Minimum Labeling Spanning Trees using Genetic Algorithms Metaheuristics

Mikael Schroeder

Maio 2021

1 Introdução

Esse trabalho apresentará um possível uso de algoritmos genéticos como metaheurística encontrar resoluções possíveis para o problema do minimum labeling spanning trees, tentando deixar a seleção natural ocorrer com as menores influências de outras heurísticas.

2 Problema

Nesse problema recebemos um grafo não orientado com N vértices, M arestas e L labels, sendo que toda aresta que conecta 2 vértices tem um label l pertencente a L. No problema se quer transformar o grafo em uma árvore geradora que minimiza o número de labels distintos. Uma possível formulação em programação linear do problema é apresentada em The k-labeled Spanning Forest Problem[1], sendo a formulação apresentada:

Dado um grafo não direcionado (N, A), onde

- N é um conjunto de n vértices
- A é um conjunto de m arestas

Seja L um conjunto de labels e assumindo que cada vértice {i, j} é associado a um valor li,j $\in L$.

Dado um vérticec i de N, Γ^{-}(i) o conjunto de arcos que terminam em i e Γ^{+}(i) o conjunto de arcos que iniciam em i

Nós queremos cada a árvore geradora T de (N, A) que tem o menor número de labels distintos, ou seja, com os arcos mais parecidos com os outros.

Já que cada aresta $\{i, j\} \in A$ resulta em dois arcos (i,j) e (j,i), chamaremos $A^{'}$ o conjunto de arcos nos quais a ordem não importa, e $A^{'}$l o subconjunto de $A^{'}$ com label l.

Para cada arco $(i, j) \in A^{'}$ consideramos uma variável Xi,j = 1 se arco (i,j) pertence a árvore geradora, ou 0 senão

Variáveis fi,j denotando o fluxo passado pelo arco (i,j)

Para cada $l \in L$ vl = 1 se label l está em algum arco da árvore geradora, ou 0 senão

Com isso se tem o modelo (já com relaxação linear)
min $\sum_{l \in L}$vl
s.a
(1) $\sum_{i\Gamma^-(j)}$ Xij = 1, J $\in N - \{1\}$
(2) $\sum_{i\Gamma^-(j)}$ fij - $\sum_{i\Gamma^+(j)}$ fij = 1
(3) Xij $\leq fij \leq (n-1)Xij, (\text{i,j}) \in A^{'}$
(4) $\sum (i,j) \in A^{'}l$ Xij $\leq min\{n-1, |Al|\}vl, l \in L$
(5) $0 \leq Xij \leq 1$, $(i,j) \in A^{'}$
(6) $0 \leq vl \leq 1$, l $\in L$
(7) fij $\geq 0, (\text{i,j}) \in A^{'}$
(8) $Xij \leq vl$, $l \in L$, $(i,j) \in A^{'}l$

A restrição (1) garante que a árvore geradora da solução tem exatamente 1 arco incidente em cada vértice, (2) garante a conservação do fluxo, deixando 1 unidade de fluxo em cada vértice, (3) implica que o fluxo de cada arco é positivo apenas se ele pertence a árvore geradora e que o número de arcos com labels da solução é menor que o número de vértices menos 1. (5) e (6) são os que garantem a integralidade de Xij e vl, (7) garante que o fluxo nunca é negativo e (8) melhora o limite ao resolver a relaxação linear.

3 Resolução com algoritmo genético

A representação do problema foi feita utilizando uma matrix de adjacência triangular, onde cada elemento dela é ou vazio ou os labels das arestas que conectam os vértices. Essa estrutura de dados foi escolhida para simplificar o fato que o grafo inicial é não direcionado, e também para ser simples acessar e modificar os labels das arestas.

A principal estrutura de dados é a do grafo, que tem a responsabilidade de gerir seus próprios arcos/labels, quebrar ciclos e a cópia dos arcos de uma instância do grafo para outra.

Além disso tem uma pequena auxiliar que é o do arco, que apenas tem sua cor e funcões auxiliares.

Defini cedo que apenas trabalharia com árvores geradoras, logo para gerar as soluções iniciais primeiro se copia o grafo "base" e vai tirando arestas aleatórias que causam ciclos dele até que ele seja uma arvore geradora. O programa fica criando soluções e as salvando até chegar em um certo tamanho de população.

As outras gerações são formadas a partir da anterior, sendo parte dela os melhores da geração anterior, outra parte mutações sobre aleatórios da população e o restante sendo criado por crossover entre pais, dando preferência para pais com menos labels. O fator de elitismo foi inserido para garantir que algumas soluções boas não sejam perdidas, sendo para os testes finais 10% da população. As mutações se copia um grafo aleatório da geração e se adiciona entre 1 e 5 arcos aleatórios do grafo base nela, e então se quebra ciclos até ter uma nova árvore geradora. Na versão final foi decidido que 20% de cada nova população seria criada via mutação, entretanto esse valor funcionou bem para

grafos menores, mas o limite de no máximo 5 novos arcos acabou tendendo a ser irrelevante quando foi testado com grafos maiores.

Cada pai do crossover é selecionado baseado em um sistema de camadas, onde se separa a população em n camadas, e camadas menores tem maior chance de serem escolhidas. Para cada camada ser definida foi pego o número de labels do grafo com mais labels e dividida a diferença entre eles pelo número de camadas, e então a população foi separada com baseada em quantas vezes se deve adicionar esse número ao número de labels que a melhor arvore geradora possui. Por exemplo, tendo uma lista com o número de labels que cada elemento tem e 3 tiers, [1,2,4,5,5,5,10] as camadas seriam baseadas em se o elemento tem mais cores que 1 + (camada atual)*3, ou seja ficariam camada 1 = [1,2,4], 2 = [5,5,5] e 3 = [10]. A preferência por camadas mais baixas vem do fato da seleção de qual camada vai vir o pai começar das camadas mais baixas e ir em direção as mais altas, e quando chega na última volta para a primeira até selecionar uma. Da camada selecionada se escolhe um pai aleatoriamente, sem nenhuma preferência. O filho recebe todos arcos de ambos os pais e, quando vai quebrar os ciclos para virar uma árvore geradora ele retira apenas arcos que pertencem a um único pai. O motivo da seleção de camadas ter uma preferência apenas levemente maior para as melhores foi porque o programa já tinha bastante elitismo por passar sempre os 10% melhores resultados, queria então aumenar a diversidade de soluções possíveis.

Para os resultados foi selecionado tamanho de população = 20 e com parada após 200 gerações, esses parâmetros encontraram um balanço ok entre tempo de execução e melhora gradual das respostas. Foi testado também como fator adicional que poderia parar cedo a execução o número de gerações sem melhora no número de labels, entretanto utilizar ambos fatores acabava baseado no número de gerações para qualquer número de gerações sem melhoras razoavelmente grande, e tentar apenas pelo número de gerações era demorado demais.

Utilizando as informações do GLPK do github base[2] enquanto as minhas não estão prontas. Marcadas com *

- – Instância: 100-990-25-3-1
  - – Relaxação linear (GLPK): N/A
  - – Melhor solução inteira (GLPK): 8*
  - – Tempo execução (GLPK): 2h*
  - – Solução inicial média (AG): 23
  - – Solução final média (AG): 20.3
  - – Desvio padrão das melhores soluções (AG): 0.675
  - – Tempo médio execução (AG): 156.38sec
  - – Desvio % entre AG e GLPK: 153.75%
- – Instância: 100-990-25-3-4
  - – Relaxação linear (GLPK):

  - Melhor solução inteira (GLPK): 5*
  - Tempo execução (GLPK): 2h*
  - Solução inicial média (AG): 23.2
  - Solução final média (AG): 20.4
  - Desvio padrão das melhores soluções (AG): 0.84
  - Tempo médio execução (AG): 156.46sec
  - Desvio % entre AG e GLPK: 308%

- – Instância: 100-990-50-6-7
  - Relaxação linear (GLPK):
  - Melhor solução inteira (GLPK): 14*
  - Tempo execução (GLPK): 2h*
  - Solução inicial média (AG): 39.6
  - Solução final média (AG): 31.9
  - Desvio padrão das melhores soluções (AG): 0.99
  - Tempo médio execução (AG): 178.96sec
  - Desvio % entre AG e GLPK: 127.85%

- – Instância: 100-990-50-6-8
  - Relaxação linear (GLPK):
  - Melhor solução inteira (GLPK): 7*
  - Tempo execução (GLPK): 2h*
  - Solução inicial média (AG): 39.9
  - Solução final média (AG): 31.8
  - Desvio padrão das melhores soluções (AG): 1.22
  - Tempo médio execução (AG): 100.95
  - Desvio % entre AG e GLPK: 354.28%

- – Instância: 100-990-100-6-5
  - Relaxação linear (GLPK):
  - Melhor solução inteira (GLPK): 17*
  - Tempo execução (GLPK): 2h*
  - Solução inicial média (AG): 57.6
  - Solução final média (AG): 43.5
  - Desvio padrão das melhores soluções (AG): 1.71
  - Tempo médio execução (AG): 109.67 sec
  - Desvio % entre AG e GLPK: 155.88%

- – Instância: 100-990-125-7-4
  – Relaxação linear (GLPK):
  – Melhor solução inteira (GLPK): 11*
  – Tempo execução (GLPK): 2h*
  – Solução inicial média (AG): 61.3
  – Solução final média (AG): 47.1
  – Desvio padrão das melhores soluções (AG): 2.96
  – Tempo médio execução (AG): 204.99sec
  – Desvio % entre AG e GLPK: 328.18%

4 Resultados e conclusões

Os resultados dessa implementação da metaheurística deixaram muito a desejar, tendo altíssimos desvios entre a melhor solução conhecida e a encontrada. Parte do motivo das soluções serem tão ruins é ter utilizado populações tão pequenas e um número relativamente pequeno de gerações, entretanto aumentar o tamanho da população aumentava muito o tempo de processamento (além do tempo de execução da lista, a criação inicial de cada elemento tendia a demorar ¿1 sec, e maiores populações aumentam o tempo de simulação também).

Outra causa desses resultados ruins é a maneira que os ciclos foram quebrados para transformar os grafos em árvores geradoras, pois o programa não se preocupa com qual o label que um arco possuia, seria possível ter também alguma heurística que dá preferência para remover arcos com labels menos utilizados. Entretanto tal heurística escapa da ideia original da escolha aleatória, que foi de ser um algoritmo genético com a menor quantidade de viéses externos.

Mais um problema que ocorreu foi a limitação da mutação a ser no máximo 5 arcos. Mutações servem para aumentar a variedade genética da população, mas por esse limite ter sido pensado para instâncias de teste menores, nas maiores foi uma diferença desprezível.

Em conclusão, a tentativa de solução puramente com a metaheurística de algoritmos genéticos não é suficiente para ter um resultado próximo do melhor resultado. Possíveis maneiras de melhorar eles seriam aumentar o tamanho das populações, número de gerações ou implementar mais heurísiticas que possam acelerar a melhora dos resultados deles.

5 Referencias

[1] R. Cerulli, A. Fink, M. Gentili, A. Raiconi, The k-labeled Spanning Forest Problem, Procedia - Social and Behavioral Sciences, Volume 108, 2014, Pages 153-163, ISSN 1877-0428, https://doi.org/10.1016/j.sbspro.2013.12.828. (https://www.sciencedirect.com/science/article/pii/S1877042813054682)

[2] https://github.com/tiagodrehmer/INF05100_Problems