Algoritmo Genético Para o Problema de Atribuição de Tarefas a Máquinas

João Pedro Ferreira (324521)
Ricardo dos Santos Brandão (312958)

Abril de 2023

1 Introdução

Este trabalho tem como objetivo implementar a meta-heurística genética para resolver o problema de atribuição de tarefas a máquinas, com mais informações na seção 3. O problema de atribuição de n tarefas a máquinas (ATM) requer a alocação de tarefas em um conjunto de máquinas, levando em conta suas capacidades individuais, com o objetivo de minimizar a quantidade necessária de máquinas. Cada tarefa é definida por uma tupla (s, f, c), onde s representa o tempo de início da tarefa, f é o tempo de término e c é o custo associado. Para garantir que uma máquina não exceda sua capacidade, a soma dos custos das tarefas alocadas a ela deve ser menor ou igual a C.

2 Formulação como Problema Inteiro

Variáveis:

$$Yij \in \{0,1\},\ X1i \in \{0,1\},\ X2i \in \{0,1\}, Zi \in R\ \forall\, i,j \in \{1,n\}$$

Função Objetivo:

$$Min \sum_{i=1}^{n} X2i$$

Restrições:

Cada tarefa é atribuída a uma máquina:

$$Min \sum_{j=1}^{n} Yij = 1, \qquad \forall\, i \in \{1,n\}$$

Restrição de cores por tempo:

$$\sum_{j=1}^{n} cj * Yij * (0\ se\ sj \leq k \leq fj\ senão\ 1) \leq C\ \forall\, i \in \{1,n\} \forall k \in \{0, \max(t)\}$$

Restrições para contar o número de máquinas usadas:

$$\sum_{i=1}^{n} Yij = Zi\ \forall\, i\ \{1,n\}$$

$$Zi \leq Xi * C\ \forall\, i\ \{1,n\}$$

$$Zi + X1i * C \leq C\ \forall\, i \in \{1,n\}$$

$$X1i + X2i = 1\ \forall\, i \in \{1,n\}$$

Número de cores utilizadas em uma tarefa deve ser menor que C:

$$ci \leq C\ \forall\, i \in \{1,n\}$$

3 Algoritmo Genético

3.1 Inicialização

A geração da população inicial foi feita de maneira aleatória.

3.2 Avaliação

A qualidade (fitness) do indivíduo na população é avaliada levando em conta o número de máquinas usadas na solução.

3.3 Crossover

Para o crossover, são selecionados os indivíduos com maior fitness na população. Uma vez escolhidos, aplicamos o processo de recombinação em um ponto. Nele, uma posição é escolhida aleatoriamente ao longo da cadeia de genes das soluções pai. A partir dessa posição, os genes de uma solução são combinados com os genes de outra solução produzindo um par de filhos. Haja vista que uma solução precisa incluir os índices de todas as tarefas enquanto preserva a unicidade dessas, aplicamos uma etapa de remoção de índices duplicados garantindo a factibilidade da solução.

3.4 Mutação

Cada filho gerado pelo crossover possui uma probabilidade de sofrer um processo de mutação. Na mutação, o cromossomo do indivíduo tem dez por cento de suas posições trocadas aleatoriamente.

3.5 Substituição

Para a substituição, utilizamos o método de elitismo em que os indivíduos com maior qualidade são preservados.

3.6 Critério de Parada

O critério de parada escolhido foi o de número de gerações.

4 Implementação

O algoritmo genético foi escrito em Python utilizando a biblioteca NumPy, escolhida a fim de garantir maior eficiência nas operações numéricas em matrizes. Para o programa inteiro utilizamos como solver a biblioteca Cbc da linguagem Julia.

4.1 Estrutura de Dados

A população é representada por uma fila de prioridade, sendo a prioridade dos indivíduos definida pela função de avaliação. Cada indíviduo é representado por um array de números que representam os índices de cada tarefa, sendo que a ordem dos índices representa a ordem de atribuição das tarefas a máquinas.

5 Teste das Instâncias

5.1 Execução com o Solver CBC

Resultados gerados utilizando o solver. O critério de parada considerado foi o tempo de uma hora para a duração.

Instância do problema	Resultado Ótimo	Resultado final encontrado
50_100_1	19	20.0
50_100_2	10	11.0
100_100_1	18	21.0
100_100_2	9	10.0
150_100_1	24	39.0
150_100_2	10	31.0
200_100_1	25	não encontra solução
200_100_2	29	54.0

5.2 Execução com a Meta Heurística Genética

Resultados gerados utilizando a meta heurística. O critério de parada utilizado foi a passagem de 100 gerações.

Instância	Tempo (em segundos)	Geração do Resultado Encontrado	Resultado Encontrado	Resultado Ótimo
50_100_1	45	7	20	19
50_100_2	27	2	11	10
100_100_1	98	16	20	18
100_100_2	51	15	10	9
150_100_1	272	28	26	24
150_100_2	115	40	11	10
200_100_1	535	8	28	25
200_100_2	520	34	30	29

6 Conclusões

Devido às características do problema e à forma como representamos as soluções, não é viável avaliar a qualidade das soluções de maneira contínua. Consequentemente, pode haver uma série de soluções com a mesma avaliação, mas que diferem na quantidade de recursos utilizados. Isso pode levar à dominação da população por indivíduos com avaliações semelhantes, mas que representam soluções com pequenas variações, reduzindo a diversidade da população. Como resultado, o algoritmo pode ter dificuldade em distinguir entre soluções que estão mais próximas de uma solução melhor e a busca pode depender de

mutações aleatórias para encontrar soluções melhores. No entanto, a probabilidade de encontrar uma solução melhor através de uma mutação aleatória é baixa e aleatória.