Algoritmo Genético e Programa Linear Inteiro para o Problema da Dominação Romana Total*

Heric da Silva Cruz[1], **Atílio Gomes Luiz**[1]

[1]Campus Quixadá – Universidade Federal do Ceará (UFC)

hericsilvaho@gmail.com, gomes.atilio@ufc.br

Abstract. *Given a graph $G = (V, E)$, a function $f : V \to \{0, 1, 2\}$ is a total Roman dominating function (TRDF) if every $v \in V$ with $f(v) = 0$ has a neighbor $u \in V$ with $f(u) = 2$, and every $v \in V$ with $f(v) > 0$ has a neighbor $u \in V$ with $f(u) > 0$. The weight of f is given by $\omega(f) = \sum_{v \in V} f(v)$, and the total Roman domination number of G is $\gamma_{tR}(G) = \min\{\omega(f) : f$ is a TRDF of $G\}$. In this work, we propose an integer linear program and a genetic algorithm to compute $\gamma_{tR}(G)$, evaluating the effectiveness of the latter on various instances.*

Resumo. *Dado um grafo $G = (V, E)$, uma função $f : V \to \{0, 1, 2\}$ é uma função de dominação romana total (FDRT) se todo $v \in V$ com $f(v) = 0$ possui um vizinho $u \in V$ com $f(u) = 2$, e todo $v \in V$ com $f(v) > 0$ possui um vizinho $u \in V$ com $f(u) > 0$. O peso de f é $\omega(f) = \sum_{v \in V} f(v)$ e o número de dominação romana total de G é $\gamma_{tR}(G) = \min\{\omega(f) : f$ é FDRT de $G\}$. Neste trabalho, propomos um programa linear inteiro e um algoritmo genético para calcular $\gamma_{tR}(G)$, avaliando a eficácia deste último em instâncias diversas.*

1. Introdução

Em 2016, Ahangar et al. introduziram o conceito de dominação romana total em grafos [Abdollahzadeh Ahangar et al. 2016], que é uma variante do conceito clássico de dominação romana [Cockayne et al. 2004]. Formalmente, dado um grafo $G = (V, E)$, sem vértices isolados, uma função $f : V \to \{0, 1, 2\}$ é uma *função de dominação romana total* (FDRT) de G se todo $v \in V$ com $f(v) = 0$ tem um vizinho $u \in V$ com $f(u) = 2$, e todo $v \in V$ com $f(v) > 0$ tem um vizinho $u \in V$ com $f(u) > 0$. O peso de uma FDRT f é dado por $\omega(f) = \sum_{v \in V} f(v)$. O *número de dominação romana total* do grafo G, denotado por $\gamma_{tR}(G)$, corresponde ao menor peso possível entre todas as funções de dominação romana total em G, ou seja, $\gamma_{tR}(G) = \min\{\omega(f) : f$ é uma FDRT de $G\}$. O *Problema de Dominação Romana Total* (PDRT) consiste em determinar $\gamma_{tR}(G)$ para um grafo arbitrário G. Sabe-se que a versão de decisão desse problema é NP-completo mesmo quando restrita a grafos de intervalo [Liu and Chang 2013]. Logo, faz-se necessário o desenvolvimento de abordagens heurísticas para o problema.

Neste trabalho, propomos o primeiro modelo de programação linear inteira (PLI) e o primeiro algoritmo genético para o PDRT, analisando a eficácia do algoritmo genético em diferentes instâncias do problema. Modelos de PLI e meta-heurísticas já foram propostos na literatura para problemas correlatos [Cai et al. 2022, Poureidi and Fathali 2023, Billore and Reddy 2023, Khandelwal et al. 2021].

*Este trabalho foi financiado pela bolsa PIBIC-ICT-FUNCAP, Processo BP2-0229-00026.01.53/24

Na Seção 2 apresentamos o PLI para o PDRT. Na Seção 3 apresentamos o algoritmo genético. Por fim, os resultados experimentais e as conclusões deste trabalho são descritos na Seção 4.

2. Programa Linear Inteiro

Dado um grafo $G = (V, E)$, definimos duas variáveis binárias x_v e y_v para cada vértice $v \in V$ tais que: $x_v = 1$ se $f(v) = 1$, ou $x_v = 0$ caso contrário; $y_v = 1$ se $f(v) = 2$, ou $y_v = 0$ caso contrário. A formulação do programa linear inteiro para o Problema da dominação romana total é, portanto, dada abaixo.

$$\text{minimize} \quad \sum_{v \in V} x_v + 2 \sum_{v \in V} y_v \tag{1a}$$

$$\text{subject to} \quad x_v + y_v + \sum_{w \in N(v)} y_w \geq 1 \qquad \forall v \in V, \tag{1b}$$

$$\sum_{w \in N(v)} (x_w + y_w) \geq x_v + y_v \quad \forall v \in V, \tag{1c}$$

$$x_v + y_v \leq 1 \qquad \forall v \in V, \tag{1d}$$

$$x_v, y_v \in \{0, 1\} \qquad \forall v \in V. \tag{1e}$$

A expressão (1a) fornece o valor da função objetivo ($\gamma_{tR}(G)$). As restrições (1b) garantem que todo vértice v com $f(v) = 0$ tenha um vizinho u com $f(u) = 2$. As restrições (1c) garantem que todo vértice v com $f(v) \in \{1, 2\}$ tenha um vizinho com peso diferente de 0. As restrições (1d) asseguram que cada vértice receba no máximo um rótulo do conjunto $\{1, 2\}$. As restrições (1e) asseguram que as variáveis x_v, y_v sejam binárias. Este modelo foi usado para computar valores ótimos de FDRT das instâncias e comparar com os resultados aproximados obtidos pelo algoritmo genético.

3. Algoritmo Genético

Algoritmos Genéticos (AGs) são meta-heurísticas de otimização inspiradas na evolução biológica, introduzidas por John Holland [Holland 1992]. Esses algoritmos realizam uma busca pelo espaço de soluções do problema em questão simulando o processo evolutivo através de operações como seleção, cruzamento e mutação, aprimorando um conjunto de soluções iniciais (cromossomos) ao longo de sucessivas gerações. Algoritmos genéticos são amplamente discutidos em livros [Holland 1992, Goldberg 1989]. Uma resenha recente sobre AGs pode ser encontrada em [Katoch et al. 2021].

A seguir, apresentamos o nosso algoritmo genético para o PDRT. Para isso, descrevemos brevemente os componentes principais de um AG que precisaram ser adaptados para o problema em questão: representação da solução; geração da população inicial de soluções; operadores de cruzamento, seleção e mutação.

3.1. Representação da solução

Dado um grafo $G = (V, E)$, sem vértices isolados, com $V = \{v_0, v_1, \ldots, v_{n-1}\}$, uma *solução (cromossomo)* para G é representada como um vetor de n inteiros $A = [0..n-1]$ onde $A[i] \in \{0, 1, 2\}$, e $A[i]$ corresponde ao rótulo $f(v_i)$ do vértice v_i, para todo $0 \leq i \leq n-1$. Assim, o *valor de aptidão* de uma solução A é o valor $\sum_{i=0}^{n-1} A[i]$.

3.2. Geração da população inicial de soluções

O nosso AG não permite que soluções inviáveis evoluam ao longo do processo. A população inicial de cromossomos é, portanto, composta por soluções viáveis, que são geradas por cinco heurísticas, brevemente descritas a seguir.

Heurística 1: Dado um grafo $G = (V, E)$, sem vértices isolados, crie uma cópia $H = (V_H, E_H)$ do grafo $G = (V, E)$. Enquanto $V_H \neq \emptyset$, faça: (i) escolha aleatoriamente um vértice $v_i \in V_H$ e atribua o rótulo $A[i] = 2$; (ii) escolha um vizinho de v_i, digamos v_j, e faça $A[j] = 1$ e os demais vizinhos de v_i recebem rótulo 0; (iii) Remova o vértice v_i e seus vizinhos do grafo H; (iv) Após a remoção, alguns vértices podem ter ficado isolados. Cada vértice isolado $v_\ell \in V_H$ recebe rótulo $A[\ell] = 1$. Como o grafo original G não possui vértices isolados, v_j tem pelo menos um vizinho v_k em G, logo esse vizinho dele também recebe rótulo $A[k] = 1$ a fim de satisfazer a propriedade da FDRT. (v) O processo descrito nas etapas (i) a (iv) se repete até que todos os vértices estejam rotulados.

A **Heurística 2** é similar à **Heurística 1**, mas prioriza vértices de maior grau ao invés de escolha aleatória. Isso reduz a quantidade de rótulos 2 utilizados, potencialmente melhorando a solução. A **Heurística 3** aperfeiçoa a **Heurística 2** ao garantir que o vizinho v_k de um vértice v_ℓ escolhido para receber $A[k] = 1$ seja aquele com maior grau. A **Heurística 4** é uma melhoria da **Heurística 3**, tratando vértices isolados de forma mais refinada. Se houver um conjunto de vértices isolados S, seus vizinhos são analisados e os rótulos são redistribuídos a fim de reduzir o peso total da rotulação. A **Heurística 5** é trivial e atribui o rótulo 1 a todos os vértices. A Heurística 1 gera 60% da população inicial e cada uma das demais heurísticas gera 10% do restante da população.

3.3. Operadores de seleção

No nosso AG, aplicamos a técnica clássica de elitismo [Goldberg 1989], onde os t melhores cromossomos gerados na iteração anterior são copiados para a população atual, onde t é um parâmetro do AG. Além disso, como operador de seleção de progenitores, utilizamos o método *k-tournament selection*, descrito em [Goldberg and Deb 1991], onde um subconjunto de k indivíduos é escolhido aleatoriamente da população gerada na iteração anterior, e o que possui o melhor valor de aptidão (o menor valor) é selecionado como progenitor. O inteiro k também é um parâmetro do AG.

3.4. Operadores de cruzamento e mutação

Como operador de cruzamento, usamos o clássico *cruzamento de um ponto* (do inglês, *one-point crossover*) [Kora and Yadlapalli 2017]. Como operador de mutação, usamos o seguinte método *ad hoc*: para cada $i \in \{0, \ldots, n-1\}$ geramos um número aleatório $r \in \{0, \ldots, 1\}$ e se r for menor que uma taxa de mutação escolhida, então fazemos $A[i] = 0$. Os cromossomos obtidos após o cruzamento de dois progenitores ou após a aplicação da mutação podem não ser soluções viáveis. Logo, esses cromossomos passam por um processo de correção, garantindo que satisfaçam as restrições de uma FDRT.

4. Resultados Experimentais e Conclusão

Para avaliar o algoritmo proposto, foram utilizados 327 grafos provenientes de diferentes coleções: 62 grafos da coleção DIMACS, 63 da coleção Miscellaneous Networks, ambos disponíveis no repositório Network Repository [Rossi and Ahmed 2015], 173 grafos

da coleção Harwell-Boeing [National Institute of Standards and Technology] e 30 grafos cúbicos gerados aleatoriamente de acordo com o modelo Erdös–Rényi.

O algoritmo genético possui 8 parâmetros de entrada, cujos valores foram ajustados automaticamente com o auxílio da ferramenta *irace* [López-Ibáñez et al. 2016]. Como resultado do processo de configuração automática conduzido pelo *irace*, os seguintes valores foram selecionados: o número máximo de gerações foi definido como 586, e o número máximo de gerações sem melhoria como 363. A taxa de cruzamento escolhida foi 0,4337, enquanto a taxa de mutação foi ajustada para 0,057. A taxa de elitismo ficou em 0,2262, e o tamanho do torneio foi definido como 6. O fator populacional foi configurado como 4,4056, ou seja, o tamanho da população para cada grafo G é igual a $|V(G)|/4{,}4056$. Por fim, o número de repetições por instância foi fixado em 10, permitindo uma avaliação do desempenho do algoritmo.

O PLI foi executado por até cinco minutos em cada uma das 327 instâncias avaliadas. Em 167 casos, ele obteve soluções ótimas, enquanto o algoritmo genético (GA) coincidiu com esse valor em 54 instâncias. Em 72 casos, ambos os métodos retornaram exatamente o mesmo valor, embora nem sempre o ótimo. A diferença percentual entre os resultados (GAP) apresentou média de $11{,}58\%$, mediana de $6{,}07\%$ e desvio padrão de $16{,}25\%$, com valores entre $0{,}00\%$ e $128{,}57\%$. O tempo médio de execução do GA foi de 35.048 ms, com mínimo de 272 ms e máximo de 840.768 ms.

A densidade média dos grafos foi de $0,1852$, variando entre $0,00098$ e $1,0$. A ordem média foi de 590 vértices (mínimo de 9 e máximo de 3.973), e o número médio de arestas foi de 15.630, com extremos de 16 a 410.781. O desempenho do GA foi melhor, em média, em grafos de baixa densidade, com um GAP médio de $9{,}48\%$, inferior ao observado em grafos de densidade média ($17{,}65\%$) e alta ($14{,}53\%$). Esse padrão sugere que grafos esparsos tendem a conseguirem o melhor resultado no algoritmo genético executado até a estagnação, frequentemente resultando em GAPs mais baixos, incluindo diversos casos em que o valor foi nulo. Por outro lado, os maiores valores de GAP foram registrados principalmente em instâncias com densidade intermediária, o que indica que essa faixa representa um maior desafio para o método. Já nos grafos mais densos, o desempenho do GA foi relativamente mais estável, com vários casos em que ele conseguiu igualar a solução ótima. Esses resultados indicam que a densidade influencia diretamente o desempenho do GA, sendo a faixa intermediária a mais crítica.

Em 34 instâncias, o GA superou o PLI, todas em grafos cúbicos de baixa densidade. Isso indica que essa topologia influencia mais o desempenho relativo dos métodos do que apenas ordem ou densidade. Observou-se correlação positiva do GAP com o número de arestas (+0,33), sugerindo que grafos mais densos tendem a apresentar maior divergência entre os resultados.

Em oito instâncias, o GA coincidiu com a heurística `h2`, e em 7 delas o valor correspondia ao ótimo, destacando a importância de boas estratégias iniciais. O código-fonte desenvolvido neste trabalho está disponível no repositório GitHub: `https://github.com/hscHeric/cl-total-rdga`.

Como trabalhos futuros, pretende-se investigar operadores de cruzamento especializados para problemas de dominação em grafos, bem como aprofundar o estudo da influência da topologia das instâncias nos resultados obtidos.

Referências

Abdollahzadeh Ahangar, H., Henning, M., Samodivkin, V., and G. Yero, I. (2016). Total roman domination in graphs. *Applicable Analysis and Discrete Mathematics*, 10:17–17.

Billore, A. and Reddy, P. V. S. (2023). Genetic algorithm based approach for solving roman 3-domination problem. In *2023 International Conference on Computer, Electronics & Electrical Engineering & their Applications (IC2E3)*, pages 1–6.

Cai, Q., Fan, N., Shi, Y., and and, S. Y. (2022). Integer linear programming formulations for double roman domination problem. *Optimization Methods and Software*, 37(1):1–22.

Cockayne, E. J., Dreyer, P. A., Hedetniemi, S. M., and Hedetniemi, S. T. (2004). Roman domination in graphs. *Discrete Mathematics*, 278(1):11–22.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. volume 1 of *Foundations of Genetic Algorithms*, pages 69–93. Elsevier.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA.

Katoch, S., Chauhan, S. S., and Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80:8091–8126.

Khandelwal, A., Srivastava, K., and Saran, G. (2021). On roman domination of graphs using a genetic algorithm. In Singh, V., Asari, V. K., Kumar, S., and Patel, R. B., editors, *Computational Methods and Data Engineering*, pages 133–147, Singapore. Springer Singapore.

Kora, P. and Yadlapalli, P. (2017). Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162:34–36.

Liu, C.-H. and Chang, G. J. (2013). Roman domination on strongly chordal graphs. *Journal of Combinatorial Optimization*, 26(3):608–619.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

National Institute of Standards and Technology. Matrix Market: A Visual Repository of Test Data for Sparse Matrix Algorithms. Disponível em: `https://math.nist.gov/MatrixMarket/`. Data de acesso: Março de 2025.

Poureidi, A. and Fathali, J. (2023). Algorithmic results in roman dominating functions on graphs. *Information Processing Letters*, 182:106363.

Rossi, R. A. and Ahmed, N. K. (2015). The network data repository with interactive graph analytics and visualization. In *AAAI*. Disponível em: `https://networkrepository.com`. Data de acesso: Março de 2025.