Artigo Técnico

Revista de
Informática
Teórica e Aplicada

# A Programação Dinâmica: um caso particular da Divisão e Conquista

Laira Vieira Toscani *
PGCC/II – Universidade Federal do Rio Grande do Sul
Paulo Augusto S. Veloso
Pontificia Universidade Católica – PUC-RJ

**Sumário**

A Programação Dinâmica e a Divisão e Conquista são dois métodos de desenvolvimento de algoritmos. Todo problema que pode ser resolvido por um algoritmo desenvolvido por Programação Dinâmica pode também ser resolvido por um algoritmo desenvolvido por Divisão e Conquista.

**Abstract**

Dynamic Programming and Divide and Conquer are algorithm development methods. All problems solved by an algorithm developed through Dynamic Programming can be solved also by an algorithm developed by the Divide and Conquer method.

## 1. Introdução

A Divisão e Conquista é um método descendente que consiste em, dado um problema, decompô-lo em subproblemas menores independentes, resolver esses problemas recursivamente e então combinar as soluções. É útil no desenvolvimento de algoritmos para problemas que podem ser resolvidos pela decomposição em problemas menores, mas do mesmo tipo.

A Programação Dinâmica é um método ascendente, que combina problemas menores e resultados, para obter e resolver problemas maiores. Os resultados são guardados para serem usados numa outra iteração. Uma vantagem do método está no fato de, uma vez resolvido um subproblema, a solução ser guardada e não mais calculada.

Este método é usado para resolver problemas cuja solução é vista como resultado de uma seqüência ótima de decisões, especialmente quando não é fácil

*atualmente na Universidade Nova de Lisboa, Portugal

chegar-se a essa seqüência ótima sem testar todas as seqüências possíveis, para então escolher a melhor. O método em geral reduz drasticamente o número de seqüências candidatas evitando aquelas que sabidamente não podem resultar ótimas. A seqüência ótima de decisões é obtida utilizando-se o Principio da Otimalidade, o qual estabelece que "uma seqüência ótima de decisões é tal que qualquer que seja o estado inicial e a decisão inicial, as decisões seguintes devem constituir uma seqüência ótima para o estado do problema resultante da primeira decisão".

# 2. Definição dos métodos

A especificação formal de um Método de Desenvolvimento de Algoritmos (MDA) consiste de um diagrama sintático que define o domínio das funções e predicados, um programa abstrato que dá a estrutura algoritmica do método e uma axiomatização que define o interrelacionamento das funções e predicados que compõem o método, além de uma verificação da correção do programa abstrato.

## 2.1 Divisão e Conquista

A especificação da Divisão e Conquista apresentada a seguir é devida a Veloso [VEL 80]

**Diagrama Sintático**

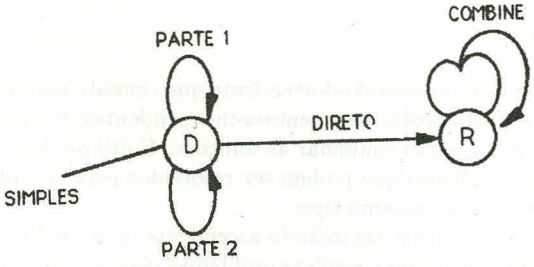


Figura 1: Diagrama Sintático da Divisão e Conquista

Considere os domínios $D$ de problemas e $R$ de resultados. Suponha que um problema é decomposto em dois subproblemas, cujas soluções serão depois com-

binadas. Assim, suponha a existência de duas operações unárias parte1 e parte2 em $D$ e uma operação binária combine em $R$, o predicado simples e a função unária de $D$ em $R$, direto.

**Programa**

```
resultado(p) := se simples(p)
                então direto(p)
                senão combine ( resultado(parte1(p)),
                                resultado(parte2(p)) )
```

Generalizando a idéia, supondo que cada problema é decomposto em $m$ subproblemas, tem-se:

```
resultado(p) := se simples(p)
                então direto(p)
                senão combine ( resultado(parte1(p)),
                                resultado(parte2(p)), ...
                                resultado(partem(p)) )
```

As funções e predicados que constam no programa resultado devem satisfazer o conjunto de axiomas DCAX.

Neste ponto torna-se necessário definir dois predicados: menor e resolveDC.

- menor: $D \times D \rightarrow \{T, F\}$ define uma relação entre dois problemas $p$ e $p'$; menor$(p, p') := p'$ é menor que $p$.
- resolveDC: $D \times R \rightarrow \{T, F\}$ fornece a condição de solução do problema; resolveDC$(p, s) := s$ é solução de $p$.

**Axiomas:** DCAX

ADC1 : $(\forall p)$ simples$(p) \rightarrow$ resolveDC$(p$,direto$(p))$

ADC2 : $(\forall p)(\forall s_1, \ldots, \forall s_k)\{\neg$ simples$(p) \rightarrow [$resolveDC(parte1$(p), s_1) \wedge \ldots \wedge$ resolveDC(partem$(p), s_k) \rightarrow$ resolveDC$(p,$ combine$(s_1, \ldots, s_k))]\}$

ADC3 : $(\forall p)\neg$ menor$(p, p)$

ADC4 : $(\forall p)\{\neg$ simples$(p) \rightarrow [$ menor$(p$,parte1$(p)), \ldots,$ menor$(p$,partek$(p))]\}$

ADC5 : menor está bem formado, i.é não possui cadeia decrescente infinita.

Usando estes axiomas e a indução estrutural de Burstall ([MAN 74]) pode ser provada a correção total do programa. Os axiomas ADC1 e ADC2 garantem a correção parcial e de ADC3 a ADC5 garantem a terminação.

**Exemplo**

Este exemplo, retirado de [TER 82] pag. 39, é um algoritmo de ordenação por intercalação.

**Programa:** OrdInter
**entrada:** $(n, M_1, M_2, \ldots, M_n)$
1. **se** $n = 1$ **então pare-com-saída**$(M_1)$
2. **senão** $k \leftarrow \lfloor n/2 \rfloor$;
3. $L_1 \leftarrow$ OrdInter $(k, M_1, M_2, \ldots, M_k)$;
4. $L_2 \leftarrow$ OrdInter $(n-k, M_{k+1}, \ldots, M_n)$;
5. "intercale $L_1$ e $L_2$ obtendo $(n, L)$";
6. **pare-com-saída**$(L)$
7. **fim-se**

**Identificação dos domínios, funções e predicados**

$D := \{(n, L)/L$ lista e $n =$ tamanho de $L\}$
$R := \{(n, L)/L$ lista ordenada e $n =$ tamanho de $L\}$
simples $(n, L) := n = 1$
direto $:= Id$
parte1 $(n, M_1, \ldots, M_n) := (k, M_1, M_2, \ldots, M_k), k = \lfloor n/2 \rfloor$
parte2 $(n, M_1, \ldots, M_n) := (n-k, M_{k+1}, \ldots, M_n), k = \lfloor n/2 \rfloor$
combine $(L1, L2) :=$ intercale $L_1$ e $L_2$
menor $((n_1, L_1), (n_2, L_2)) := n_2 < n_1$
resolveDC $(L, L') := L'$ tem os mesmos elementos de $L$, mas ordenados.

## 2.2 Programação Dinâmica

A Programação Dinâmica exige uma certa preparação dos dados (problema) que será apresentada como uma Redução.

Antes de formalizar a idéia de redução são necessárias as definições de problema e solução de problema.

Um **problema** ([VEL 81]) é uma terna $p =< D, R, q >$, onde $D$ é o domínio de dados, $R$ domínio de resultados e $q$ uma relação de $D$ em $R$ que define o problema. A **solução** de um problema $p =< D, R, q >$ é uma função $\alpha : D \rightarrow R$ tal que $\forall d \in D(d, \alpha(d)) \in q$.

A **Redução** foi definida em [VEL 84a] como segue: dados dois problemas $p =< D, R, q >$ e $p' =< D', R', q' >$, uma Redução $\Gamma$ de $p$ em $p'$ é um par $< t, v >$ de funções tal que $t : D \rightarrow D'$ e $v : R' \rightarrow R$. Diz-se que $\Gamma$ é uma **Redução Boa** *sss* para qualquer solução $\alpha' : D' \rightarrow R'$ de $p'$, a função $\alpha : D \leftarrow R$ definida como $\alpha(d) = v(\alpha'(t(d))$ é uma solução de $p$.

Assim, uma Redução de um dado problema $p$ em um problema $p'$ é um par $< t, v >$, onde $t$ é a função que transforma os dados do problema $p$ em dados para o problema $p'$ e $v$ a função que transforma um resultado do problema $p'$

em um resultado para o problema $p$. $\alpha$, solução de $p$ é definida a partir de $\alpha', t$, e $v$. Esta é uma definição possível para Redução, a mais intuitiva; [VEL 84b] discute outras definições.

Serão usadas durante o trabalho as funções $Id : X \rightarrow X$, função identidade, e $\Pi : X \rightarrow X$, função projeção, e as operações entre funções assim definidas: $(f \circ g)(x) = f(g(x)), (f \times g)(x, y) = (f(x), g(y))$ e $(f, g)(x) = (f(x), g(x))$.

A Programação Dinâmica requer uma Redução que dado um problema o decompõe em subproblemas mínimos (que devem ter solução fácil) e inicializa a solução parcial com a solução desses problemas, de maneira que a entrada da Programação Dinâmica seja uma seqüência de problemas e uma seqüência de soluções.

**Redução**

Sejam $D$ = conjunto de problemas e $R$ = conjunto de soluções possíveis. $Q = D^+$ = conjunto das seqüências de elementos de $D$, i.é $Q = \bigcup_{i \in N - \{0\}} D^i$, com $D^i = \{x / x$ é uma seqüência de $i$ elementos de $D\}$. $M = R^+$.

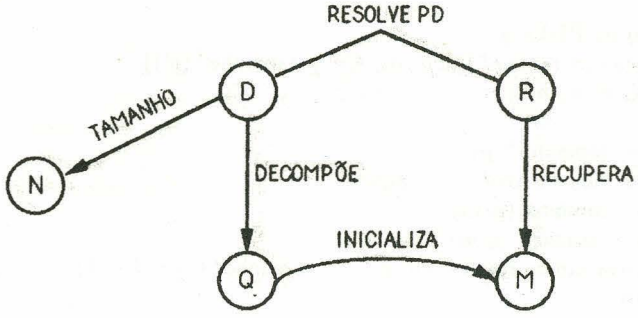


Figura 2: Diagrama de Redução para Programação Dinâmica

A Redução $\Gamma =< t, v >$ t.q. $t : D \rightarrow Q \times N \times M$ e $v : M \rightarrow R$ é uma variação da definição de redução da secção 2.1.1, diferindo daquela nos tipos das funções $t$ e $v$.

$t$:= (decompõe, tamanho, inicializa o decompõe)

$v$:= recupera

- decompõe: decompõe o problema numa seqüência de problemas de tamanho mínimo.
- tamanho: calcula o tamanho do problema.
- inicializa: resolve os problemas resultantes de decompõe.

- resolvePD: dá a condição de solução do problema.

**Diagrama Sintático da Programação Dinâmica**

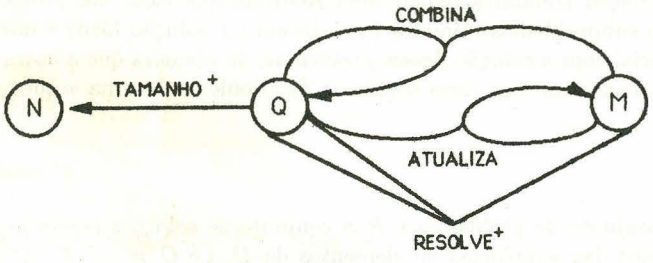


Figura 3: Diagrama Sintático da Programação Dinâmica

**Programa:** PDProg
$\{\varphi(p', n, m) := \text{resolve}^+(p', p', m) \wedge n \geq \text{tamanho}^+(p')\}$
**entrada:** $p', n, m$;
1. $p \leftarrow p'$;
2. mínimo $\leftarrow$ tamanho$^+(p)$
3. **para** $k =$ mínimo **até** $n - 1$ **faça**
4. $\quad p \leftarrow$ combina $(p, m)$
5. $\quad m \leftarrow$ atualiza $(p, m)$
$\quad$ { invariante: resolve$^+(p', p, m) \wedge$ tamanho$^+(p) = k + 1$}
6. **fim-para**
**saída:** $p, m$
$\{ \Psi(p', n, p, m) :\doteq \text{resolve}^+(p', p, m) \wedge \text{tamanho}^+(p) = n\}$

A cada iteração da malha 3-6 a função combina, combina os problemas atuais para criar um novo conjunto de problemas de tamanho uma unidade maior e a função atualiza resolve os problemas atuais, usando as soluções dos problemas de níveis anteriores.

**Funções e Predicados Auxiliares**

- tamanho$^+$ : $Q \leftarrow N$; tamanho$^+(p_1, \ldots, p_i) = \max_{j=1,\ldots,i} \{\text{tamanho}(p_i)\}$.

- resolve$^+$ : $Q \times Q \times M \rightarrow \{T, F\}$; onde resolve$^+((p'_1, \ldots, p'_k), (p_1, \ldots, p_i)$,

$(s_1, \ldots, s_i)) = (p_1, \ldots, p_i)$ foi obtida de $(p'_1, \ldots, p'_k)$ por sucessivas aplicações da função combina e $\bigwedge_{j=1,\ldots,i}$ resolvePD $(p_j, s_j))$.

**Axiomas:** PDAX

APD1 : $(\forall p)(\forall p')(\forall m)\{\text{resolve}^+(p', p, m) \rightarrow \text{resolve}^+(p', \text{combina}(p, m), \text{atualiza}(\text{combina}(p, m), m))\}$

APD2 : $(\forall p)(\forall m)\ \text{tamanho}^+(\text{combina}(p, m)) = \text{tamanho}^+(p) + 1$

**Axiomas da Redução**

ARPD1 : $(\forall p)$ resolve$^+$( decompõe$(p)$, decompõe$(p)$,inicializa$(p)$)

ARPD2 : $(\forall p)(\forall q)(\forall m)\{[$ resolve$^+(p, q, m) \wedge$ tamanho$^+(q) =$ tamanho $(p)]\} \rightarrow$ resolvePD$(p,$ recupera$(m))$

A prova da correção do programa é obtida facilmente.

**Exemplo**

Em [TER 82] é apresentado um exemplo simples que ilustra bem o método. O exemplo será transcrito aqui.

Deseja-se multiplicar $n$ matrizes, i.é, calcular $M = M_1 \times M_2 \times \cdots \times M_n$ , onde cada matriz $M_i$ tem $b_{i-1}$ linhas e $b_i$ colunas, $1 \leq i \leq n$ . O algoritmo trivial $((A \cdot B)_{ij} = \sum_{k=1}^{p} A_{ik} \cdot B_{kj}, i = 1, \ldots, p; j = 1, \ldots, r)$ requer $p \times q \times r$ multiplicação de elementos para multiplicar uma matriz $p \times q$ por outro $q \times r$.

A multiplicação de matrizes é associativa. Existem, portanto, várias maneiras possíveis de se realizar esta multiplicação, com diferentes números de operações correspondentes.

Verifique o seguinte exemplo:

$M =$ $M_1$ $\times$ $M_2$ $\times$ $M_3$ $\times$ $M_4$

$200 \times 2$ $\quad$ $2 \times 30$ $\quad$ $30 \times 20$ $\quad$ $20 \times 20$

$M = ((M_1 \times M_2) \times M3) \times M4)$ requer 212.000 operações

$M = M_1 \times ((M_2 \times M_3) \times M_4)$ requer 10.000 operações

Este exemplo ilustra o fato de que a ordem em que são realizadas as multiplicações influi substancialmente no número total de operações requeridas.

O problema consiste, então, em determinar uma seqüência de multiplicações que requeira um número mínimo de operações.

Um algoritmo que enumere todas as seqüências possíveis de multiplicações, calcule os respectivos números totais de operações requeridas e, em seguida, escolha uma seqüência ótima terá complexidade exponencial em $n$ (número de matrizes), o que é inviável na prática quando $n$ é relativamente grande. Utilizando a programação dinâmica pode-se diminuir a ordem de complexidade para uma ordem polinomial.

O algoritmo que resolve o problema pela programação dinâmica consiste, num primeiro passo, em decompor o problema em $n$ subproblemas de tamanho 1, resolvê-los, guardar os resultados na diagonal de uma matriz. Passo a passo, são combinadas as soluções de $(n-u+1)$ problemas de tamanho $u$ para resolver $(n-u)$ problemas de tamanho $(u+1)$ $(u = 1, 2, \ldots, n-1)$, guardando-se esses resultados nas diagonais superiores da matriz até terminar o processo com a solução de 1 problema de tamanho $n$.

Chama-se de $m_{ij}$ o custo do produto, isto é o número mínimo de multiplicações necessárias para calcular $M_i \times M_{i+1} \times \cdots \times M_j$. Assim, naturalmente $m_{ii} = 0$.

Para $i \leq k \leq j$, chamando-se $M' = M_i \times M_{i+1} \times \cdots \times m_k$ e $M'' = M_{k+1} \times M_{k+2} \times \cdots \times M_j$. O custo de $M'$ é $m_{ik}$ e o custo de $M''$ é $m_{k+1,j}$ . Note que $M'$ é uma matriz de dimensões $b_{i-1} \times b_k$ e que $M''$ é de dimensão $b_k \times b_j$.

O produto $M$ pode ser obtido fazendo $M' \times M''$ e o custo total será $m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j$. Para o custo ser mínimo tem-se que escolher o $k$ apropriado, isto é

$$m_{ij} = \min_{i \leq k < j} (m_{ik} + m_{m+1,j} + b_{i-1} \times b_k \times b_j)$$

Note que, como $i \leq k < j$ tem-se: $k - i < j - i$ e $j - (k+1) < j - i$ portanto os custos (mínimos) $m_{ik}$ e $m_{k+1,j}$ já foram obtidos. Assim não há necessidade de recursão.

**Entrada:** $(b_0, b_1, \ldots, b_n)$, onde $b_{i-1}$ e $b_i$ são as dimensões da matriz $M_i$
**Saída:** $(m_{1n})$, o custo mínimo para se obter o produto $M_1 \times M_2 \times \cdots \times M_n$.
**Programa:**OrdOtima
**entrada:** $(n, b_0, b_1, \ldots, b_n)$

1. **para** $i = 1$ **até** $n$ **faça** $m_{ii} \leftarrow 0$ **fim-para**;
2. **para** $u = 1$ **até** $n-1$ **faça**
3. **para** $i = 1$ **até** $n-1$ **faça**
4. $j \leftarrow i + u$; $\quad (*u = j - i*)$;
5. $m_{ij} \leftarrow \min_{i \leq k < j} (m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j)$
6. **fim-para**
7. **fim-para**
8. **pare-com-saída**$(m_{1n})$

A seqüência ótima das multiplicações pode ser obtida guardando-se os valores de $k$, obtidos na linha 5 do algoritmo.

A iteração da linha 1 é executada $n$ vezes. A iteração das linhas 2 a 7 é executada $O(n)$ vezes, o mesmo acontecendo com as iterações das linhas 3 a 6 e linhas 5. A complexidade total do algoritmo é portanto $O(n^3)$. Vê-se

contudo que a propagação dinâmica tornou possível resolver com uma ordem de complexidade polinomial o problema para o qual o algoritmo imediato teria ordem de complexidade exponencial.

**Identificação dos Domínios e Funções**

$D$ - domínio de instância do problema = conjunto de seqüências de matrizes reais multiplicáveis. $p = (M_1, M_2, \ldots, M_n)$.
$R$ - domínio dos resultados = $N$
$Q$ - $D^+$

**Redução**

- tamanho–número de matrizes consideradas no problema.
- decompõe–decompõe o problema de tamanho $n$ em $n$ problemas de tamanho 1, isto é, ao invés de considerar o produto $M_1 \times M_2 \times \cdots \times M_n$, são consideradas as matrizes $M_1, M_2, \cdots, M_n$.
- inicializa–inicializa a diagonal da matriz $m$ com zeros (solução dos problemas de tamanho 1), linha 1 do programa OrdOtima.
- recupera–recupera o valor contido em $m_{1n}$ que é o número mínimo de operações para efetuar o produto $M_1 \times \cdots \times M_n$, linha 8 do programa OrdOtima.

Funções do Programa PDProg

- combina–combina $n - u + 1$ problemas atuais de tamanho $u$ de maneira a encontrar todos os possíveis $n - u$ problemas de tamanho $u + 1$, isto é, considera todos $(n - u)$ produtos parciais (de tamanho $u+1$) $M_i \times M_{i+1} \times \cdots \times M_{i+u}, i = 1, \ldots, n - u$
- atualiza–preenche a próxima diagonal superior da matriz, da seguinte maneira: $m_{ij} \leftarrow \min_{i \leq k < j} (m_{ik} + m_{k+1,j} + b_{i-1} \times b_k \times b_j)$ para $i = 1, 2, \ldots, n - u$ e $j = i + 1$, iteração 3-6 do programa OrdOtima.

Predicado

- resolvePD$(p, r)$–$r$ é o número de equações necessárias para multiplicar as matrizes de $p$.

## 3. Análise dos métodos

Como foi visto na seção anterior, é possível especificar formalmente MDAs, como a Divisão e Conquista e a Programação Dinâmica. Agora será formalizado o conceito de MDA e feita uma análise comparativa entre estes dois MDAs.

## 3.1 Definições

Dado uma problema $p =< D, R, q >$ e uma solução $\alpha$ para $p$, um **algoritmo** $a_\alpha$ é segundo [KNU 83] um método abstrato que computa $\alpha$.

Recordando as partes que compõem uma especificação formal de um MDA, nota-se que um MDA fica definido a partir da definição de um programa abstrato e um conjunto de axiomas. Assim define-se:

Um **MDA** é um par (Prog, Axio), onde Prog é um programa abstrato definido a partir de funções sintaticamente bem definidas e Axio é um conjunto de axiomas que definem a semântica das funções.

Se $m$ = (Prog, Axio) é um MDA, uma instância de Prog que satisfaz Axio é um **algoritmo desenvolvido** por $m$. Se $p$ é um problema e $\alpha$ é solução de $p$, então $i(m, p, \alpha)$ é o conjunto de todos algoritmos desenvolvidos por $m$, que computam $\alpha$. E $D(m)$, **domínio** de $m$ é o conjunto de todos pares $(p, \alpha)$ tal que $p$ é um problema, $\alpha$ é solução de $p$ e $i(m, p, \alpha) \neq \Phi$.

Se $m_1$ e $m_2$ são dois MDAs, diz-se que $m_2$ é pelo menos tão geral quanto $m_1$, $m_1 \sqsubset m_2$ sss (se $(p, \alpha) \in D(m_1)$, então $\exists \alpha'$ t.q. $(p, \alpha') \in D(m_2)$).

## 3.2 Análise Comparativa

A Divisão e Conquista divide sucessivamente o problema, empilha os subproblemas até atingir o nível mínimo (simples $(p)$), então resolve-os e vai recompondo o problema (é um método recursivo descendente). A Programação Dinâmica recebe uma seqüência de problemas de tamanho mínimo, soluciona esses problemas, guarda os resultados, combina subproblemas menores e seus resultados para obter e resolver problemas maiores, até recompor e resolver o problema original. O problema na Programação Dinâmica é decomposto uma única vez (pela redução), assim os subproblemas menores são gerados antes dos subproblemas maiores (é um método ascendente).

Seja Prog-Din-Seq= (PDProg, PDAX) e seja $(p, \alpha) \in D$(Prog-Din-Seq). Então $p =< D, R, q >$ é um problema, $\alpha$ uma solução de $p$ e $i$(Prog-Din-Seq, $p, \alpha) \neq \Phi$.

Seja PD uma instância de Prog-Din-Seq para $(p, \alpha)$, PD= (PGPD,AXPD) e PGPD uma instância de PDProg obtida pela instanciação das funções tamanho, combina e atualiza.

Seja o predicado resolve tal que resolve $(p, r) := r$ é solução de p (condição de solução de $p$). Seja a redução $\Gamma =< t, v >$ tal que $\vee$(PGPD($t(p)$))= $\alpha(p)$, onde PGPD($x$) é a saída do programa PGPD para a entrada $x$. $t$ = (decompõe, tamanho, inicializa o decompõe), $v$ = recupera.

Seja Div-Con-Seq= (resultado, DCAX)

**Afirmação:** Prog-Din-Seq $\sqsubset$ Div-Con-SEq.

Quer se provar que existe $\alpha'$ t.q. $i$(Div-Con-Seq, $p, \alpha') \neq \Phi$.

**Definição de $\alpha'$:**

Se $i$(Div-Con-Seq, $p, \alpha'$) $\neq \Phi$, então existe PGDC $\in i$(Div-Con-Seq, $p, \alpha'$), PGDC é uma instância de resultado obtida pela instanciação do predicado simples, das funções direto, partei ($i = 1, \ldots, m$), combine e do predicado menor.

Domínio de dados: $D$

Domínio de resultados: $D \times R$

**Definição de PGDC:**

PGDC($p$) := **se** simples($p$)
**então** direto($p$)
**senão** combine ($p$,PGPD (parte1($p$)), $\cdots$, PGPD(partek($p$))

onde:

simples($p$) := $p$ = decompõe($p$)

menor ($p_1, p_2$) := tamanho($p_1$) < tamanho($p_2$)

direto = (Id, inicializa)

combine = procura o( Id $\times$(( combina, atualiza o( combina,$\Pi\times$ Id ) o $\zeta$)), onde $\zeta : (D \times R)^+ \rightarrow D \times R$ é t.q. se $s = (s_1, \ldots, s_e)$ e $s_i = (s_i^1, s_i^2) i = 1, \ldots,$ e então $\zeta(s) = (q, m)$ t.q. $q = (s_1^1, s_2^1, \ldots, s_e^1)$, $m = (s_1^2, s_2^2, \ldots, s_e^2)$ e procura: $D \times D^+ \times R^+ \rightarrow D \times R$ é t.q se $q = (q_1, \ldots, q_k)$ e $r = (r_1, \ldots, r_k)$ então procura $(p, (q, r)) = (p, r_i)$ t.q. $p = q_i$. $\zeta$ e procura são obviamente computáveis.

As funções partei computam os problemas partei($p$), cujas soluções são necessárias para a execução de solução($p$).

Esses problemas devem ser identificados na definição da função atualiza, pois são as soluções utilizadas na execução de atualiza $(q, m)$ (onde $q$ é uma seqüência de problemas de tamanho tamanho($p$) e $m$ contém todas soluções de problemas já geradas). Estas funções partei certamente são computáveis e tamanho (partei($p$)) < tamanho($p$). Uma maneira possível, mas não eficiente de computá-los é executar PGPD para t($p$) e tomar as seqüências $p'$ computadas durante a execução.

Como foi reescrito o conjunto de resultados, uma pequena adaptação se faz necessária no predicado resolveDC, o qual passa a ser resolveDC: $P \times P \times R \rightarrow \{T, F\}$ e resolveDC($p, q, r$) sss $p = q$ e resolvePD($p, r$).

**Verificação dos Axiomas**

ADC1 : ($\forall p$) simples($p$) $\rightarrow$ resolveDC ($p$,direto($p$))

Suponha simples($p$). Então $p$ = decompõe($p$)

resolveDC($p$,direto($p$)) = resolveDC($p, p$, inicializa($p$))

= resolve$^+$($p, p$,inicializa($p$))

= resolve$^+$ (decompõe($p$), decompõe($p$),inicializa($p$))

$\Rightarrow$ resolveDC($p$,direto($p$)) por ARPD1

ADC2 : $(\forall p)(\forall s_1,\ldots,\forall s_k)\{\neg$simples$(p) \rightarrow$ [resolveDC$(p$,parte1$(p), s_1) \wedge$ resolveDC $(p,$ parte2$(p), s_2) \wedge \cdots \wedge$ resolveDC$(p$,partek$(p), s_k) \rightarrow$ resolveDC$(p,$ combine$(p, s_1, s_2, \ldots, s_k)]$

Suponha $\neg$ simples$(p)$ i.é $p \neq$ decompõe$(p)$.

Suponha resolveDC$(\text{parte1}(p), s_1) \wedge \cdots \wedge$ resolveDC$(\text{partek}(p), s_k)$.

combine $(p, s_1, \ldots, s_k) =$

$=$procurao[Id$\times$((combina,atualiza o(combina,$\Pi\times$Id))o$\zeta$)]$(p, s_1, \ldots, s_k)$

$=$ procura $(p,$ (combina, atualiza o ( combina, $\Pi\times$ Id )) $\zeta$ $(s_1, \ldots, s_k))$

$=$ procura $(p,$combina$(p, m),$ atualiza( combine$(p, m), m))$

$=$ procura $(p, p'', m')$

$= (p, m_j)$

onde $\zeta(s_1, \ldots, s_k) = (p, m)$, com

$s_i = (s_i^1, s_i^2) i = 1, \ldots, k, p = (s_1^1, \ldots, s_k^1), m = (s_1^2, \ldots, s_k^2)$;

combina$(p, m) = p'' = (p''_1, \ldots, p''_k)$;

atualiza(combine$(p, m), m) = m' = (m'_1, \ldots, m'_k), m_j$ é t.q. $p = p''_j$.

Por hipótese tem-se resolveDC $(p,$ parte1$(p), s_1) \wedge \cdots \wedge$ resolveDC$(p,$partek$(p)$, $s_k)$, i.é resolve (partei$(p)$, $s_i)$, $i = 1, \ldots, k$, i.é resolve$^+(p',$ (parte1$(p)$ ), parte2$(p), \cdots,$ partek$(p)), (s_1, s_2, \ldots, s_k)$. Então resolve$^+(p',$ combina$(p, m),$ atualiza (combina$(p, m), m)$ ) pelo axioma APD1:

$\Rightarrow$ resolve$^+(p', p'', m')$

$\Rightarrow$ resolve$(p''_i, m'_i)$ $\qquad i = 1, \ldots, k$

$\Rightarrow$ resolve$(p''_j, m'_j)$

$\Rightarrow$ resolveDC$(p''_j, p''_j, m_j)$

$\Rightarrow$ resolveDC $(p, p''_j, m_j)$ $\qquad (p = p''_j)$

ADC3 : $(\forall p)\neg$ menor$(p, p)$

I.é $\neg$(tamanho$(p) <$ tamanho$(p))$. Óbvio.

ADC4 : $(\forall p)\{\neg$ simples$(p) \rightarrow$ [menor$(p,$ parte1$(p)),\cdots,$ menor $(p,$ partek$(p))\}$

Sai da definição de partei.

ADC5 : menor está bem formado, i.é não possui cadeia decrescente infinita.

Sai da definição de menor.

# 4. Conclusão

Defina $\alpha' : P \rightarrow R$ t.q. se PGDC($p$)= $(p', r), \alpha'(p) = r$, i.é, $\alpha' = (\pi, Id)$ o PGDC. $\alpha'$ é solução de $p$ e $(p, \alpha') \in D(m_2)$. Assim dado um problema resolvido por Prog-Din-Seq tomada esta solução, construimos uma solução $\alpha'$ que resolve $p$ por Div-Con-Seq. Isto é, Prog-Din-Seq $\sqsubset$ Div-Com-Seq.

Conclusão: todo problema que tem solução por Programação Dinâmica tem também solução por Divisão e Conquista.

Em [TOS 88c], no Apêndice B2 são apresentados dois exemplos de problemas resolvidos tradicionalmente por Programação Dinâmica, resolvidos lá por Divisão e Conquista. Os problemas são PCMCEV (Problema do caminho mais curto entre vértices) e POOMM (Problema da ordem ótima da multiplicação de matrizes), este último apresentado aqui, na seção 2.2.

A partir de uma formalização de MDAs torna-se possível o estudo de suas propriedades e das relações existente entre eles. Alguns resultados interessantes já foram obtidos, como o que foi aqui apresentado e aquele que diz "o método Guloso é um caso particular da Programação Dinâmica" [TOS 88a]. O mais importante, entretanto, é que mostrou-se ser possível provar aquelas propriedades para as quais se tem uma noção intuitiva, e outras tais como: através da utilização apropriada de paralelismo pode-se reduzir a complexidade de algoritmos e saber exatamente as condições em que: (a) é possível obter ganhos, (b) é possível esperar um ganho ótimo, (c) é impossível obter-se ganho. Alguns resultados nesse sentido são apresentados em [TOS 88b].

# Referências

[AHO 74] AHO, A.V.; HOPCROFT, J.E. & ULLMAN, J.D. "The Design and Analysis of Computer Algorithms". Reading, Mass., Addison-Wesley, 1974.

[KNU 83] KNUTH, D.E. Algorithm and program, information and data. Communication of the ACM, New York 26(1):56, Jan. 1983. pp. 56.

[MAN 74] MANNA, Z. "The Mathematical Theory of Computation", New York, McGraw-Hill, 1974.

[SMI 85] SMITH, D.R. "The design of divide and conquer algorithms". Science Computer Programming, 5. 1985, p. 37-58.

[TER 82] TERADA, R. Desenvolvimento de Algoritmos e Complexidade de Computação. RJ, Terceira Escola de Computação, Depto de Informática, PUC, 1982.

[TOS 86a] TOSCANI, L.V. & VELOSO, P.A.S. Divisão e Conquista: análise da complexidade. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, 13., Olinda, 19-25 Jul., 1986. Anais. Recife, SBC, 1986. p. 89-104.

[TOS 86b] TOSCANI, L.V. & VELOSO, P.A.S. Especificação formal e análise da complexidade da programação dinâmica. Porto Alegre, CPGCC da UFRGS, 1986. (RP 049).

[TOS 87] TOSCANI, L.V. & RIBEIRO, C.C. Análise de complexidade da técnica da divisão e conquista para máquinas paralelas com estrutura de árvore. Pesquisa Operacional, 7(2):66-86, dez. 1987.

[TOS 88a] TOSCANI, L.V. & VELOSO, P.A.S. Programação dinâmica e método guloso. In: SIMPÓSIO BRASILEIRO de ENGENHARIA de SOFTWARE, 2., Canela, out. 27-28, 1988. Anais. Porto Alegre, SBC/UFRGS, 1988.

[TOS 88b] TOSCANI, L.V. Métodos de desenvolvimento de algoritmos: análise comparativa e de complexidade. Rio de Janeiro, Dept. de Informática da PUC/RJ, 1988. Tese de Doutorado.

[VEL 80] VELOSO, P.A.S. "Divide-and-Conquer via data types". In: LATIN AMERICAN CONFERENCE ON INFORMATICS, 7., Caracas, 1980. Proceedings.

[VEL 81] VELOSO, P.A.S. & VELOSO S.R.M. Problem decomposition and reduction: applicability, soundness, completeness. In: PROGESS in CYBERNETICS and SYSTEMS RESEARCH. Washington, Hemisphere, 1981.

[VEL 84a] VELOSO, P.A.S. Outlines of mathematical theory of general problems. Philopophia Naturalis, 21(2-4): 354-67. 1984.

[VEL 84b] VELOSO, P.A.S. & MARTINS, R.C.B. "On reducibilities among general problems". In: INTERNATIONAL CONGRESS CYBERNETICS, 10., 1984. Proceedings. Oxford, Pergamon, 1984.

**Curriculum vitae** (Laira Vieira Toscani)

Bacharel em Matemática pela PUC/RJ em 1971, Mestre em Informática pela PUC/RJ em 1973, Doutora em Informática pela PUC/RJ em 1988. Professora do Instituto de Informática e Curso de Pós-Graduação em Ciência da Computação da UFRGS desde 1975. Atualmente em Pós-Doutorado no Grupo de Programação em Lógica e Inteligência Artificial da Universidade Nova de Lisboa, Portugal. Área de interesse: Análise e Desenvolvimento de Algoritmos. E-mail: LVT@HOST.FCTUNL.RCCN.PT.

**Curriculum vitae** (Paulo A. S. Veloso)

PhD em Ciência da Computação pela Universidade da Califórnia, Berkley, 1975. Atualmente Professor Associado da PUC-RJ. Áreas de interesse: Especificações Formais, Teoria e Metodologias de programação.

*Convertido, editado, formatado e composto por Raul F. Weber a partir do texto original em Chi-Writer fornecido pelos autores em disquete. Figuras realizadas por Taisy S. Weber a partir de cópias fornecidas pelos autores.*