        Linha 14: 109 242 278
        Linha 15: 133
        Linha 16: 44 58
        Linha 17: 278
        Linha 18: 185
        Linha 19: 77
        Linha 20: 133
        Linha 21: 58 278
        Linha 22: 129
        Linha 23: 58
        Linha 24: 133
        Linha 25: 242
        Linha 26: 44
        Linha 27: 10 77
        Linha 28: 133
        Linha 29: 129
        Linha 30: 44
        Linha 31: 109
        Linha 32: 278
        Linha 33: 77
        Linha 34: 44 58
        Linha 35: 242
        Linha 36: 278
        Linha 37: 129
        Linha 38: 185 242
        Linha 39: 58
        Linha 40: 109
        Linha 41: 10
        Linha 42: 10
        Linha 43: 129
        Linha 44: 77
        Linha 45: 278
        Linha 46: 185
        Linha 47: 185
        Linha 48: 129
        Linha 49: 10 185
        Linha 50: 44
```

4 O Algoritmo Genético Desenvolvido

Essa seção descreve os algoritmos e procedimentos adotados na solução do PCC pela meta-heurística do algoritmo genético. Os pseudocódigos são apresentados em caixas com linhas enumeradas para melhor visualização.

O corpo principal do algoritmo é apresentado a seguir:

```
1 Populacao = Gerar_Populacao_Inicial()
2 Ngeracoes = 0
3 Enquanto Ngeracoes < MAX_GERACOES
4         ++Ngeracoes
5         Gerar_Novo_Descendente(Populacao)
6 Retorna melhor solucao presente em Populacao
```

Onde MAX_GERACOES é o número máximo de gerações que o algoritmo percorre até parar.

4.1 Gerar População Inicial

A população inicial é gerada a partir de uma seleção aleatória de linhas e uma escolha gulosa para as colunas que podem cobrir a linha selecionada. Esse procedimento é descrito abaixo:

```
1  Gerar_Solucao_Inicial(P)
2  {
3          Linhas_Nao_Cobertas = P.Linhas
4          Enquanto Linhas_Nao_Cobertas != Vazio
5                  Nova_Linha = Escolha_Aleatoria(Linhas_Nao_Cobertas)
6                  Min = +Infinito
7                  for Coluna Col in P.Colunas
8                          if Nova_Linha esta em Col e
9                          (Col.Custo / Col.Tamanho) < Min
10                                 Min = Col.Custo / Col.Tamanho
11                                 Col_Selec = Col
12                 Solucao.Adicionar(Col_Selec)
13                 for Linha Lin in Col_Selec
14                         Linhas_Nao_Cobertas =
15                         Linhas_Nao_Cobertas - Lin
16         retorna Solucao
17 }
```

Em que P é o problema, contendo as linhas P.Linhas e as colunas P.Colunas. O procedimento acima é repetido até que se preencha toda a população inicial.

4.2 Gerar Novo Descendente

O algoritmo para gerar soluções descendentes seleciona duas soluções para o cruzamento, aplica-o às duas, aplica a mutação e retira uma solução aleatória da população, para que essa permaneça com o mesmo tamanho. O procedimento é detalhado a seguir:

```
1  Gerar_Novo_Descendente(Populacao)
2          Casal = Escolher_Casal(Populacao)
3          Novo = Cruzamento(Casal)
4          Mutacao(Novo)
5          Escolhe Aleatoriamente Sol em Populacao,
6          tal que Sol nao eh a melhor solucao
7          Populacao = Populacao - Sol
8          Populacao.Inserir(Novo)
```

As funções envolvidas nesse algoritmo são apresentadas nas subseções seguintes.

4.3 Função Escolher Casal (Fitness)

Essa função elege aleatoriamente um par de soluções para serem combinadas. A seleção é feita de forma que as soluções de melhor aptidão possuam maior chance de serem escolhidas para o cruzamento. O algoritmo de escolha é apresentado abaixo:

```
1  Escolhe_Casal(Populacao)
2          Populacao.Sort()
3          Soma = 0
4          for i=1 ate Populacao.Tamanho
5                  Soma += Populacao[i].Aptidao
6          for i=1 ate Populacao.Tamanho
7                  Oposto = Populacao.Tamanho +1 -i
8                  Probabilidade[i] = Populacao[Oposto].Aptidao/Soma
9          for i=2 ate Populacao.Tamanho
10                 Probabilidade[i] += Probabilidade[i-1]
11         Sorteio = Fracao_Aleatoria()
12         for i=1 ate Populacao.Tamanho
13                 if Sorteio < Probabilidade[i]
14                         Casal.First = Populacao[i]
15                         Break
16         Casal.Second = Casal.First
17         Enquanto Casal.Second == Casal.First
18                 Sorteio = Fracao_Aleatoria()
19                 for i=1 ate Populacao.Tamanho
20                         if Sorteio < Probabilidade[i]
21                                 Casal.Second = Populacao[i]
22                                 Break
23         Retorna Casal
```

A função Fracao_Aleatoria() retorna um ponto flutuante entre 0 em 1, usado para escolher uma solução de acordo com o vetor de probabilidades. Note que, quanto maior a diferença entre as aptidões da melhor e da pior solução, maior a chance das melhores soluções serem escolhidas.

4.4 Cruzamento de Soluções

Essa etapa, também conhecida como *Crossing Over*, é usada para combinar as duas soluções selecionadas pelo algoritmo anterior. Esse cruzamento ocorre segundo o algoritmo:

```
Cruzamento(Casal)
        Nova.Colunas = Uniao(Casal.First.Colunas,
                                        Casal.Second.Colunas)
        Enquanto Ha_Redundancia(Nova) == true
                Eliminar_Redundancia(Nova)
        Retorna Nova
```

4.5 Eliminar Redundâncias

Uma redundância ocorre em uma solução quando há uma coluna cujas linhas que ela cobre já estão todas cobertas por outras colunas. Nesse caso, a coluna em questão pode ser eliminada, o que melhora a aptidão da solução em que ela estava contida.

```
Eliminar_Redundancia(Solucao)
        Colunas_Redundantes = VAZIO
        for Coluna Col in Solucao.Colunas
                Se Solucao.Linhas[Lin].size() >= 2
                para toda Linha em Col
                        Colunas_Redundantes += Col
        Rem = Escolha_Aleatoria(Colunas_Redundantes)
        Solucao.Colunas -= Rem
        Retorna Colunas_Redundantes.Tamanho
```

4.6 Mutação

Uma mutação pode ocorrer em um novo indivíduo gerado, de acordo com uma taxa de mutação. Essa possui um valor mínimo, definido pelo usuário, e é calculada da seguinte forma:

$$Taxa\_Mutacao = \frac{Taxa\_Min}{1-e^{\frac{-apt_0-apt_n}{apt_0}}}$$

Em que apt_0 é a aptidão da melhor solução e apt_n é a aptidão da pior solução.

```
Mutacao(Populacao, Solucao, Taxa_Minima)
        a = Melhor_Solucao(Solucao).Aptidao
        b = Pior_Solucao(Solucao).Aptidao
        Taxa_Mutacao = Taxa_Minima/(1- (exp((-a -b)/a)))
        Sorteio = Porcentagem_Aleatoria()
        if Sorteio <= Taxa_Mutacao
                for i=1 to Solucao.Numero_de_Colunas
                        Nova_Coluna = Coluna_Aleatoria()
                        Solucao += Nova_Coluna
                Enquanto Ha_Redundancia(Solucao) == true
                        Eliminar_Redundancia(Solucao)
        Retorna Solucao
```

5 Calibragem

Para a calibragem dos parâmetros, utilizou-se uma instância menor do PCC, cuja solução ótima já é conhecida. Esse caso de teste possui 557,44 como valor de aptidão para a melhor solução. Foram variados os parâmetros tamanho de população, indicado pela variável p, e a taxa mínima de mutação, indicada por m, em porcentagem.

Para cada combinação de p e m, 10 testes foram realizados para medir o tempo de execução (em segundos) e o melhor resultado obtido para o problema. As tabelas mostram os resultados médios dessas duas características da execução. O critério de parada de todos os testes de calibragem é o limite máximo de 1000 gerações.

Os testes de calibragem foram realizados em ambiente Linux, rodando em um sistema Intel Pentium 4, 1x3GHz, 500MB de RAM.

Tabela 1: Calibragem - Resultado Médio da Solução Ótima

	m = 5%	m = 6%	m = 7%	m = 8%	m = 9%
p = 100	590,32	589,62	589,40	589,40	588,81
p = 200	597,21	589,29	588,65	588,25	587,65
p = 300	580,85	588,83	588,90	572,45	580,20

Tabela 2: Calibragem - Tempos Médios de Execução (em segundos)

	m = 5%	m = 6%	m = 7%	m = 8%	m = 9%
p = 100	119,17	118,88	119,22	118,59	118,24
p = 200	273,27	273,22	272,71	272,55	272,44
p = 300	464,15	461,26	461,93	462,02	457,40

Dos resultados da calibragem, a combinação mais eficiente de parâmetros foi com $m = 8\%$ e $p = 300$, desconsiderando os tempos de execução. Esses valores serão utilizados para a realização dos testes da sessão seguinte.

6 Testes

A aplicação desenvolvida foi empregada para a realização de testes em oito diferentes instâncias do problema de cobertura de conjuntos. Essas estão contidas em arquivos *.dat* no diretório *./bin* da aplicação. Instruções de como compilar e executar o código estão no arquivo *readme.txt*, junto dos arquivos de código fonte.

A máquina utilizada para a realização dos testes foi a mesma empregada para a calibragem dos parâmetros: um ambiente Linux em um sistema Intel Pentium 4, 1x3GHz, 500MB de RAM. Os parâmetros utilizados foram aqueles obtidos na seção anterior: $m = 8\%$ e $p = 300$ indivíduos. O critério de parada também foi o mesmo usado na etapa anterior, sendo ele o limite de 1000 gerações a partir do início.

Tabela 3: Resultados dos Testes

	N_l	N_c	c_{min}	t_{exec}
Teste_01	50	300	594,06	466,24
Teste_02	50	500	559,25	651,31
Teste_03	50	700	515,3	875,03
Teste_04	100	500	1204,99	824,50
Teste_05	100	700	1063,06	1018,17
Wren_01	200	539	8428	1240,26
Wren_02	222	5522	14428	6248,10
Wren_03	219	4990	14654	5733,37

Na tabela, N_l e N_c são o número de linhas e o número de colunas dos casos de teste apresentados, respectivamente. O valor mínimo de custo característico da melhor solução encontrada está presente na coluna c_{min}. O tempo total de execução para o caso de teste, em segundos, aparece em t_{exec}.

7 Conclusões

A meta-heurística dos algoritmos genéticos permitiu boas aproximações da solução para o problema de cobertura de conjuntos. Mesmo não garantindo soluções exatas, a heurística foi bem empregada para instâncias grandes do problema, em que o tempo de execução do algoritmo exato seria inviável para aplicações práticas.

Entretanto, o tempo de execução do algoritmo genético sofreu um aumento maior do que o esperado para situações em que se acresceu o parâmetro da população máxima, ou até mesmo nos casos em que a instância do problema aumentaram muito de tamanho.

Uma sugestão para futuros trabalhos seria a modificação da função *fitness* para escolha de casais para cruzamentos, uma vez que ela pode percorrer linearmente o vetor população várias vezes para calcular a probabilidade de escolha de um indivíduo para o *crossing over*.

8 Referências

CORMEM, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos: Teoria e Prática**. 3ª Edição. Elsevier, 2012.