CONGRESSO CATARINENSE DE CIÊNCIA DA COMPUTAÇÃO
2022. Ano 9, Vol. 9. ISSN: 2319-0655

ALGORITMO GENÉTICO PARA MONTAGEM DE GRADE HORÁRIA DE AULAS

Carlos Henrique Pinheiro Cordeiro
Estudante do curso de Bacharelado em Ciência da Computação, do Instituto Federal Catarinense de Rio do Sul, SC.
carloshenpincord@gmail.com

Abstract. *This work aims to bring, commented and abstracted, implementation of the artificial intelligence technique named genetic algorithm, on the objective of elaborating a weekly timetable for offering classes, to different disciplines. As well as presenting your results.*
Key-words: *Chromosome; Discipline; Grid.*

Resumo. *Este trabalho visa trazer implementação comentada e abstraída, da técnica de inteligência artificial chamada de algoritmo genético, com o objetivo de elaborar uma grade horária semanal para oferta de aulas, para diferentes disciplinas. Bem como também apresentar os seus resultados.*
Palavras-chave: *Cromossomo; Disciplina; Grade.*

1. Introdução

Dada a evolução dos estudos dentro da Inteligência Artificial, em especial os avanços sobre a técnica dos Algoritmos Genéticos, este trabalho possui o objetivo de trazer uma implementação deste, comentada e discutida, que realize montagem de uma grade horária, de aulas sobre diferentes disciplinas com disponibilidades diferentes, ao longo de uma semana.

2. Algoritmo Genético

Sobre os algoritmos genéticos em si, estes consistem em, conforme Pacheco (1999), uma técnica de otimização e busca altamente paralela, com inspiração no princípio Darwiniano da seleção natural e reprodução genética. Onde tal se estabelece em privilegiar os indivíduos aptos, relativamente ao ambiente em que estes se encontram, com maior longevidade, portanto possuindo maior probabilidade de reprodução. Portanto, garantindo-os grandes chances de perpetuar seus códigos genéticos, conferindo identidade, presente nos cromossomos, para os novos indivíduos das próximas gerações.

Sendo assim, estes princípios acabam por ser imitados durante a construção destes algoritmos, buscando a melhor solução para um problema estabelecido. Através da mesma ideia de evolução das populações e soluções apresentadas por estas, desenvolvidos e caracterizados por cromossomos artificiais. Consistindo em uma

estrutura de dados que representa uma das possíveis soluções para a problemática proposta.

Conforme os princípios citados, estes mesmos cromossomos estão sujeitos aos processos que envolvem a evolução, como a seleção de certos destes para reprodução por compartilhamento de material genético e mutação. Por fim, conforme as gerações vão correndo e vários ciclos de reprodução ocorrem, a população deverá apresentar indivíduos mais aptos - melhores soluções - para resolver o problema estabelecido.

3. Aplicação

Nas subseções seguintes, serão discutidos os pilares da aplicação do algoritmo proposto, estabelecendo a abstração do seu funcionamento, para que se atinja o objetivo proposto, discutido na seção de Testes posteriormente. Segue link para repositório online onde se encontra o projeto, desenvolvido em Python em um Jupyter Notebook: https://github.com/CarlosHenriquePinheiroCordeiro/algoritmo-genetico-grade-aula.

3.1. Representação Cromossômica

A representação cromossômica do problema foi abstraída como uma representação de um horário de aulas. Representada, algoritmicamente, como um vetor de seis posições. A primeira posição guarda o valor heurístico de *fitness* atribuído a esta solução proposta. Os cinco restantes representam cada dia da semana, de segunda-feira até sexta-feira.

Estes mesmos também foram abstraídos como vetores, de até no máximo duas posições, indicando que por dia letivo, com espaço para no máximo cinco aulas, podem ser ministradas até duas disciplinas. E essas, também são *arrays*, de três *slots*, com o primeiro guardando o nome da matéria em questão, o segundo seu código identificador, e por terceiro a quantidade de aulas por semana que esta deve ser ministrada, para que se cumpra sua suposta carga horária estabelecida.

```
horario = [
    5,      #VALOR FITNESS FICTÍCIO
    [
        #objetoDeMateria5  #FECHOU AS CINCO MATÉRIAS DO DIA (5) - Segunda-feira
    ],
    [
        #objetoDeMateria2, #FECHOU AS CINCO MATÉRIAS DO DIA (2+3) - Terça-feira
        #objetoDeMateria3
    ],
    [
            #Quarta-feira
    ],
    [
            #Quinta-feira
    ],
    [
            #Sexta-feira
    ]
]
```

Figura 1 - Abstração de uma grade horária.

```
#Matérias
m1 = ['Inteligência Artificial'  , 1, 5]
m2 = ['Paradigmas de Programação', 2, 2]
m3 = ['Métodos Numéricos'        , 3, 3]
m4 = ['Matemática Discreta'      , 4, 5]
m5 = ['Sistemas de Informação'   , 5, 3]
m6 = ['Empreendedorismo'         , 6, 2]
m7 = ['Engenharia de Software'   , 7, 5]
materias = [m1, m2, m3, m4, m5, m6, m7]
```

Figura 2 - Abstração das matérias no algoritmo.

3.2. Função de Avaliação

A função de avaliação, ou *fitness* dentro da aplicação, atribui, ao horário enviado como parâmetro para ela, um valor heurístico real. No caso desta aplicação, quanto mais próximo de cinco este for, melhor. Vale constar que, geralmente, estes valores inicialmente se encontram nos números negativos, e absolutamente, por conta do funcionamento da aplicação, este nunca será superior a cinco.

Esta heurística, tomando como base o argumento em questão, é determinada pela quantidade de dias da semana que estão completos referente à carga horária (seja isso K), ou seja todas as cinco aulas do dia estão preenchidas, subtraída pela quantidade de disciplinas que estão se repetindo ao longo da semana (seja L), o que impede todas as matérias de serem ofertadas.Sendo assim, *fitness* = K - L. As demais funções auxiliares a este cálculo podem ser encontradas no repositório do projeto.

```
#Função de fitness que retorna um valor heurístico para o cromossomo, representando o quanto este converge para o resultado ideal.
#Caso um cromossomo alcance valor 5, este é o resultado ideal
def fitness(grade):
    diasAulas = [quantidadeAulasDia(grade[1]), quantidadeAulasDia(grade[2]), quantidadeAulasDia(grade[3]), quantidadeAulasDia(grade[4]), quantidadeAulasDia(grade[5])]
    return diasCompletosAulas(diasAulas) - disciplinasRepetidas(grade)
```

Figura 3 - Função de avaliação do algoritmo.

3.3. Operador Genético

O operador genético adotado, referente ao compartilhamento/troca do material genético entre os cromossomos, foi o *crossover* orientado a um ponto. No algoritmo, este define uma posição onde duas grades de aula trocarão seus dias, contendo as suas aulas, com a outra. Por exemplo, a grade A troca a sua segunda-feira com a da grade B. Vale constar que a escolha dos candidatos para a reprodução se dá por meio da técnica de Roleta Viciada, conferindo maior chance das melhores soluções serem selecionadas para prosseguir com a evolução do algoritmo.

```
def compartilhaMaterialGenetico(cromossomo, cromossomoParceiro, pontoCrossover):
    novoCromossomo = []
    for i in cromossomo[:pontoCrossover]:
        novoCromossomo.append(i)
    for i in cromossomoParceiro[pontoCrossover:]:
        novoCromossomo.append(i)
    return novoCromossomo

def crossover(cromossomoI, cromossomoII):
    pontoCrossover = ceil(random.uniform(0.0, 5.0))
    return (compartilhaMaterialGenetico(cromossomoI, cromossomoII, pontoCrossover), compartilhaMaterialGenetico(cromossomoII, cromossomoI, pontoCrossover))
```

Figura 4 - Funções referente ao *crossover*.

```
def roleta(cromossomos):
    roletaViciada = []
    for cromossomo in cromossomos:
        vicia = 1
        if cromossomo[0] > 0:
            vicia = cromossomo[0] * 3
        for i in range(vicia):
            roletaViciada.append(cromossomo)
    sorteado = ceil(random.uniform(0.0, float(len(roletaViciada)))) - 1
    if sorteado < 0:
        sorteado = 0
    return roletaViciada[sorteado]
```

Figura 5 - Roleta viciada para seleção de um candidato para reprodução.

3.4. Mutação

Agora, no tocante ao fator de mutação dos cromossomos, este foi implementado de forma que, dois dias de aula, dentro de uma grade horária, troquem alguma matéria sua com uma de outro dia. Por exemplo, na grade A, a terça-feira troque sua primeira aula com a de quarta-feira. Foi adotada uma chance de mutação a 0.1%, visto que esta se encontra na medida certa para ajudar o algoritmo a convergir para o resultado. Se esta chance cresce, atrapalha, se diminuir, não ajuda.

```python
def mutacao(cromossomo):
    novoCromossomo = cromossomo
    if random.random() * 100 <= float(chanceMutacao):
        diaI       = ceil(random.uniform(1.0, 5.0))
        diaII      = ceil(random.uniform(1.0, 5.0))
        iMateria   = ceil(random.uniform(0.0, len(novoCromossomo[diaI])))  - 1
        iMateriaII = ceil(random.uniform(0.0, len(novoCromossomo[diaII]))) - 1
        if (iMateria < 0):
            iMateria = 0
        if (iMateriaII < 0):
            iMateriaII = 0
        materiaI  = novoCromossomo[diaI][iMateria]
        materiaII = novoCromossomo[diaII][iMateriaII]
        novoCromossomo[diaI][iMateria]    = materiaII
        novoCromossomo[diaII][iMateriaII] = materiaI
    return novoCromossomo
```

Figura 5 - Função da aplicação de mutação do cromossomo.

4. Testes

Agora, serão demonstrados testes com três casos de parâmetros diferentes, sendo a quantidade de gerações e quantidade de indivíduos para uma população inicial. Vale constar que, o melhor resultado é uma grade horária que todos os dias sejam ofertadas matérias diferentes entre si, sem repetição, e que sejam ocupados todos os cinco horários de cada dia. Seguem os casos de teste:

1. 200 Gerações, 10 indivíduos.
2. 500 Gerações, 15 indivíduos.
3. 1000 Gerações, 17 indivíduos.

Por fim, são demonstrados os resultados exibidos pelo algoritmo, e o do fluxo principal que a aplicação executa.

```python
            if solucao[0] > melhorFitness:
                melhorSolucao = solucao
                melhorFitness = solucao[0]
        printResultado(melhorSolucao)
```

```
✓ 0.1s
Melhor horário encontrado:
Segunda-Feira - Matemática Discreta - 5.
Terça-Feira   - Sistemas de Informação - 3.
Quarta-Feira  - Paradigmas de Programação - 2; Empreendedorismo - 2.
Quinta-Feira  - Engenharia de Software - 5.
Sexta-Feira   - Inteligência Artificial - 5.
```

Figura 6 - Testes caso 1.

```python
                melhorFitness = solucao[0]
        printResultado(melhorSolucao)
```

```
✓ 0.2s
Melhor horário encontrado:
Segunda-Feira - Sistemas de Informação - 3; Paradigmas de Programação - 2.
Terça-Feira   - Inteligência Artificial - 5.
Quarta-Feira  - Engenharia de Software - 5.
Quinta-Feira  - Matemática Discreta - 5.
Sexta-Feira   - Engenharia de Software - 5.
```

Figura 7 - Testes caso 2.

```
                melhorSolucao = solucao
                melhorFitness = solucao[0]
        printResultado(melhorSolucao)

✓ 0.4s

Melhor horário encontrado:
Segunda-Feira - Matemática Discreta - 5.
Terça-Feira   - Engenharia de Software - 5.
Quarta-Feira  - Sistemas de Informação - 3; Paradigmas de Programação - 2.
Quinta-Feira  - Métodos Numéricos - 3; Empreendedorismo - 2.
Sexta-Feira   - Inteligência Artificial - 5.
```

Figura 8 - Testes caso 3.

```
populacao = populacaoInicial()
geracoes  = 200
solucoes  = []
achou     = False
for geracao in range(geracoes):
    if achou:
        break
    for horario in populacao:
        horario[0] = fitness(horario)
        if horario[0] == 5:
            printResultado(horario)
            achou = True
            break
        solucoes.append(horario)
    novaPopulacao = []
    for i in range(quantidadePopulacaoInicial):
        candidatoI  = roleta(populacao)
        candidatoII = roleta(populacao)
        novoHorarioI, novoHorarioII = crossover(candidatoI, candidatoII)
        novaPopulacao.append(mutacao(novoHorarioI))
        novaPopulacao.append(mutacao(novoHorarioII))
    populacao = novaPopulacao

if not achou:
    melhorSolucao = []
    melhorFitness = -7
    for solucao in solucoes:
        if solucao[0] > melhorFitness:
            melhorSolucao = solucao
            melhorFitness = solucao[0]
    printResultado(melhorSolucao)
```

Figura 9 - Fluxo da aplicação.

5. Conclusão

Dados os resultados dos testes sobre diferentes circunstâncias no contexto estabelecido (quantidade de matérias por dia e afins), referente aos valores das variáveis iniciais, pode-se concluir que, conforme o algoritmo desenvolvido atualmente, é necessário uma grande quantidade de gerações com, relativamente, muitos indivíduos para uma população inicial, para que o programa convirja para um resultado final aceitável e aplicável, exercendo seu objetivo proposto. Assim, mostrando que se encontra sujeito à melhorias, mesmo que com um tempo de execução bastante rápido. Por fim, concluindo a proposição do trabalho de exposição explicativa sobre a implementação proposta.

Referências

PACHECO, Marco Aurélio Cavalcanti. **Algoritmos Genéticos: Princípios e Aplicações**. Disponível em: <http://www.inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/MQ-intro_apost.pdf>.