Lista de exercícios 3

Lucas Emanuel Resck Domingues

15 de agosto de 2021

Problema: Suponha que temos n tarefas onde cada uma toma um tempo t_i para processar em m máquinas idênticas nas quais desejamos dizer qual a sequência de tarefas em cada máquina. Uma mesma tarefa não pode ser divida entre máquinas. Para um dado planejamento, A_j é o conjunto de tarefas colocados na máquina j. Seja $L_j = \sum_{i \in A_j} t_i$ o tempo da máquina j. O tempo de execução dessas tarefas será o maior L_j dentre as máquinas.

Exercício 1 (2.0 pontos) Proponha uma representação/estrutura de dados para o problema acima.

Solução: Podemos criar uma representação vetorial de tamanho n em que cada entrada i desse vetor corresponde à máquina para a qual a tarefa i foi alocada. Por exemplo, se temos $n = 3$ tarefas e $m = 2$ máquinas, uma possível representação seria $\boxed{0}\ \boxed{0}\ \boxed{1}$, indicando que as duas primeiras tarefas serão alocadas à máquina 0, e a última, à máquina 1.

Exercício 2 (2.0 pontos) Proponha uma heurística construtiva para o problema acima.

Solução: De forma análoga ao enunciado do Exercício 4 da Lista 2, ordenamos as n tarefas de forma que

$$t_1 \geq \cdots \geq t_n.$$

Na iteração i, alocamos a tarefa i à máquina que possui menor carga até agora, ou seja, alocamos a tarefa i à máquina $j = \arg\min_{1 \leq j \leq m} L_j$.

Exercício 3 (2.0 pontos) Proponha uma busca local para o problema acima.

Solução: Seja $r \in \{1, \cdots, n\}$ fixo. Desaloca-se, aleatoriamente, r tarefas de suas respectivas máquinas (isso equivale a, no vetor representação $\boxed{j_1}\boxed{j_2}\cdots\boxed{j_n}$, substituir r das entradas por um valor vazio). Dentre os possíveis planejamentos que contêm o planejamento dado pela representação atual, calcula-se aquele que tem o menor tempo de execução $\min L_j$. Se esse novo planejamento é de alguma forma melhor do que o planejamento da iteração anterior, atualiza-se a solução e repete-se esses passos; em caso negativo, finalizamos o procedimento com a solução anterior. veja o Pseudocódigo 1.

Algorithm 1 Busca local para a alocação de máquinas.

Require: $1 \leq r \leq n$
Require: Solução inicial $s_0 = \boxed{j_1}\boxed{j_2}\cdots\boxed{j_n}$

```
s_i ← s_0
while true do
    Desaloca r tarefas de s_i
    s ← o melhor planejamento possível baseado em s_i
    if o custo de s é menor do que o de s_i then
        s_i ← s
    else
        Break while loop
    end if
end while
return s_i
```

Exercício 4 (2.0 pontos) Como você estruturaria a lista restrita de candidatos da heurística construtiva?

Solução: Imagine que, na heurística construtiva do Exercício 2, ao invés de alocarmos t_i a alguma máquina na iteração i, introduzamos algum tipo de aleatoriedade. Podemos fazer da seguinte forma: na iteração i, das $n-i+1$ tarefas restantes, escolhemos as maiores $\lceil \alpha(n-i+1) \rceil$ para formar uma lista de candidatos, e aí sorteamos, nessa lista, aquela tarefa que será alocada. α é um parâmetro em $(0,1)$, que deve ser especificado de antemão.

Exercício 5 (2.0 pontos) Como você estruturaria os movimentos tabu para a busca local acima?

Solução: Podemos manter nossa "lista tabu" contendo as últimas soluções visitadas pelo algoritmo de busca local. Isso impede que, quando buscando na vizinhança de uma próxima iteração, o algoritmo volte à solução que originou essa iteração.

Uma outra forma de se estruturar os movimentos tabu é, ao invés de impedir o algoritmo de voltar a uma solução antiga, impedi-lo de desalocar ou alocar tarefas que foram desalocadas ou alocadas recentemente, a não ser que esse movimento gere uma solução melhor do que o melhor já encontrado anteriormente (critério clássico de aspiração).

Exercício 6 (2.0 pontos) Proponha uma estratégia de cruzamento e mutação para esse problema. Lembre-se a representação pode atrapalhar ou te ajudar muito na hora de fazer esse processo.

Solução: Lembremos que, no Exercício 1, propusemos a representação $\boxed{j_1}\boxed{j_2}\cdots\boxed{j_n}$, onde a entrada i do vetor significa que a tarefa i está alocada na máquina j_i. Dado isso, é bastante conveniente sugerir cruzamentos entre duas soluções. Podemos, por exemplo, utilizar um cruzamento clássico e pegar as r primeiras alocações de um dos pais e a restante do outro pai.

- Pais: $\boxed{j_1}\boxed{j_2}\cdots\boxed{j_n}$ e $\boxed{k_1}\boxed{k_2}\cdots\boxed{k_n}$
- Filhos: $\boxed{j_1}\cdots\boxed{j_r}\boxed{k_{r+1}}\cdots\boxed{k_n}$ e $\boxed{k_1}\cdots\boxed{k_r}\boxed{j_{r+1}}\cdots\boxed{j_n}$

Uma outra possibilidade, também bastante conveniente, é o cruzamento uniforme, onde o filho, para alocar a tarefa i, tem igual chance de escolher a máquina que aloca a tarefa i de qualquer um de seus pais.

Para a estratégia de mutação, podemos simplesmente trocar duas entradas do vetor, ou seja, duas alocações. Por exemplo:

$$\boxed{j_1}\cdots\boxed{j_r}\cdots\boxed{j_s}\cdots\boxed{j_n} \longrightarrow \boxed{j_1}\cdots\boxed{j_s}\cdots\boxed{j_r}\cdots\boxed{j_n}.$$

Outras estratégias, como colocar duas entradas do vetor juntas ou inverter a ordem do vetor entre duas entradas específicas (mutações possíveis para o problema do caixeiro viajante), por exemplo, acredito que não sejam adequadas aqui. Isso ocorre porque essas mutações envolvem inverter a ordem de entradas do vetor ou realizar um "shift" das posições do vetor, o que potencialmente altera drasticamente a solução. Por exemplo, se invertermos o vetor (entre duas entradas específicas), estamos potencialmente alterando todas as alocações de máquinas para essas tarefas que foram invertidas. Isso é muito mais drástico do que apenas inverter o caminho, como no caso do caixeiro viajante.