Problema de Otimização - Maximização dos Produtos dos Cortes

Daniel Campagna

Novembro 2019

1 Introdução

O problema de Maximização do Produto dos Cortes (MPC) pode ser definido da seguinte maneira:

> Def. Seja o valor $n > 1$, pertencente ao conjunto dos números naturais, dado como entrada. Determine qual é o maior produto n_1, n_2, ..., $n_{\text{m-1}}$, n_m, com m_2, onde n_i é natural, $i \in \{1, 2, ..., m\}$ e tal que $n_1 + n_2 + ... + n_{\text{m-1}} + n_m = n$.

Outra forma mais comum (e mais simples) de descrever o problema - e que por isso é usada nas seções que seguem - é: suponha uma corda de tamanho n. Desejamos cortar a corta em m_2 partes (i.e., pelo menos um corte) de tamanhos inteiros maiores que zero, a fim de que o produto do tamanho das partes cortadas seja máximo. Sabemos que para $n = 2$ o MPC é 1. Para $n = 3$, o MCP é 2; $n = 4$, 4; e $n = 5$, o MPC será 6.

Este trabalho se propõe a apresentar duas soluções possíveis para resolver o problema de Maximização dos Produtos dos Cortes. Na seção 2 busca uma solução através de uma abordagem top-down. A seção 3 traz a abordagem Botton-up. Em ambas é apresentado seu algoritmo, bem como a prova da corretude e o cálculo da complexidade.

2 Abordagem Top-down

Algoritmo 1: ALGORITMO DE MAXIMIZAÇÃO DO PRODUTO DOS CORTES

Entrada: n

Saída: O Máximo Produto dos Cortes

```
início
    se n <= 1 então
        retorna 0
    fim
    max = 0
    para i de 1 até n − 1 faça
        p1 = i ∗ (n − i)
        p2 = MPC(n − i) ∗ i
        se p1 > max então
            max = p1
        fim
        se p2 > max então
            max = p2
        fim
    fim
    retorna max
fim
```

O algoritmo, a cada chamada feita a ele, gera todos $n-1$ possíveis cortes binários (i.e. $m = 2$) $n_{1,\mathrm{i}}$ e $n_{2,\mathrm{i}}$, para $i \in \{1, 2, ..., n-1\}$, para um dado $n \geq 2$. Em seguida, para cada uma dessas possibilidades i, ele calcula: o produto das duas partes desse corte (i.e. $p_{1,\mathrm{i}} = n_{1,\mathrm{i}} \cdot n_{2,\mathrm{i}}$); e o produto da primeira parte com o MPC da segunda parte (i.e. $p_{2,\mathrm{i}} = n_{1,\mathrm{i}} \cdot MPC(n2, i)$), e fica com o maior desses produtos $resultado_i = max\{p_{1,\mathrm{i}}; p_{2,\mathrm{i}}\}$. Por fim, o algoritmo retorna o maior $resultado_i$ encontrado (i.e. $max\{resultado_i; i = 1, 2, \cdots, n-1\}$). Em outras palavras, o algoritmo pode ser definido da seguinte maneira:

$$MPC(n) = \begin{cases} \max\left\{\max\left\{n_{1,i}n_{2,i}, n_{1,i}MPC(n_{2,i})\right\}; i = 1, ..., n-1\right\}, & \text{se } n \geq 2 \\ 0, & \text{se } n \leq 1 \end{cases} \qquad (1)$$

A Figura 1 ilustra com um exemplo as chamadas da funcão.

2.1 Corretude

Queremos demonstrar que 1 sempre retorna um produto $MPC(n) = n_1 \cdot n_2 \cdot \cdots \cdot n_{m-1} \cdot n_m$, com $m \geq 2$, onde $n_i \in N$ para cada $i \in \{1, 2, ..., m\}$ e tal que $n_1 + n_2 + \cdots + n_{m-1} + n_m = n$, de forma que $m' \geq 2$, e $n'_1, \ldots, n'_{m'} \in N$ satifazendo $n'_1 + n'_2 + \cdots + n'_{m'-1} + n'_{m'} = n$, tem-se que: $n_1 \cdot n_2 \cdots n_{m-1} \cdot n_m \geq n'_1 \cdot n'_2 \cdots n'_{m'-1} \cdot n'_{m'}$.

Base: Pela definição do algoritmo, sabemos que seu caso base é $n = 1$, onde temos que o $MPC(n) = 0$.

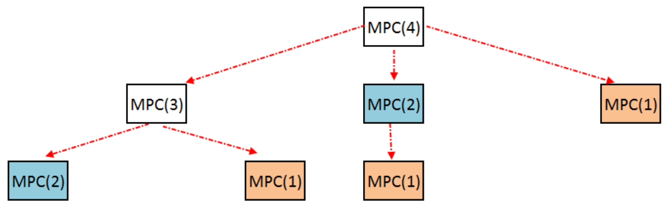

Figure 1: Árvore de chamada das função MPC iniciando com n = 4. Referência: ¡https://algorithms.tutorialhorizon.com/dynamic-programming-maximum-product-cutting-problem/¿

Hipótese: Para $k \in \{2, 3, \ldots, n\}$, supomos, por hipótese, que o $MPC(k) = k_1 k_2 \cdots k_m$, com $m \geq 2$, satifaz o que desejamos demonstrar.

Indução: Queremos demonstrar, agora, que em $MPC(n+1)$ também é válida a propriedade a ser demonstrada nesta prova.

Pela definição do algoritmo, sabemos que:

$$MPC(n) = \quad \max\{\max\{n_{1,i}n_{2,i}, n_{1,i}MPC(n_{2,i})\}; i = 1, ..., n-1\}, \quad \text{se } n \geq 2 \tag{2}$$

A partir de 2, existe $j \in \{1, \ldots, n-1\}$ e este é o que será obtido pelo 2, isto é:

$$MPC(n) = \quad \max\{n_{1,j}n_{2,j}, n_{1,j}MPC(n_{2,j})\} \tag{3}$$

Deste modo, temos dois casos a considerar:

caso 1: $n_{1,j}n_{2,j} > n_{1,j}MPC(n_{2,j})$, logo, $n_{1,j}n_{2,j}$ é o maior produto possível pela maximalidade do j, mantendo a propriedade válida para $MPC(n+1)$.

caso 2: $n_{1,j}MPC(n_{2,j}) \geq n_{1,j}n_{2,j}$. Neste caso, como o $n_{2,j} \leq n$, temos pela hipótese de indução que $MPC(n_{2,j}) = k_1 k_2 \cdots k_r$ é o produto máximo do corte. Note que é suficiente mostrar que $MPC(n+1) = n_{1,j}k_1 k_2 \cdots k_r$ é o produto máximo de cortes. Suponha por absurdo que $n+1 = l_1 + l_2 + \cdots + l_t$, com $l \in N$ e $t \geq 2$, tal que

$$l_1 l_2 \cdots l_t > n_{1,j}MPC(n_{2,j}) = n_{1,j}k_1 k_2 \cdots k_r$$

Observe que $l_1 MPC(l_2 + \cdots + l_t) \geq l_1 l_2 \cdots l_t$, uma vez que a hipótese é válida para $l_2 + \cdots + l_t \leq n$. O que implica

$$l_1 MPC(l_2 + \cdots + l_t) > n_{1,j}MPC(n_{2,j})$$

Por outro lado,

$$\begin{aligned}\max\{l_1(l_2+\cdots+l_t), l_1MPC(l_2+\cdots+l_t)\} &\geq l_1MPC(l_2+\cdots+l_t)\}\\ &> n_{1,j}MPC(n_{2,j})\\ &= \max\{n_{1,j}n_{2,j}, n_{1,j}MPC(n_{2,j})\}\end{aligned}$$

Ou seja, conseguimos uma corte para $n+1$ em duas partes tal que:

$$\max\{l_1(l_2+\cdots+l_t), l_1MPC(l_2+\cdots+l_t)\} > \max\{n_{1,j}n_{2,j}, n_{1,j}MPC(n_{2,j})\}$$

O que é um absurdo pois contraria a escolha de j.

Logo, $n_{1,j}MPC(n_{2,j})$ é o produto máximo par $n+1$. Concluindo que $MPC(n)$ é o produto máximo para qualquer n.

2.2 Complexidade

Para definir a função de complexidade do algoritmo, é suficiente identificar o conjunto das complexidades de cada passo do algoritmo. As linhas 1-5 são executadas em tempo constante $\theta(2)$,para $n>1$, as linha 6-15 também são executadas em tempo constante $\theta(5)$, contudo são repetidas $n-1$ vezes. Além disso, nessas linhas há uma chamada recursiva para o algoritmo, passando um parâmetro $n-i<n$. Logo, podemos definir.

$$T(n) = 2 + 5(n-1) + \sum_{i=1}^{n-1}(T(n-i))$$

Sendo a forma simplificada desse somatório:

$$\sum_{i=1}^{n-1}(T(n-i)) = (n-1)\frac{(T(1)+T(n-1)}{2} = (1+T(n-1))\frac{(n-1)}{2}$$

O que nos dá a seguinte função:

$$T(n) = 2 + (11 + T(n-1))\frac{(n-1)}{2} \tag{4}$$

Agora, precisamos resolver essa recorrência. Para isso, podemos utilizar o método iterativo.

$$\begin{array}{llcl}
\text{it 1} & T(n) & = & 2+(11+T(n-1))\frac{(n-1)}{2}\\
 & & = & 2+(11+[\mathbf{2+(11+\boldsymbol{T}(\boldsymbol{n}-2))\frac{(n-2)}{2}}])\frac{(n-1)}{2}\\
\text{it 2:} & & = & 2+13\frac{(n-1)}{2}+(11+T(n-2))\frac{(n-2)}{2}\frac{(n-1)}{2}\\
 & & = & 2+13\frac{(n-1)^2}{2}+(11+[\mathbf{2+(11+\boldsymbol{T}(\boldsymbol{n}-3))\frac{(n-3)}{2}}])\frac{(n-2)}{2}\frac{(n-1)}{2}\\
\text{it 3:} & & = & 2+13\frac{(n-1)}{2}+13\frac{(n-2)}{2}\frac{(n-1)}{2}+(11+T(n-3))\frac{(n-3)}{2}\frac{(n-2)}{2}\frac{(n-1)}{2}
\end{array}$$

Podemos deduzir que na i-ésima iteração, teremos:

$$T(n) = 2 + \sum_{j=1}^{i-1}(13\prod_{k=1}^{j}\frac{(n-j)}{2}) + (11 + T(n-i))\prod_{k=1}^{i}\frac{(n-k)}{2} \tag{5}$$

Na i-ésima iteração onde $i = n - 1$, teremos

$$T(n) = 2 + \sum_{j=1}^{n-2}(13\prod_{k=1}^{j}\frac{(n-j)}{2}) + (11 + T(1))\prod_{k=1}^{n-1}\frac{(n-k)}{2}$$

$$T(n) = 2 + \sum_{j=1}^{n-2}(13\prod_{k=1}^{j}\frac{(n-j)}{2}) + (12)\prod_{k=1}^{n-1}\frac{(n-k)}{2} \tag{6}$$

Percebemos que a função $T(n) = O(n^{(n-1)})$.

3 Abordagem Bottom-Up

Algoritmo 2: ALGORITMO DE MAXIMIZAÇÃO DO PRODUTO DOS CORTES

Entrada: n

Saída: Uma lista com o Máximo Produto dos Cortes de cada $n' \leq n$

```
início
    mat = [n+1];
    para i de 1 até n+1 faça
        mat[i] = 0;
    fim
    para i de 1 até n+1 faça
        mp = 0;
        para j de 0 até i faça
            max = 0;
            se j * mat[i - j] ¿ j * (i - j) então
                max = j * mat[i - j];
            fim
            senão
                max = j * (i - j);
            fim
            se max ¿ mp então
                mp = max;
            fim
        fim
        mat[i] = mp;
    fim
    retorna mat
fim
```

Perceba na Figura 1 que o Algoritmo 1 repete cálculo de mesmos subproblemas várias vezes. Isso é suficiente para que esse problema seja identificado como problema de Programação Dinâmica.

A ideia do Algoritmo 2 é criar uma lista mat de tamanho $n+1$, para $n \geq 1$, onde cada que cada mat_i, $i \in \{1, 2, \cdots, n+1\}$, armazena a informação do produto máximo de corte para o tamanho de entrada igual a i.

O algoritmo inicia com todos os $mat_i = 0$, em seguida, percorre toda a lista do início até o fim. Enquanto percorre o item mat_i ele olha para cada elemento anterior mat_j, $j < i$, e escolhe o maior resultado entre o produto do corte binário $n_{1,i} = j$ e $n_{2,i} = i - j$, sendo $n_{1,i} + n_{2,i} = n$, ou o produto do corte $n_{1,i} = j$ e mat_{i-j} (que é produto máximo de corte para o tamanho $i - j$).

Em outras palavras, podemos escrever a função como sendo:

$$MPC(n) = \max\{j \cdot (n-j), j \cdot mat_{n-j}; \forall j \in \{1, 2, \cdots, n-1\}\} \tag{7}$$

3.1 Corretude

Esta demonstração é similar à demonstração dada na seção 2.1, devido à semelhança do problema. A diferença aqui consiste no fato de que a cada novo passo dado pela definição do algoritmo a melhor solução dos passos anteriores já foi calculada.

3.2 Complexidade

Para definir a função de complexidade do algoritmo, é suficiente identificar o conjunto das complexidades de cada linha do algoritmo.

A linha 2 é executada em $\theta(1)$. As linhas 3-5, em $\theta(n+1)$. As linhas 6-21 são executadas $n+1$ vezes, e em cada iteração i é executada $O(5 \cdot i)$. Logo, nossa função de complexidade será

$$T(n) = 1 + (n+1) + \sum_{i=1}^{n+1} 5i$$

Substituindo o somatório pela função:

$$\sum_{i=1}^{n+1} 5i = (n+1)\frac{(5 + 5(n+1))}{2}$$

Obtemos

$$\begin{aligned} T(n) &= 1 + (n+1) + (n+1)\tfrac{(5+5(n+1))}{2} \\ &= 2 + n + \tfrac{5}{2}(n+1)(n+2) \\ &= 2 + n + \tfrac{5}{2}(n^2 + 3n + 2) \\ &= \tfrac{5}{2}n^2 + \tfrac{17}{2}n + 7 \end{aligned} \tag{8}$$

Logo, $T(n) = O(n^2)$.

4 Exemplos

Esta seção apresenta os resutados de um experimento feito envolvendo as duas funções. Nesse experimento ambos algoritmos foram executados separadamente com o n variando de 1 até 24. Os resultados do tempo de execução foram plotados na Figura 2.

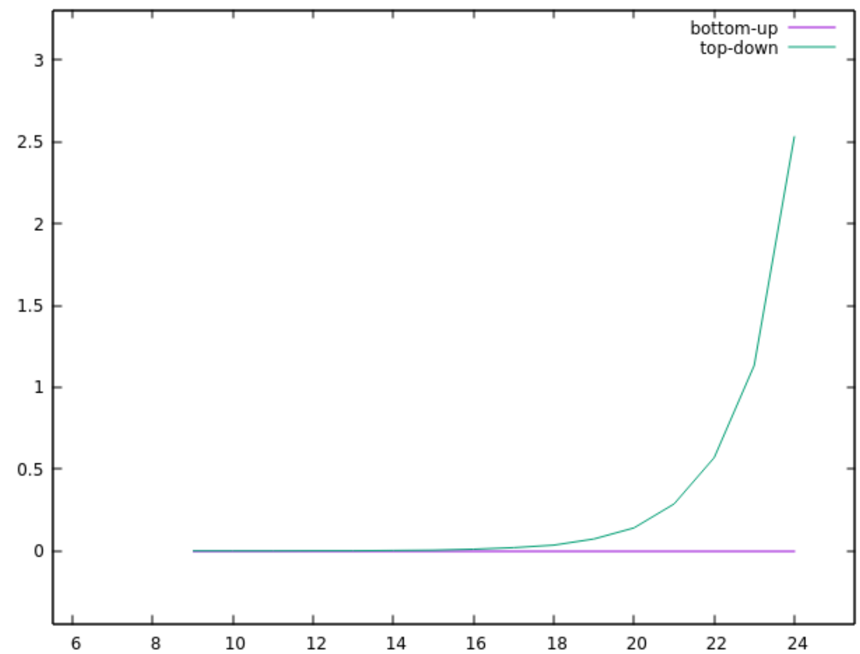

Figure 2: Resultado do experimento envolvendo o Algoritmo 1 e o Algoritmo 2. No Eixo x o tempo de execução, no eixo Y o valor do n. Perceba que o comportamento da abordagem top-down se comporta exponencialmente.

Note que o comportamento da abordagem top-down assemelha-se ao de uma função exponencial, corroborando o resultado da análise de complexidade exposto na seção 2.2. Para observar o comportamento do algoritmo bottom-up, foi elaborado um experimento que executou a função com n variando de 100 até 1000. O resultado, apresentado na Figura 3, mostra que se comporta semelhantemente a uma parábola, o que também dá ensejo ao resultado obtido na sua seção 3.2 de cálculo da complexidade.

5 Conclusão

Neste trabalho foi abordado o problema de Maximização dos Produtos dos Cortes e em seguida apresentados dois algoritmos que solucionam este prob-

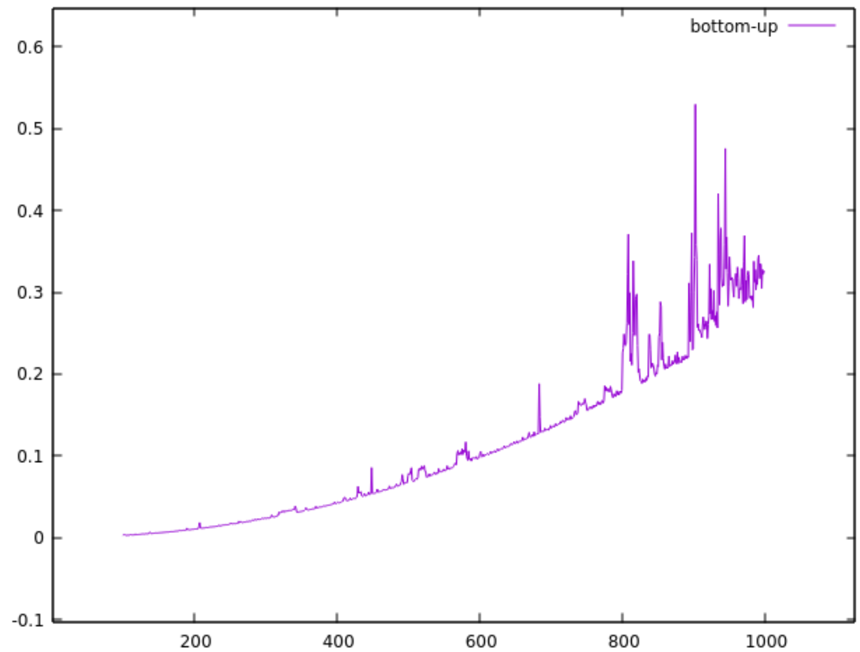

Figure 3: Resultado do experimento envolvendo o Algoritmo 2. No Eixo x o tempo de execução, no eixo Y o valor do n. Perceba se comportamenta semelhante a uma parábola.

lema. Tais soluções tiveram sua corretude provada, bem como a análise de sua complexidade.

Todos os gráfico, códigos e experimentos podem ser obtidos através do endereço ¡https://github.com/danielpcampagna/DynamicProgrammingAlgorithms/settings¿

Apesar do excelente desempenho apresentado pela abordagem bottom-up, existe uma solução que é ainda melhor do que essa. Ela é deriva de uma propriedade que o problema de Maximização dos Produtos dos Cortes tem, que diz:

> Propriedade: Se $n \geq 4$, então $MPC(n) = k_1 k_2 \cdots k_m$ onde $k_i \in \{2, 3\} \forall i = 1, 2, \ldots, m$.

A prova desta propriedade está no Apêndice 12.

Deste modo, podemos utiliar o Algoritmo 3.

Algoritmo 3: ALGORITMO DE MAXIMIZAÇÃO DO PRODUTO DOS CORTES CONSTANTE

Entrada: n

Saída: Uma lista com o Máximo Produto dos Cortes de cada $n' \leq n$

início
    res = n mod 3
    se *res == 0* **então**
        retorna $3^{\lfloor \frac{n}{3} \rfloor}$
        fim
        se *res == 1* **então**
            retorna $3^{\lfloor \frac{n}{3} - 1 \rfloor} \cdot 4$ **fim**
            fim
            se *n == 2* **então**
                retorna $3^{\lfloor \frac{n}{3} \rfloor} \cdot 2$
                fim

A vantagem deste algoritmo, como se pode perceber, é que sua complexidade é O(1). Esse fato pode ser visto na Figura 4, que compara os tempos de execução do Algoritmo 2 e 3. Note que mesmo para valores altos de n, seu tempo de execução parece invariável.

12.

Apêndice 1

A veracidade desta propriedade pode ser concluída a partir da veracidade do Algoritmo 1.

De fato, por simetria, podemos supor que na primeira etapa de geração dos primeiros $n-1$ possíveis cortes binários do Algoritmo 1, é suficiente considerarmos apenas aqueles onde $n_{1,i} \leq n_{2,i}$.

Afirmação 1: Sempre podemos escolher um j satisfazendo 3 e tal que $n_{1,j} \leq 3$.

De fato, se $n_{1,j} \leq 3$ para o j escolhido, não há nada o que fazer. Suponha então que $n_{1,j} \geq 4$. Neste caso temos que $MPC(n_{1,j}) \geq n_{1,j}$ e $MPC(n_{1,j}) \geq n_{1,j}$ e, portanto,

$$max\left\{n_{1,j} \cdot n_{2,j}, n_{1,j} \cdot MPC(n_{2,j})\right\} = n_{1,j} \cdot MPC(n_{2,j})$$

Agora, considere a seguinte partição $n + 1 = 2 + (n_{2,j} + n_{1,j} - 2)$. Pela escolha de j temos que

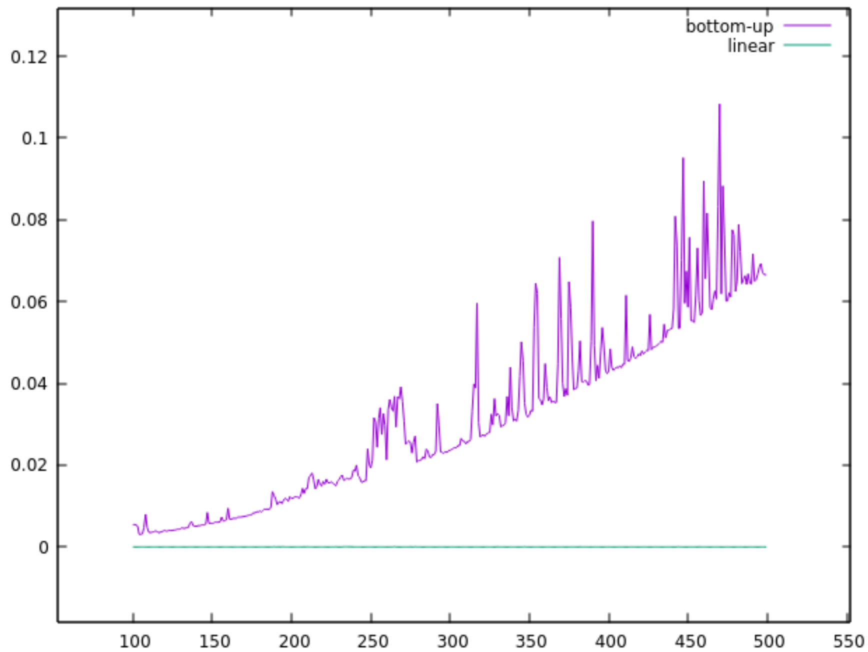

Figure 4: Resultado do experimento envolvendo o Algoritmo 2 e o Algoritmo 3. No Eixo x o tempo de execução, no eixo Y o valor do n. Perceba que o Algoritmo 3 se comportamenta semelhantemente à uma reta.

$$
\begin{array}{rcl}
n_{1,j} \cdot MPC(n_{2,j}) & = & max\,\{n_{1,j} \cdot n_{2,j}, n_{1,j} \cdot MPC(n_{2,j})\} \\
& \geq & max\,\{2 \cdot (n_{2,j} + n_{1,j} - 2), 2 \cdot MPC(n_{2,j} + n_{1,j} - 2)\} \\
& \geq & 2 \cdot MPC(n_{2,j} + n_{1,j} - 2) \\
& \geq & 2(n_{1,j} - 2) \cdot MPC(n_{2,j}) \\
& \geq & n_{1,j} \cdot MPC(n_{2,j})
\end{array}
\tag{9}
$$

Onde a última desigualdade segue do seguinte fato

Fato: Se $n \geq 4$, então $2(n-2) \geq n$.

Com efeito, se $n = 4$, então $2(4-2) = 4$ e a afirmação é verdadeira.

Agora suponha por indução que a afirmação seja verdadeira para algum $n \geq 5$ e vamos mostarar que também é verdade para $n+1$. De fato,

$$2((n+1)-2) = 2 + 2(n-2) \geq 2 + n > n + 1$$

Exatamente o que queríamos.

Voltanto ao nosso problema, temos então que

$$n_{1,j} \cdot MPC(n_{2,j}) \geq 2 \cdot MPC(n_{2,j} + n_{1,j} - 2) \geq 2(n_{1,j} - 2) \cdot MPC(n_{2,j}) \geq n_{1,j} \cdot MPC(n_{2,j})$$

De onde obtemos as seguintes igualdades:

$$2(n_{1,j} - 2) \cdot MPC(n_{2,j}) = n_{1,j} \cdot MPC(n_{2,j})$$

$$n_{1,j} \cdot MPC(n_{2,j}) = 2 \cdot MPC(n_{2,j} + n_{1,j} - 2)$$

Da primeira igualdade obtemos que $2(n_{1,j} - 2) = n_{1,j}$ o que implica $n_{1,j} = 4$. Da segunda igualdade obtemos que a partição $n + 1 = 2 + (n_{2,j} + n_{1,j} - 2)$ também nos dá o máximo, e portanto, podemos considerar esta nova partição onde o $n_{1,j} \leq 3$ o que conclui nossa afirmação.