Sistemas Operativos
2021 / 2022

Licenciatura em Engenharia Informática

Trabalho Prático #1

Introdução

O Problema de Corte Unidimensional (*Cutting Stock Problem*) consiste na realização do corte de peças pequenas a partir de uma peça de tamanho maior. Este corte deverá respeitar um determinado padrão de forma a maximizar o número de peças cortadas e minimizar o desperdício.

Neste trabalho iremos assumir que as peças a cortar numa chapa de aço (denominado "placa"), são rectângulos ou quadrados cuja largura é a mesma da placa. Ou seja, será apenas necessário um corte (vertical) para obter a peça desejada (fig.1).

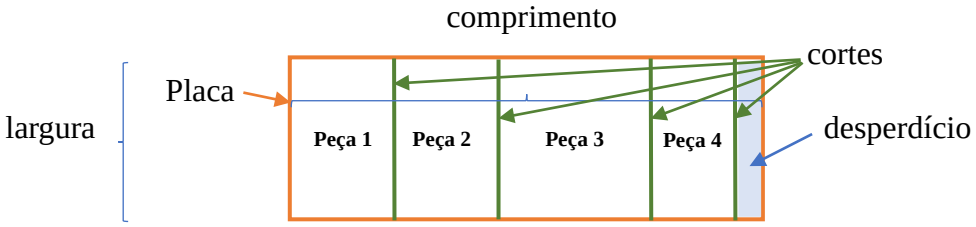


Figura 1. Exemplo de placa com peças a cortar e respetivo desperdício

Exemplo:

Sendo:

- *n* o número de padrões a ser gerados
- *m* o número de diferentes comprimentos de peças em que se pretende cortar cada uma das placas
- *maxComprimento* o comprimento da placa, em metros
- *compPecas* um vetor cujos elementos representam o comprimento (em metros) das peças a cortar
- *qtddPecas*, um vetor cujos elementos representam as quantidades de peças a cortar de cada comprimento definido no vetor *compPecas*, respeitando a mesma ordem

com os seguintes dados iniciais:

- *n* = 3
- *m* = 3
- *maxComprimento* = 9
- *compPecas* = [2.0, 3.0, 4.0]
- *qtddPecas* = [20, 10, 20]

Com esta informação conseguimos saber que se pretende gerar 3 padrões de organização das peças na placa, que a placa tem 9m de comprimento, que as peças a cortar possuem três comprimentos diferentes (2.0, 3.0 e 4.0 metros), e que pretendemos obter 20 peças de 2.0m, 10 peças de 3.0m, e 20 peças de 4.0m (fig. 2).

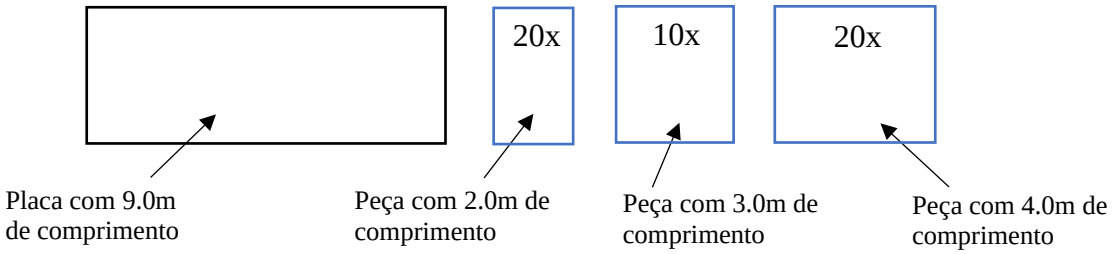


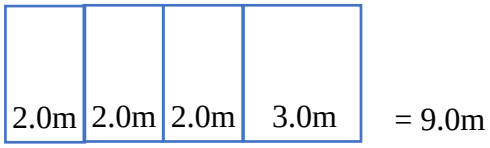
Figura 2. Placa e peças a cortar referentes ao exemplo

Para se calcularem os padrões que representam a organização das peças na placa, há que gerar uma matriz P (Padrões) para os representar, cuja dimensão é *m* x *n*, sendo *m* a dimensão do vetor *compPecas* (ou do vetor *qttdPecas*, que tem a mesma dimensão), como por exemplo:

$$P = \begin{bmatrix} 3 & 0 & 2 \\ 1 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

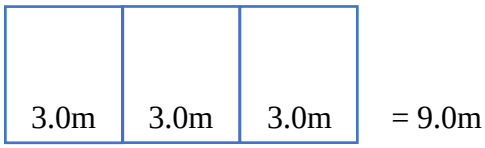
Esta matriz, gerada aleatoriamente, representa os 3 padrões, um padrão em cada coluna, sendo que as linhas representam cada tipo de peça. Assim sendo, a matriz *P* representa:

- Padrão da 1ª coluna: 3 peças de 2.0m + 1 peça de 3.0m + 0 peças de 4.0m

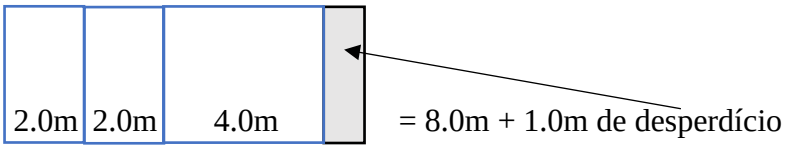


= 9.0m

- Padrão da 2ª coluna: 0 peças de 2.0m + 3 peças de 3.0m + 0 peças de 4.0m



- Padrão da 3ª coluna: 2 peças de 2.0m + 0 peças de 3.0m + 1 peça de 4.0m



A matriz P é gerada aleatoriamente, seguindo a geração do primeiro padrão, que se encontra na coluna 0, o seguinte raciocínio:

- Primeira posição do primeiro padrão (posição (0,0) da matriz): o valor calcula-se dividindo a dimensão do espaço disponível (todo o comprimento da placa), pelo comprimento da peça da primeira linha, neste caso temos 9m:2m = 4.5. Como a placas têm, neste caso, um comprimento de 2m, concluímos que, no máximo, cabem 4 placas com comprimento de 2.0m na placa. Gera-se um número aleatório entre 0 e 4 para representar o número de placas de 2.0m do primeiro padrão. No caso do exemplo, o número gerado foi o 3. Vamos ter 3 placas de 2.0m no primeiro padrão, ou seja, dos 9m da placa, 6m já estão ocupados com as 3 placas de 2.0m, restando 9m – 3·2m = 3m na placa.
- Segunda posição do primeiro padrão (posição (1,0) da matriz): dos 3m de placa que ainda está livre tem de se verificar quantas placas de 3.0m cabem. Seguindo o mesmo raciocínio que para a primeira linha, dividimos o espaço disponível pela dimensão da peça da segunda linha, neste caso temos 3m:3m = 1.0. Ou seja, podemos ter no máximo 1 peça de 3m no espaço disponível. Gera-se um número aleatório entre 0 e 1 para representar o número de placas de 3.0m do primeiro padrão. No caso do exemplo, o número gerado foi o 1. Vamos ter 1 placa de 3.0m no primeiro padrão, ou seja, já não temos mais espaço disponível na placa.
- Terceira posição do primeiro padrão (posição (2,0) da matriz): como já não temos espaço disponível, esta posição da matriz terá o valor 0, significando que existem 0 peças com comprimento de 4.0m no primeiro padrão.

O raciocínio é semelhante para os restantes 2 padrões, isto é, para as colunas 1 e 2 da matriz *P*, respetivamente, sendo que cada coluna vai representar um padrão que será mais ou menos eficiente. Não pode haver um padrão com um total de 0 peças.

A solução para o problema do corte será um vetor cujos elementos indicam quantas placas de cada padrão, respetivamente, deverão ser cortadas para se obterem as peças desejadas. Este vetor é gerado aleatoriamente. Por exemplo:

sol = [1, 3, 20]

Ou seja, nesta solução, seriam cortadas 1 placa usando o primeiro padrão, 3 placas usando o segundo padrão e 20 placas usando o terceiro padrão.

- Número de peças de 2.0m: 3x1 + 0x3 + 2x20 = 43

- Número de peças de 3.0m: 1x1 + 3x3 + 0x20 = 10
- Número de peças de 4.0m: 0x1 + 0x3 + 1x20 = 20

Obviamente que esta não é uma solução ótima visto que corta peças a mais num caso (2.0m):

- Número de peças de 2.0m: 43 > 20 peças a cortar
- Número de peças de 3.0m: 10 = 10 peças a cortar
- Número de peças de 4.0m: 20 = 20 peças a cortar

Em termos de desperdício podemos fazer as seguintes contas:

- Usando o primeiro padrão são cortados 3 x 2.0m + 1 x 3.0m = 9.0m, logo não há desperdício.
- Usando o segundo padrão são cortados 3 x 3.0m = 9.0m, logo não há desperdício.
- Usando o terceiro padrão são cortados 2 x 2.0m + 1 x 4.0m = 8.0m, logo há 1m de desperdício. As 20 peças cortadas usando o terceiro padrão vão gerar 20 x 1.0m = 20.0m de desperdício.
- Como o número de peças de 2.0m cortadas (43) excede o número de peças pedidas (20), podemos considerar que as peças a mais também são desperdício, logo temos 43-20=23 peças desperdiçadas, que correspondem a 23 x 2.0m = 46.0m de desperdício.

No total existem 20.0m + 46.0m = 66.0m de material desperdiçado.

O objetivo consiste em encontrar o melhor vetor solução, para a matriz de padrões gerada aleatoriamente (esta matriz, dado que é gerada aleatoriamente, provavelmente não será a matriz ótima para resolver o problema), tendo em conta o número de peças cortadas e a quantidade de metros de desperdício.

Só são considerados vetores solução válidos aqueles em que o número de peças cortadas permita **pelo menos** satisfazer as necessidades pedidas. No exemplo anterior as necessidades eram de 20 peças de 2.0m, 10 peças de 3.0m, e 20 peças de 4.0m. Por exemplo o vetor solução [3, 1, 18] seria considerado inválido, já que não seria possível obter 20 peças de 4.0m porque que o número de peças de 4.0m cortadas seria apenas 18 (inferior a 20).

Algoritmos de resolução

O algoritmo mais óbvio é o algoritmo de força bruta, em que todas combinações para o vetor solução possíveis são testadas até se encontrar a solução que corte um número de placas o mais próximo possível ao pretendido (em que fique o menor número de placas por cortar), com o menor desperdício. Infelizmente este algoritmo demoraria muito tempo a pesquisar soluções para além da necessidade de ter também todos os padrões de corte possíveis!

Assim, o algoritmo proposto para este trabalho é o algoritmo **AJR Pseudo-Evolutivo (AJR-PE)**, com o seguinte funcionamento:

1. Inicializa-se uma matriz de padrões de forma aleatória, e gera-se um vetor solução de forma aleatória.

2. Gera-se um vetor solução de forma aleatória a partir da solução já existente, alterando apenas um dos elementos do vetor.

3. Calcula-se o comprimento de peças que ficaram por cortar e a quantidade de material desperdiçado, ambos em metros.

4. O algoritmo volta ao ponto 2 enquanto não houver uma condição de término dada pelo tempo ou número de iterações máxima.

No fim, o algoritmo deverá ser capaz de retornar o vetor solução com o menor valor da soma do número de peças por cortar com o desperdício encontrado durante a sua execução. Note que a melhor solução encontrada pelo algoritmo pode ou não ser a melhor solução em termos globais.

Implementação concorrencial do algoritmo AJR-PE

Dado que o algoritmo AJR-PE tem uma forte componente aleatória, um dos grandes fatores que pode influenciar a solução é o número de iterações realizadas pelo algoritmo (ou de forma indireta, o tempo que se dá ao algoritmo para tentar encontrar a melhor solução).

Desta forma, propomos a implementação paralela e concorrencial do algoritmo nas suas versões *Base* e *Avançada*.

Versão Base

1. Criar *p* processos (número parametrizável) em que cada processo corre o algoritmo AJR-PE.

2. À medida que cada processo encontra uma solução melhor que a anterior, coloca a sua solução na memória partilhada.

3. Dado que dois ou mais processos podem aceder simultaneamente à memória partilhada e corrompê-la, a atualização desta deve ser feita de forma controlada.

4. Ao fim de algum tempo, termina-se a execução e informa-se o utilizador da melhor solução encontrada.

Versão Avançada

A versão avançada é semelhante à versão base com as seguintes alterações:

1. À medida que cada processo encontra uma solução melhor que a anterior, coloca a sua solução na memória partilhada e envia um sinal ao processo principal avisando-o que houve uma atualização da melhor organização das peças na placa.

2. O processo principal, sendo informado que um dos processos encontrou um caminho melhor, deverá enviar um sinal a todos os processos filhos de modo que estes atualizem a sua matriz com uma coluna (escolhida aleatoriamente) da melhor solução encontrada, de maneira a que possam ter um padrão de corte em comum com a melhor solução.

3. Todos os processos deverão resumir recomeçar a sua atividade com o novo padrão adicionado.

Dada a possibilidade que um processo pode estar no meio da execução do algoritmo AJR-PE e ser informado da atualização da organização das peças na placa por parte do processo pai (invalidando ou corrompendo os dados) considere a utilização de mecanismos de sincronização, como por exemplo semáforos.

Desenvolvimento

A aplicação deverá ser feita na linguagem de programação C, em Linux, usando as técnicas de programação concorrencial utilizadas nas aulas laboratoriais, nomeadamente *fork*, funções de manipulação de memória partilhada, semáforos, etc.

Entradas

A entrada de informação é feita usando ficheiros de texto, um para cada problema. Cada ficheiro de texto está separado por linhas, em que na primeira linha é indicado o número de padrões a gerar (*n*), na segunda linha o número de comprimentos diferentes (m), na terceira linha o comprimento em metros da placa (*maxComprimento*), na quarta linha os comprimentos em metros das peças a cortar separados por um espaço (vetor *compPecas*) e na quinta linha as quantidades de peças a cortar de cada comprimento separadas por um espaço (vetor *qtddPecas*).

```
3
3
9
2 3 4
20 10 20
```

O programa deverá ser capaz de ser invocado pela linha de comandos passando como argumentos o nome do ficheiro de texto com o problema, o número de processos a serem criados e o tempo máximo de execução do algoritmo (em segundos). Por exemplo, o comando **pcu prob03.txt 10 60** deverá executar o ficheiro de teste “prob03.txt” usando 10 processos em paralelo durante 60 segundos.

Resultados

De modo a se validar a qualidade do algoritmo, deverá ser construída uma tabela com as seguintes colunas:

a) Número do teste (de 1 a 10).
b) Nome do teste.
c) Tempo total de execução.

d) Número de processos usado (parâmetro *p* na descrição dos algoritmos).
e) Vetores solução encontrados, valor total do comprimento das peças por cortar em metros para cada solução do vetor, e valor total do desperdício em metros para cada solução do vetor.
f) Número de iterações necessárias para chegar ao melhor vetor solução encontrado.
g) Tempo que demorou até o programa atingir o melhor vetor solução encontrado.

Cada teste deverá ser repetido 10 vezes para os mesmos parâmetros de entrada, e deverá ser possível obter valores médios de tempo e número de iterações, assim como o número de vezes em que se encontrou a solução ótima. Os ficheiros de teste a utilizar serão disponibilizados no Moodle da disciplina, assim como um ficheiro com alguns resultados.

Entrega e avaliação

Os trabalhos deverão ser realizados em grupos de 2 alunos da mesma turma de laboratório, e deverão ser originais. Trabalhos plagiados ou cujo código tenha sido partilhado com outros serão atribuídos nota **zero**.

Todos os ficheiros deverão ser colocados num **ficheiro zip** (com o número de todos os elementos do grupo) e submetido via *Moodle* **até às 23:55 do dia 12/Dezembro/2021**. Deverá também ser colocado no zip um **relatório em pdf** com a identificação dos alunos, as tabelas de resultados e a descrições das soluções que considerarem relevantes. Este documento deverá ser mantido curto e direto (2-3 páginas).

Irá considerar-se a seguinte grelha de avaliação:

Algoritmo AJR-PE	04 val.
Algoritmo concorrencial	
Versão Base	04 val.
Versão Avançada	03 val.
Outra versão original	02 val.
Utilização de memória partilhada	01 val.
Utilização de semáforos	01 val.
Relatório com a tabela de testes	02 val.
Qualidade da solução e código	03 val.

As discussões dos projetos serão realizadas na semana seguinte à entrega do projeto, no horário das aulas laboratoriais. As notas poderão ser atribuídas aos alunos de forma individual.

Bom trabalho!