

Algoritmo Genético: Problema da Coloração de Grafos

Amanda G. Jabroski

Ciência da Computação – Universidade do Vale do Rio dos Sinos (UNISINOS)
Caixa Postal 275 – 93022-750 – São Leopoldo – RS – Brasil

amanda.grams@gmail.com

Abstract. *This paper will describe the process of implementing a genetic algorithm for the problem of coloring graphs, detailing the main methods and criteria of the implementation.*

Resumo. *Neste artigo será descrito o processo de implementação de um algoritmo genético para o problema da coloração de grafos, detalhando principais métodos e critérios da implementação.*

1. Conceitos Gerais

A coloração de grafos é um problema NP-Completo, e pode ser descrito de forma simples como: Uma maneira de colorir os vértices de um grafo, tal que não podem haver dois vértices adjacentes compartilhando a mesma cor.

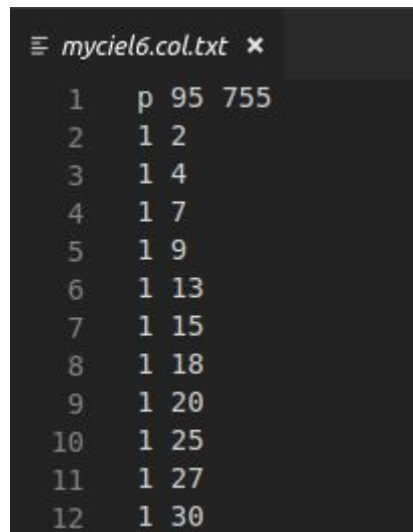
Problemas classificados como NP-Completo tem complexidade exponencial para qualquer algoritmo implementado, desta maneira é importante a utilização de técnicas heurísticas para solução aproximada em tempo polinomial.

2. Implementação

Para implementar o problema de coloração de grafos, representou-se as cores através de números inteiros sem sinal, pois não é importante saber que cores os números representam, embora seja importante saber o número de cores necessárias para colorir o grafo. Criou-se uma classe chamada Indivíduo, que é usada para representar os indivíduos na implementação do algoritmo genético. A função de fitness atribui um número que será a adequação de um indivíduo, se o indivíduo tem menos colisões, sua pontuação será maior, uma vez que o indivíduo não tenha colisões na solução, a aptidão aumentará e o valor do *fitness* também será maior.

2.1. Instâncias Utilizadas

As instâncias de grafos utilizados nesta implementação são: *myciel6*, *games120* e *anna*. O grafo *myciel6* que é baseado na transformação de *Mycielski* é difícil de resolver porque é livre de triângulo deste modo o número de cores aumenta na resolução do problema. O clique máximo no grafo *games120* não é mais que n , e o valor de coloração não é menor que n . O grafo *anna* que é a representação de um grafo de livro, e seus nós representam caracteres, dois nós são conectados por uma aresta se e somente se os caracteres correspondentes de encontrarem.



```
≡ myciel6.col.txt ✕  
1 p 95 755  
2 1 2  
3 1 4  
4 1 7  
5 1 9  
6 1 13  
7 1 15  
8 1 18  
9 1 20  
10 1 25  
11 1 27  
12 1 30
```

Figura 1. Myciel6

A figura 1 traz o exemplo de uma instância de entrada e seu formato de organização. A primeira linha define o tamanho do problema, o primeiro número da primeira linha define a quantidade de nós, e o segundo número da primeira linha define a quantidade de arestas. As demais linhas representam as arestas do grafo, que é definida pela tupla (A,B), em que A e B representam os nós conectados pela data aresta.

3. Algoritmo Genético

Os Algoritmos Genéticos diferem dos métodos de busca e otimização tradicionais, pois trabalham com uma codificação de parâmetros e não com os próprios parâmetros, trabalham com uma população e não com um único ponto, utilizam informações de custo e não derivadas ou outro conhecimento auxiliar, utilizam regras de transição probabilísticas e não determinísticas.

Algoritmos Genéticos são eficientes para busca de soluções ótimas, ou próximas de ótimas, em uma grande variedade de problemas, uma vez que o algoritmo não impõem muitas das limitações encontradas nos métodos de busca tradicionais.

Nesta implementação o sistema começa com uma população que é criada aleatoriamente, e então os melhores indivíduos são escolhidos, a partir destes indivíduos é criada uma nova população aplicando algumas operações.

O programa nunca irá parar porque não sabe se encontrou a melhor solução ou apenas outra solução normal, por isso forneceu-se uma condição de parada. A condição de parada neste caso é: O programa terminará sua execução se oito dos dez indivíduos encontram a solução ou se atingir o número máximo de iterações. Utilizou-se oito e não todos os indivíduos porque o processo de mutação sempre terá impacto sobre alguns dos indivíduos. A operação de mutação é uma das três operações usadas nesta implementação para encontrar novas populações:

Melhores Indivíduos: Os 40% da nova população são encontrados com esta operação, para calculá-lo, o sistema apenas calcula a adequação de todos os indivíduos da população anterior e escolhe os 40% que tem a melhor aptidão

Crossover: Esta operação calcula também 40% dos melhores indivíduos. Para fazer isso basta pegar dois indivíduos do conjunto de "melhores indivíduos" e mesclar seus genes para criar novos indivíduos

Mutação: Os 20% restantes dos indivíduos são calculados com este método, para fazer isso a função de mutação escolhe um indivíduo aleatório e altera aleatoriamente alguns dos seus genes.

4. Execução do Programa

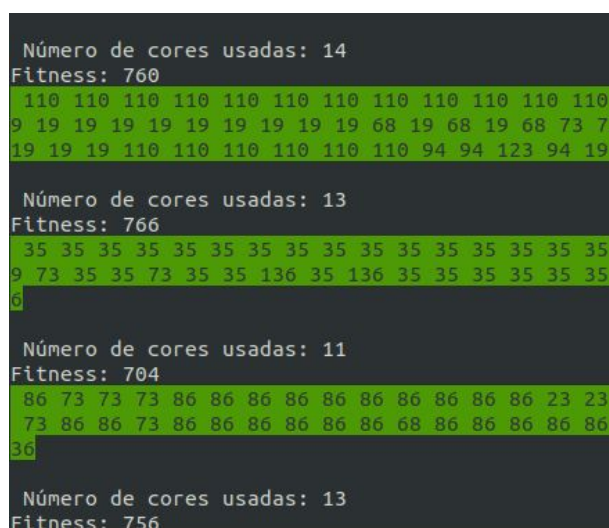
O desenvolvimento foi realizado em ambiente Debian Linux na linguagem C++, para execução do mesmo é necessário compilar através do comando make, conforme mostra a Figura 2.



```
amanda@Amanda-pc: ~/Documentos/ColoracaoGrafoGA
Arquivo Editar Ver Pesquisar Terminal Ajuda
amanda@Amanda-pc:~/Documentos/ColoracaoGrafoGA$ ls
ag.cpp bin entradas include individuo.cpp main.cpp makefile
amanda@Amanda-pc:~/Documentos/ColoracaoGrafoGA$ make
g++ main.cpp ag.cpp individuo.cpp -o bin/main
>> Compilado com sucesso.
amanda@Amanda-pc:~/Documentos/ColoracaoGrafoGA$ ./bin/main
Selecione uma instância:
[0] anna.col
[1] games120.col
[2] myciel6.col
Digite um Número: 0
```

Figura 2. Projeto

A Figura 3 mostra o resultado de uma execução do algoritmo utilizando como entrada a instância *anna*, o algoritmo parou somente quando atingiu o número máximo de iterações, definido no *main*.



```
Número de cores usadas: 14
Fitness: 760
110 110 110 110 110 110 110 110 110 110 110 110 110 110
9 19 19 19 19 19 19 19 19 19 68 19 68 19 68 73 73
19 19 19 110 110 110 110 110 110 110 94 94 123 94 19

Número de cores usadas: 13
Fitness: 766
35 35 35 35 35 35 35 35 35 35 35 35 35 35 35 35
9 73 35 35 73 35 35 136 35 136 35 35 35 35 35 35
6

Número de cores usadas: 11
Fitness: 704
86 73 73 73 86 86 86 86 86 86 86 86 86 86 23 23
73 86 86 73 86 86 86 86 86 86 68 86 86 86 86 86
36

Número de cores usadas: 13
Fitness: 756
```

Figura 3. Finalização Execução

O número de cores utilizadas finalizou em 13 cores, mas é possível verificar que o algoritmo encontrou solução com 11 cores também, isto acontece devido ao fato do

programa não saber qual a melhor solução.

Referências

- Kokosinski, Z. and Kwarciany, K. (2005) “Efficient Graph Coloring With Parallel Genetic Algorithms”, In: Computing and Informatics, Vol. 24., 123–147.
- Abbasian, Reza, and Mouhoub, Malek. 2011. “An efficient hierarchical parallel genetic algorithm for graph coloring problem”, In: Proceedings of The 13th annual conference on Genetic and evolutionary computation, 521-528. Dublin, Ireland: ACM
- Shengning, Wu, and Sikun, Li. 2007. “Extending Traditional Graph-Coloring Register Allocation Exploiting Meta-heuristics for Embedded Systems”. In: Proceedings of The Third International Conference on Natural Computation. ICNC, 324-329. Haikou, Hainan, China
- Croitoru, Cornelius, Luchian, Henri, Gheorghies, Ovidiu, and Apetrei, Adriana. (2002). “A New Genetic Graph Coloring Heuristic”. In: Proceedings of The Computational Symposium on Graph Coloring and its Generalizations, 63-74. Ithaca, New York, USA
- Filho, G. , Lorena, N. (2000) “Constructive Genetic Algorithm and Column Generation: An Application to Graph Coloring”. In: Proc. Asia Pacific Operations Research Symposium.
- Gwee, B. H., Lim, M. H., and Ho, J. S. (1993). “Solving four-colouring map problem using genetic algorithm”. In: Proceedings of First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, 332-333. New Zealand