Trabalho
Meta-Heurísticas

Professor:

Marcus Rolf Peter Ritt

Integrantes:

Bruno Grohs Vergara (00324256)
Erick Larratéa Knoblich (00324422)

Universidade Federal do Rio Grande do Sul
Bacharelado em Ciência da Computação
INF-05010 Otimização Combinatória

1 Introdução

O objetivo deste trabalho foi implementar uma meta-heurística para resolver o problema Planejamento para uma Escola, sendo que a meta-heurística escolhida foi um algoritmo genético.

O problema Planejamento para uma Escola é: dado um conjunto n de professores cada um com um salário s_i tal que $i \in [n]$, um conjunto de m alunos cada um com uma demanda de atendimento em horas h_j tal que $j \in [m]$, um conjunto $P(j) \subseteq [n]$ de professores que possivelmente podem atender o aluno j e um número mínimo de horas H, temos que a solução é uma seleção de professores $\mathcal{P} \subseteq [n]$ e uma seleção de alunos $\mathcal{A} \subseteq [m]$ tal que para cada aluno $j \in \mathcal{A}$ pelo menos um professor em $P(j)$ é selecionado, a soma das demandas de atendimento em horas de cada aluno $j \in \mathcal{A}$ seja no mínimo H e a soma dos salários de cada professor $i \in \mathcal{P}$ seja minimizada.

2 Formulação do Programa Inteiro

- Um conjunto de n professores.
- $\mathcal{P} \subseteq [n]$: a seleção de professores.
- s_i, $i \in [n]$: o salário do professor i.
- $x_i \in \{0, 1\}$, $i \in [n]$: as variáveis binárias que representam se o professor i do conjunto de professores foi selecionado para o conjunto $\mathcal{P}$.
- Um conjunto de m alunos.
- $\mathcal{A} \subseteq [m]$: a seleção de alunos.
- h_j, $j \in [m]$: a demanda em horas do aluno j.
- H: a demanda total em horas que precisa ser atingida.
- $y_i \in \{0, 1\}$, $y \in [m]$: as variáveis binárias que representam se o aluno j do conjunto m foi selecionado para o conjunto $\mathcal{A}$.
- $P(j) \subseteq [n]$: o conjunto de professores que possivelmente podem atender o aluno j, sendo que para cada aluno $j \in \mathcal{A}$ pelo menos um professor em $P(j)$ é selecionado.

Função Objetivo:

$$min \sum_{i \in [n]} s_i \, x_i$$

Restrições:

$$\sum_{j \in [m]} h_j \, y_j \geq H \tag{1}$$

$$y_j \leq \sum_{i \in P(j)} x_i \qquad \forall j \in \mathcal{A} \tag{2}$$

- A função objetivo minimiza a soma dos salários s_i de todos os professores selecionados $i \in \mathcal{P}$ usando as variáveis binárias x_i dos professores como parâmetro de seleção.
- A primeira restrição restringe a soma das demandas de atendimento em horas h_j de todos os alunos selecionados $j \in \mathcal{A}$ para que ela seja maior ou igual à H.
- A segunda restrição impossibilita que a demanda de atendimento em horas de um aluno qualquer selecionado seja considerada sem que pelo menos um professor possa atendê-lo, sendo que sua demanda de atendimento em horas é dividida entre os professores, ou seja, se dois ou mais professores atendem-no, a sua demanda de atendimento em horas não é considerada mais de uma vez.

3 Algoritmo Genético

Um algoritmo genético consiste de alguns conceitos que compõem sua estrutura:

- Cromossomo: Um cromossomo representa uma solução, seja ela viável ou inviável, representada da forma $c = (c_1, c_2, c_3, ..., c_n)$, onde os valores $c_i \in \{0, 1\}$ e cada valor indica se o professor i é utilizado ou não na solução.
- Gene: Um gene representa um único valor c_i em um cromossomo $c = (c_1, c_2, c_3, ..., c_n)$.
- População: Uma população é composta por um conjunto de cromossomos. Podemos referir cada nova população como uma geração em algumas situações.
- *Crossover*: O *crossover* é realizado ao selecionar dois cromossomos e realizar a troca de genes entre os dois. O método de *crossover* é discutido na seção 5.4.
- Seleção: A seleção significa selecionar dois cromossomos de uma população para realizar *crossover*. O método de seleção é discutido na seção 5.3.
- Mutação: Uma mutação consiste em alterar o valor de um ou mais genes em um cromossomo. O método de mutação é discutido na seção 5.5.

4 Parâmetros do Algoritmo

Para a execução do algoritmo, possuímos alguns parâmetros previamente determinados para definir alguns fatores do algoritmo. Primeiramente, o parâmetro *populationSize* determina o tamanho que uma população possui, ou seja, quantos cromossomos compõem uma população. Em seguida temos os parâmetros *crossoverRate* e *mutationRate*, que determinam as taxas de *crossover* e mutação. Por último temos o parâmetro *generationWithoutInprovement*, que indica o máximo de gerações em sequência sem melhora no melhor resultado, o que faz com que o algoritmo termine.

5 Pseudoalgoritmo

```
population ← GenerateInitialPopulation()
bestSolution ← GetBestSolution(population)
while generationWithoutInprovement < g do
    newPopulation ← ∅
    while length(newPopulation) < length(population) do
        c1, c2 = Selection(population)
        if random(0, 1) < crossoverRate then
            offspring = Crossover(c1, c2)
        else
            offspring = Best(c1, c2)
        end if
        if random(0, 1) < mutationRate then
            Mutation(offspring)
        end if
        if ValidateSolution(offspring) then
            newPopulation ∪ offspring
        end if
    end while
    population ← newPopulation
end while
```

O algoritmo consiste em gerar novas populações através da troca genética entre uma já existente população, além de pequenas mutações aplicadas à prole.

5.1 Geração da População Inicial

A população inicial é gerada aleatoriamente, porém todos os cromossomos criados são testados e verificados, caso não sejam válidos. O tamanho da população inicial varia, mas será sempre constante durante a execução do algoritmo.

5.2 Obtendo a Melhor Solução de uma população

Cada vez que uma nova população é gerada, é obtida a melhor solução dessa geração e comparada com a melhor solução geral. No início, a melhor solução da população inicial será considerada a melhor de todas inicialmente.

5.3 Seleção

A seleção de indivíduos de uma população para *crossover* é realizada por torneio. A seleção por torneio funciona da seguinte maneira: são selecionados aleatoriamente de 2 cromossomos a 25% do tamanho da população, cromossomos. Desses selecionados, são escolhidos os dois melhores, ou seja, os dois com menor valor de função objetivo. O valor de 25% foi escolhido após múltiplos testes.

5.4 *Crossover*

O tipo de *crossover* escolhido foi o *Single-point crossover*, onde é escolhido um ponto aleatoriamente de um cromossomo e então os pais são divididos em dois. Em sequência, de forma aleatória, é selecionada a parte à esquerda do corte do pai 1 ou 2 e a parte à direita do corte do outro pai. Abaixo um exemplo

Pai 1 = 011011-01001111110
Pai 2 = 011000-10110100101
Filho = 011000-01001111110

5.5 Mutação

Após a geração de um novo cromossomo, existe a chance de haver a mutação de alguns genes da prole. A mutação seleciona um gene do cromossomo aleatoriamente e inverte o seu valor. Abaixo um exemplo de mutação:

Filho Original = 01100001001111110
Filho Mutado = 01100001001101110

5.6 Critérios de Terminação

Os critérios de terminação escolhidos para determinar a parada do algoritmo são o número de gerações em sequência sem haver melhora na melhor solução encontrada e um critério de tempo percorrido. Em geral, devido a testes de variação no número máximo de gerações sem melhora, pudemos perceber que ao aumentar muito o valor não havia melhora significativa no objetivo, então o critério de tempo percorrido não foi frequentemente utilizado, mas quando utilizado, havia um limite de 30 minutos.

6 Plataformas de Implementação e Linguagem

O algoritmo foi implementado e executado em duas máquinas pessoais distintas, uma com Windows 11 Pro com processador Intel(R) Core(TM) i7-12700 2.1GHz e 16GB de memória e outra com sistema operacional macOS Ventura 13.4.1, processador Apple M2 e memória de 8GB.

A linguagem de programação utilizada para implementar o algoritmo foi Python 3.11.4 64-bit.

7 Testes das Instâncias

Nesses testes, foram testadas as instâncias ppue01, ppue02, ppue03, ..., ppue10, disponibilizadas pelo professor. Em todos os testes, foi utilizado um mesmo número como *seed* (-5508469125140711083), gerado previamente aos testes de maneira aleatória. A tabela abaixo mostra os valores de n e m para todas as instâncias, o melhor resultado conhecido e o resultado da solução inicial, que é composta por 1 e 0 alternantes, ou seja, SI = $(1, 0, 1, 0, 1, 0, ...)$.

Instância	n	m	Melhor Solução	Solução Inicial
ppue01	100	100000	77	3052
ppue02	100	100000	76	2861
ppue03	100	10000	74	2706
ppue04	100	10000	102	2922
ppue05	100	1000	73	2676
ppue06	100	1000	69	2386
ppue07	100	50000	82	2637
ppue08	100	50000	77	2653
ppue09	100	5000	90	2685
ppue10	100	5000	84	2564

7.1 Teste com Altas Taxas de Mutação e *crossover*

Os testes abaixo foram realizados com $crossoverRate = 0.9$ e $mutationRate = 0.75$. O tamanho da população era 50 e o máximo de gerações sem melhora no objetivo foi igual a 100.

Instância	Melhor Solução	Solução Encontrada	Desvio para Melhor Solução (%)	Desvio para Solução Inicial (%)
ppue01	77	91	-18.18	97.01
ppue02	76	87	-14.47	96.95
ppue03	74	84	-13.51	96.89
ppue04	102	121	-18.62	95.85
ppue05	73	102	-39.72	96.18
ppue06	69	78	-13.04	96.73
ppue07	82	85	-3.65	96.77
ppue08	77	77	0.0	97.09
ppue09	90	94	-4.44	96.49
ppue10	84	103	-22.61	95.98

7.2 Teste com Baixas Taxas de Mutação

Os testes abaixo foram realizados com $crossoverRate = 0.9$ e $mutationRate = 0.3$. O tamanho da população era 50 e o máximo de gerações sem melhora no objetivo foi igual a 100.

Instância	Melhor Solução	Solução Encontrada	Desvio para Melhor Solução (%)	Desvio para Solução Inicial (%)
ppue01	77	225	-192.20	92.62
ppue02	76	102	-34.21	96.43
ppue03	74	154	-108.10	94.30
ppue04	102	154	-50.98	94.72
ppue05	73	139	-90.41	94.80
ppue06	69	86	-24.63	96.39
ppue07	82	135	-64.63	94.88
ppue08	77	124	-61.03	95.32
ppue09	90	172	-91.11	93.59
ppue10	84	216	-157.14	91.57

7.3 Teste com Baixas Taxas de *Crossover*

Os testes abaixo foram realizados com $crossoverRate = 0.4$ e $mutationRate = 0.75$. O tamanho da população era 50 e o máximo de gerações sem melhora no objetivo foi igual a 100.

Instância	Melhor Solução	Solução Encontrada	Desvio para Melhor Solução (%)	Desvio para Solução Inicial (%)
ppue01	77	176	-128.57	94.23
ppue02	76	189	-148.68	93.39
ppue03	74	129	-74.32	95.23
ppue04	102	226	-121.56	92.26
ppue05	73	215	-194.52	91.96
ppue06	69	161	-133.33	93.25
ppue07	82	207	-152.43	92.15
ppue08	77	203	-163.63	92.34
ppue09	90	214	-137.77	92.02
ppue10	84	227	-170.23	91.14

7.4 Teste com Baixas Taxas de Mutação e *Crossover*

Os testes abaixo foram realizados com $crossoverRate = 0.4$ e $mutationRate = 0.3$. O tamanho da população era 50 e o máximo de gerações sem melhora no objetivo foi igual a 100.

Instância	Melhor Solução	Solução Encontrada	Desvio para Melhor Solução (%)	Desvio para Solução Inicial (%)
ppue01	77	92	-19.48	96.98
ppue02	76	80	-5.26	97.20
ppue03	74	87	-17.56	96.78
ppue04	102	114	-11.76	96.09
ppue05	73	121	-65.75	95.47
ppue06	69	74	-7.24	96.89
ppue07	82	101	-23.17	96.16
ppue08	77	78	-1.29	97.05
ppue09	90	101	-12.22	96.23
ppue10	84	110	-30.95	95.70

7.5 Teste com Solver Gekko

Foram realizados testes utilizando a biblioteca Gekko, de Python. Ela monta problemas lineares e inteiros e os soluciona. Foi imposto um limite de no máximo 30 minutos.

Instância	Melhor Solução	Solução Encontrada
ppue01	77	–
ppue02	76	–
ppue03	74	–
ppue04	102	–
ppue05	73	73
ppue06	69	69
ppue07	82	–
ppue08	77	–
ppue09	90	–
ppue10	84	–

A resolução dessas instâncias acaba por ser inviável devido ao grande número de alunos em algumas delas e, por consequência, o enorme número de restrições. Nos casos em que não foi obtido resultado, muitos passaram o limite de restrições que a biblioteca suportava.

8 Conclusão

É perceptível com os resultados dos testes que as menores taxas de mutação e *crossover* influenciam diretamente a encontrarmos soluções piores. Variando-as individualmente para valores menores que 0.5, obtemos resultados notavelmente piores, mostrando que a variabilidade genética no caso desse problema é extremamente importante para a melhor otimização dele. Ao executar com ambas as taxas com valores mais baixos, obtemos resultados muito voláteis, que as vezes são melhores que ao testar com taxas altas, mas em outras vezes são significativamente piores. Outro fator perceptível foi que o tempo para que os testes terminassem quando as taxas de mutação e *crossover* eram menores foi significativamente maior, o que mostra mais uma vantagem de se utilizar mais mutação e *crossover*.

9 Referências

- Referências sobre algoritmos genéticos: `https://www.ijcse.com/docs/IJCSE10-01-03-29.pdf`
- Documentação da biblioteca Gekko: `https://gekko.readthedocs.io`
- Referências em Single-point crossover `https://medium.com/@samiran.bera/crossover-operator-the-heart-of-genetic-algorithm-6c0fdcb405c0`