Algoritmo Genético para o Problema do Emparelhamento Diversificado

Lucas Lima de Melo

29 de dezembro de 2021

1 Introdução

O objetivo deste trabalho é implementar a metaheurística algoritmo genético para resolver o problema do Emparelhamento Diversificado.

O problema do Emparelhamento Diversificado é definido como: Dado um grafo não direcionado G=(V,A) onde cada aresta $a \in A$ possui um tipo ta, deseja-se encontrar um emparelhamento,ou seja um conjunto de arestas tal que não existem duas arestas adjacentes neste conjunto, que maximize o número de arestas nele contido,no qual todas arestas possuem tipos diferentes.

2 Formulação Como Programa Inteiro

- $x_e \in \{0,1\}, \quad \forall e \in A$, onde

$$x_e = \begin{cases} 1, & \text{Caso a aresta e faça parte do emparelhamento} \\ 0, & \text{Caso contrario} \end{cases}$$

- $y_{ek} \in \{0,1\}, \quad \forall e,k \mid e \in A, k \in C$, onde

$$y_{ek} = \begin{cases} 1, & \text{Caso k seja a cor da aresta e} \\ 0, & \text{Caso contrario} \end{cases}$$

- $z_{ek} \in \{0,1\}, \quad \forall e,k \mid e \in A, k \in C$, onde

$$z_{ek} = \begin{cases} 1, & \text{Caso } x_e \wedge y_{ek} \text{ é verdade} \\ 0, & \text{Caso contrario} \end{cases}$$

Função Objetivo:

$$\text{Max.} \quad \left(\sum_{e \in A} x_e \right)$$

Restrições:

$$\sum_{e \in A \ \wedge \ e\#v} x_e \leq 1, \quad \forall v \in V, \ \text{e\#v representa que a aresta e é incidente em v.} \tag{1}$$

$$y_{ek} = 0, \quad \forall e, k \mid e \in A, k \in C \wedge k \neq t_e \tag{2}$$

$$\sum_{k \in C} y_{ek} = 1, \quad \forall e \in A \tag{3}$$

$$\sum_{e \in A} z_{ek} \leq 1, \quad \forall k \in C \tag{4}$$

C representa o conjunto de cores(tipos) únicas.
A restrição (1) garante que as arestas não sejam incidentes.
A restrição (2) garante que y_{ek} seja 0 quando a cor k for diferente da cor de entrada t_e da aresta e.
A restrição (3) garante que cada aresta possua exatamente 1 cor.
A restrição (4) garante que no máximo 1 aresta de cor k faça parte do emparelhamento.

3 Algoritmo Genético

3.1 Parâmetros

Os parâmetros utilizados são:

- **Population_Size:** Este parâmetro representa a quantidade de indivíduos na população.
- **New_p_Size:** Este parâmetro representa a quantidade de novos indivíduos que serão gerados a cada iteração do algoritmo.
- **Mutation_Rate:** Este parâmetro representa a taxa de mutação, valor entre 0 e 1.
- **Max_Non_Improving_Gen:** Este parâmetro representa um dos critérios de parada do programa, que irá terminá-lo caso 'Max_Non_Improving_Gen' gerações forem geradas sem melhora na solução final.
- **Max_time:** Tempo limite: o algoritmo executará por no máximo Max_time segundos.

3.2 População Inicial

A população inicial é gerada aleatoriamente.São gerados 'Population_Size' vetores binários aleatórios,como provavelmente estes vetores não serão factíveis, uma aresta,representada por um gene, é escolhida aleatoriamente para fazer parte do emparelhamento e a solução é refactibilizada a partir desta aresta.A refactibilização é feita percorrendo a lista de genes.Caso o gene seja 1, ele é testado conforme as condições do emparelhamento diversificado, levando em consideração a aresta escolhida anteriormente que já faz parte do emparelhamento e setado para 0 caso não possa participar do emparelhamento.Assim no final,1 indica que a aresta faz parte do emparelhamento e 0 caso contrário.

3.3 Método de Seleção de Indivíduos para Crossover

O Método de Seleção de Indivíduos para Crossover é feito através de torneio, onde k indivíduos são selecionados para o torneio e é retornado o indivíduo com maior fitness. Ocorrem duas execuções do método para escolher dois indivíduos que serão, posteriormente, recombinados.

3.4 Crossover

O crossover é feito através do método de Uniform-Crossover.Dado os indivíduos p1 e p2,cada bit do indivíduo p1 tem 50 % de chance de ser trocado com o bit correspondente do indivíduo p2, gerando assim os dois novos indivíduos que serão retornados.Também é necessário executar uma função que refactibilize o indivíduo, pois o processo pode levar a um indivíduo infactível.A refactibilização é feita percorrendo a lista de genes.Caso o gene seja 1, ele é testado conforme as condições do emparelhamento diversificado,e setado para 0 caso ele não possa participar do emparelhamento.O algoritmo de refactibilização possui complexidade de O(N) para o melhor caso e O(N^2) no pior caso.

3.5 Mutação

A mutação é feita através de uma variação do método de Bit-Flip-Mutation, onde k bits são escolhidos e se o valor do bit for 0, seu valor é trocado para 1, com a probabilidade da troca acontecer dada pelo parâmetro Mutation_Rate.Esta operação também pode causar infactibilidade, portanto também é necessário refactibilizar a solução.A refactibilização possui a mesma lógica descrita no crossover.

3.6 Critério de Parada

O algoritmo para sua execução e retorna a melhor solução encontrada até o momento, caso:

- Caso **Max_time** seja atingido.
- Caso **Max_Non_Improving_Gen** seja atingido.

3.7 Seleção de Indivíduos da Nova População

A Seleção de indivíduos da nova População é feita retirando quantos indivíduos novos foram gerados na nova população .Como New_p_Size novos indivíduos são gerados por geração através de crossover e mutação, os New_p_Size indivíduos com piores fitness irão sair da população agregada(população atual + novos indivíduos gerados).

3.8 Pseudocódigo

Algorithm 1 Algoritmo Genético

```
1: populacao ← GeraPopulacaoInicial(Population_Size)
2: melhor_solucao ← Torneio(populacao, tamanho(populacao))
3: geracoes_sem_melhora ← 0
4: while Not CriteriosDeParada() do
5:    nova_populacao ← ∅
6:    while tamanho(nova_populacao) < New_p_Size do
7:       p1 ← Torneio(populacao, k)
8:       p2 ← Torneio(populacao, k)
9:       filho1, filho2 ← Crossover(p1, p2)
10:      filho1 ← Mutacao(filho1, Mutation_Rate)
11:      filho2 ← Mutacao(filho2, Mutation_Rate)
12:      nova_populacao ← nova_populacao ∪ filho1 ∪ filho2
13:   end while
14:   for cromossomo ∈ nova_populacao do
15:      cromossomo ← Refactibiliza(cromossomo)
16:      populacao ← populacao ∪ cromossomo
17:   end for
18:   populacao ← RemovePiores(populacao, New_p_Size)
19:   nova_solucao ← Torneio(populacao, tamanho(populacao))
20:   if nova_solucao ≤ melhor_solucao then
21:      geracoes_sem_melhora ← geracoes_sem_melhora + 1
22:   else
23:      melhor_solucao ← nova_solucao
24:      geracoes_sem_melhora ← 0
25:   end if
26: end while
27: return melhor_solucao
```

4 Plataforma de Implementação

O trabalho foi implementado usando a linguagem Python 3.8 e usa somente bibliotecas padrões.A plataforma utilizada foi o Windows 10 com processador Intel(R) Core(TM) i5 com 6 núcleos e cache L2 de 1.5MB e 16GB de memória.

4.1 Estrutura de Dados

A representação de um cromossomo é feita utilizando uma lista com elementos binários, como dito anteriormente.

4.2 Cálculo do Fitness de um cromossomo

O cálculo do fitness de um cromossomo é feito simplesmente percorrendo a lista com os os genes do cromossomo contando o número de genes iguais a "1", ou seja contando quantas arestas estão presente no emparelhamento representado por esse cromossomo.

5 Teste de Parâmetros e Operadores

Os testes foram realizados usando a configuração inicial Population_Size=40,Mutation_Rate=0.9, Max_Non_Improving_Gen=30,Max_Time=1800(seg) e New_p_Size=6.Todos os testes foram realizados com as instâncias RM01 e RM02 ao menos utilizando 5 seeds diferentes.

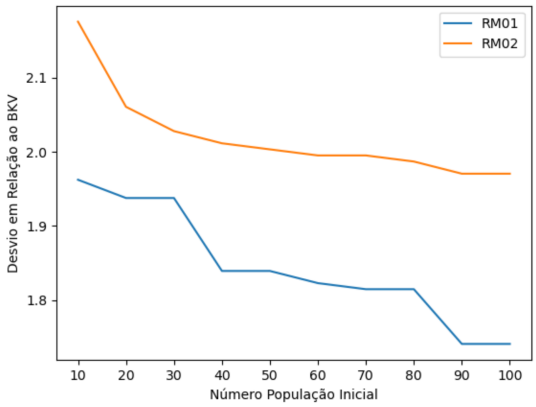
Para cada teste foram analisados o desvio em relação ao BKV e o tempo de execução médio.O desvio em relação ao BKV foi calculado da seguinte maneira, onde S é a média das soluções encontrados e BKV é o melhor valor conhecido.

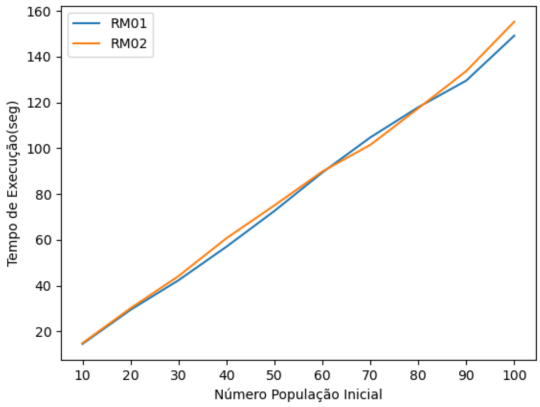
$$\frac{BKV - S}{BKV} \cdot 100$$

Portanto quanto menor for o desvio, melhor é a média das soluções.

5.1 Teste do parâmetro Population_Size

Para este teste foram avaliados somente o quanto a população inicial influencia na solução inicial, sem os operadores de crossover e mutação.





Neste teste podemos observar uma pequena melhora de aproximadamente 0.2 % no desvio da solução em relação ao BKV variando do menor para o maior valor da população, porém observa-se também um grande aumento no tempo de execução quanto maior for este número .Foi escolhido o número 40 por possuir bons resultados nas instâncias de teste e um tempo de execução relativamente baixo de aproximadamente um minuto.

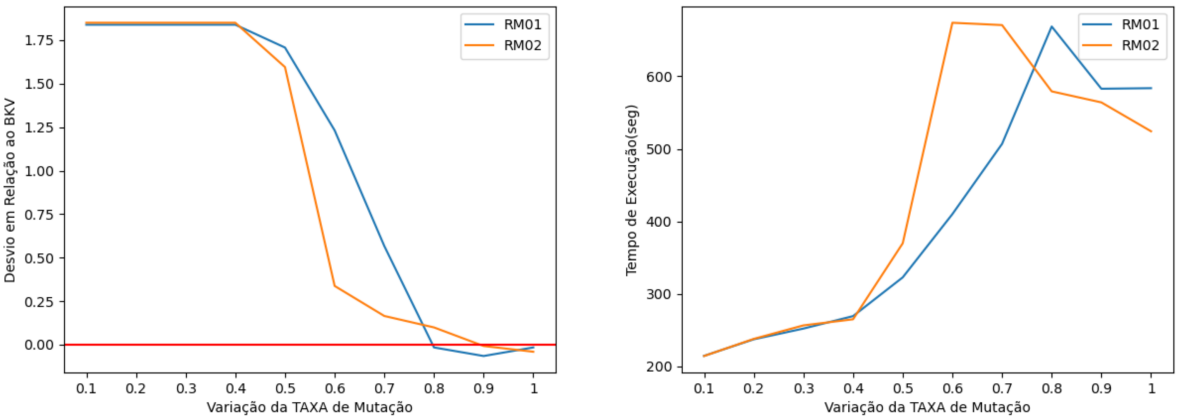
5.2 Teste do operador Crossover

Instância	Valor Médio da Solução Inicial (SI)	Tempo Médio de Execução do SI (segundos)	Valor Médio da Solução com Crossover (SC)	Tempo Médio de Execução do SC (segundos)	Desvio da Solução SC em Relação ao SI(%)	Aumento do Tempo SC em Relação ao SI (%)
RM01	2391	56.5	2391	193.8	0	243.00
RM02	2387	60.4	2387	196.8	0	225.82

Neste teste podemos observar que somente o operador de crossover não conseguiu produzir valores melhores que a solução inicial, isto se dá pelo fato das múltiplas infactibilidades que podem ser geradas através desse operador.É possível observar também que o tempo aumenta com um desvio médio de aproximadamente +235% em relação a solução inicial,fato este também devido a ter que tratar estas infactibilidades.

5.3 Teste do operador de mutação com o parâmetro Mutation_Rate

Neste teste foi implementado o operador de mutação e analisado o parâmetro Mutation_Rate variando de 0.1 a 1 com incrementos de 0.1. A configuração continuou Population_Size=40, Max_Non_Improving_Gen=30,Max_Time=1800(seg) e New_p_Size=6



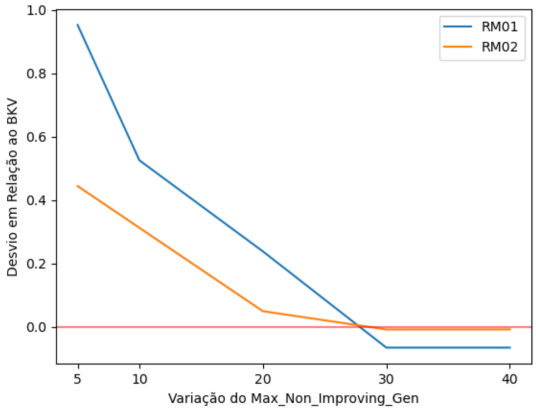
Analisando os resultados do teste, podemos perceber que com essa configuração, o desvio em relação ao BKV começa a diminuir apenas com taxa de mutações maiores ou iguais a 0.5 com média melhor que o BKV para taxas entre 0.8 e 1 , o que indica que o operador de mutação possui um impacto grande na geração de soluções melhores.

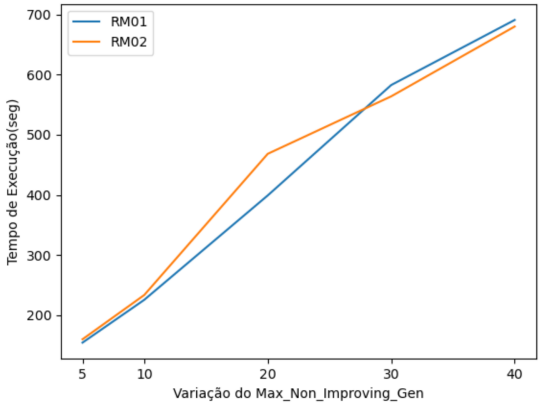
O tempo de execução aumenta com aumento da taxa com valores entre 0.1 e 0.8 aproximadamente,observa-se também que ocorre uma diminuição do tempo com taxas de 0.9 e 1 em relação ao pico máximo do tempo.

Foi escolhida a taxa de mutação com o valor 0.9 para configuração final por apresentar o melhor balanceamento entre desvio e tempo juntamente com a taxa de 1.0.

5.4 Teste do parâmetro Max_Non_Improving_Gen

Neste teste foi analisado o parâmetro Max_Non_Improving_Gen variando entre os valores{5,10,20,30,40}. A configuração utilizada foi a inicial.





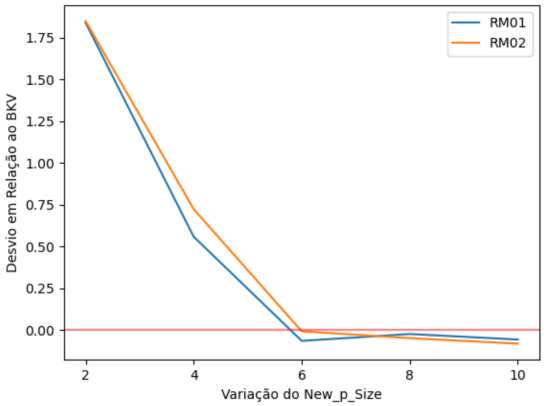
É possível observar que ocorre a diminuição do desvio em relação ao BKV com o aumento do número de gerações sem melhora,e estabilizando no valor 30,inclusive com a média de soluções melhores que o BKV.

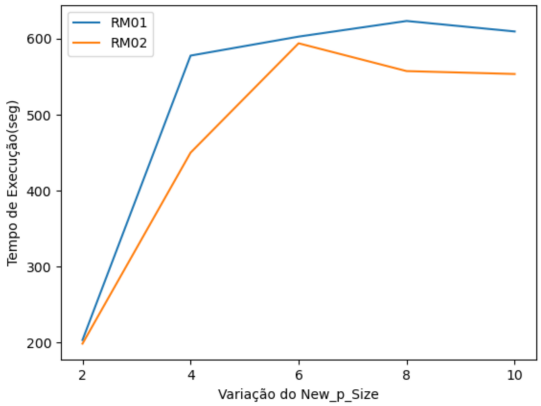
O tempo de execução aumenta de forma quase linear com o aumento desse número,pois o número de iterações do algoritmo também aumenta.

O valor 30 foi escolhido para configuração final, pois é o ponto de estabilização do desvio, assim como possui menor tempo que valores maiores.

5.5 Teste do parâmetro New_p_Size

Para este teste foi analisado o parâmetro New_p_Size variando de 2 a 10 com incrementos de 2.O teste também foi feito utilizando a configuração inicial.





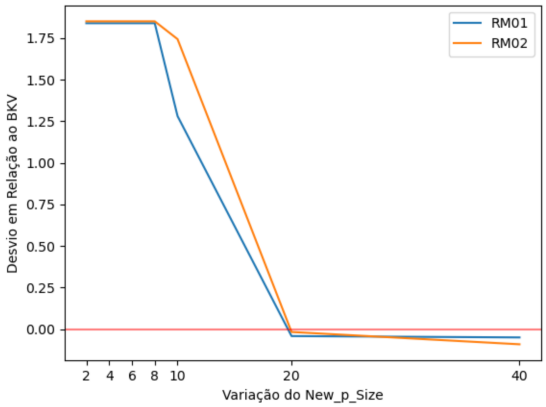
Percebe-se que o aumento de novos indivíduos gerados por geração também melhora a média da solução final, encontrando valores médios melhores que o BKV a partir do valor 6.

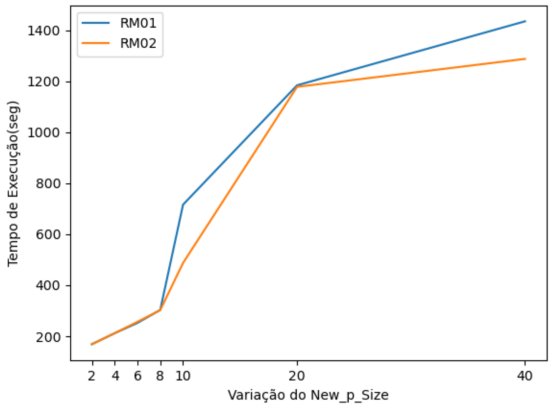
Também é possível perceber que o tempo de execução cresce até aproximadamente o valor 6 com tempos parecidos com valores maiores que 6.

O valor 10 foi escolhido para configuração final para tentar maximizar as soluções finais , pois foi o valor que achou as melhores soluções e possui tempo parecido com os valores 6 e 8 que também mostraram-se bons valores.

5.6 Teste do parâmetro New_p_Size com mutação baixa

Neste teste foi analisado o que acontece se variarmos o parâmetro New_p_Size, porém com mutação baixa dessa vez. A configuração para o teste é a seguinte: Population_Size=40,Mutation_Rate=0.3, Max_Non_Improving_Gen=30,Max_Time=1800(seg) e New_p_Size variando entre os valores {2,4,6,8,10,20,40}.





Através dos resultados, podemos ver que também é possível chegar em soluções médias melhores que o BKV com a taxa de mutação baixa, porém o tempo de execução aproximadamente dobra em relação a configuração com taxa de mutação=0.9, pois soluções equivalentes só são encontradas a partir de 20 novos indivíduos gerados a cada geração, fazendo assim que muitos indivíduos necessitem ser refactibilizados a cada iteração do algoritmo.

6 Teste de Instâncias

6.1 Algoritmo Genético

Os testes foram executados com a seguinte configuração gerada após as análises da seção anterior: Population_Size = 40,New_p_Size=10, Mutation_Rate =0.9 e Max_Non_Improving_Gen = 30.

Nesta seção foram analisados os valores finais obtidos em relação ao BKV.O cálculo do desvio manteve a lógica descrito na seção 5, portanto menores desvios significam que melhores valores foram encontrados.

A tabela abaixo mostra os resultados obtidos:

Instância	Média do Tempo de Execução (segundos)	Valor Médio da Solução Final (SF)	Desvio da Solução em Relação ao BKV (%)	Melhor Valor Obtido na Solução Final (SF)	Desvio do Melhor Valor em Relação ao BKV (%)
RM01	609	2437	-0.041	2439	-0.123
RM02	553	2434	-0.082	2437	-0.205
RM03	656	2436	0	2436	0
RM04	590	2440	-0.041	2445	-0.246
RM05	647	2436	0	2437	-0.041
RM06	502	2435	-0.164	2439	-0.329
RM07	612	2439	-0.082	2440	-0.123
RM08	540	2435	-0.082	2436	-0.123
RM09	560	2436	0	2437	-0.041
RM10	471	2432	-0.123	2437	-0.329
RM11	575	2434	0	2434	0
RM12	615	2440	0	2441	-0.040

6.2 Algoritmo Genético-Parte 2

Nesta seção foram analisados os valores finais obtidos em relação aos valores obtidos na solução inicial. O desvio foi calculado da seguinte maneira:

$$\frac{SI - SF}{SI} \cdot 100$$

A tabela abaixo mostra os resultados obtidos:

Instância	**Valor Médio da Solução Inicial (SI)**	**Valor Médio da Solução Final (SF)**	**Desvio de SF em Relação ao SI (%)**
RM01	2391	2437	-1.923
RM02	2387	2434	-1.968
RM03	2388	2436	-2.010
RM04	2391	2440	-2.049
RM05	2390	2436	-1.923
RM06	2390	2435	-1.882
RM07	2388	2439	-2.135
RM08	2388	2435	-1.968
RM09	2389	2436	-1.967
RM10	2390	2432	-1.757
RM11	2391	2434	-1.798
RM12	2390	2440	-2.092

7 Conclusão

Levando em consideração os testes das seções anteriores, pode-se concluir que o algoritmo genético é uma opção muito viável para solucionar o problema do emparelhamento diversificado, pois para as todas 12 instâncias de teste, as médias dos valores ficaram pelo menos iguais ao BKV, e maiores que o BKV para 7 das 12 instâncias, com a média de tempo de execução de aproximadamente 10 minutos.

Analisando os melhores resultados, também observa-se que estes foram encontrados com o taxa de mutação de valor 0.9, o que indica que as melhores soluções foram encontradas quando o algoritmo se comportou como uma busca aleatória com elementos e operadores do algoritmo genético.Porém, na seção 5.6 foi observado que resultados semelhantes também podem ser obtidos com mutação baixa, com o prejuízo do aumento do tempo de execução.

Em virtude dos aspectos abordados, considera-se que se conseguiu encontrar uma boa abordagem para o problema, visto que resultados melhores que o BKV podem ser obtidos com múltiplas configurações.Entretanto, melhorias poderiam ser feitas para diminuir o tempo de execução e do impacto da aleatoriedade no algoritmo, principalmente na função de refactibilização do cromossomo.

Referências

[1] Siew Mooi Lim, Abu Bakar Md. Sultan, Md. Nasir Sulaiman, Aida Mustapha, and K. Y. Leong. *Crossover and Mutation Operators of Genetic Algorithms*. International Journal of Machine Learning and Computing, February 2017, Volume: 07, No: 01

[2] Reeves C.R. (2010) *Genetic Algorithms. In: Gendreau M., Potvin JY. (eds) Handbook of Metaheuristics.* International Series in Operations Research & Management Science, vol 146. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1665-5_5