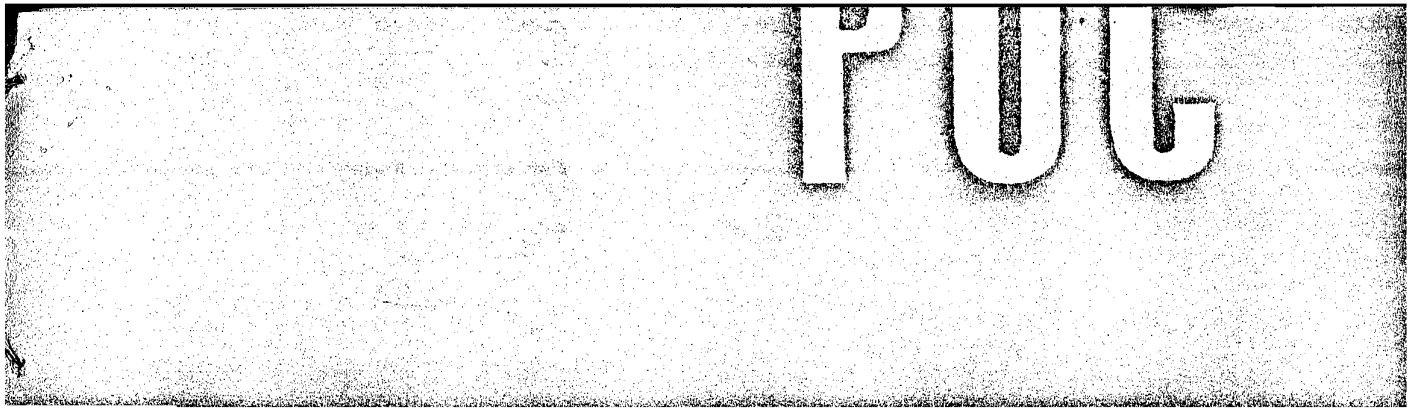


Series: Monographs in Computer Science
and Computer Applications

Nº 11/70

RECURSIVE TECHNIQUES IN DYNAMIC PROGRAMMING

by

Antonio Luz Furtado

Computer Science Department - Rio Datacenter

CENTRO TECNICO CIENTIFICO
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 209 - ZC 20
Rio de Janeiro - Brasil

RECURSIVE TECHNIQUES IN DYNAMIC PROGRAMMING

by

Antonio Luz Furtado

Computer Science Department - Rio Datacenter

ABSTRACT:

Dynamic programming is an important area of operations research. It allows the solution of a wide class of optimization problems that are expressible in a certain very general recursive formula.

Since recursion was not available in most programming languages, these problems could not be solved through a direct application of the formula. A frequently laborious task has been undertaken to devise iterative techniques for each problem.

However if one has an efficient implementation of recursion, the direct approach becomes more attractive, for no preliminary reformulation is needed and ease and naturalness in the transition fron the problem to the programming algorithm are ensured.

The example we present here-the determination of the maximum or minimum path in a network - also illustrates the use of list processing to provide a quite natural input format and a convenient storage representation for a graph. The use of a non - numerical technique, like processing, to an essentially numerical problem is a noticeable trend in present day programming.

1 - THE PROBLEM:

Let $G = (X, U)$ be a network, where X is the set of vertices and U the set of edges.

To each edge $(X_i, X_j) \in U$ a value $c_{ij} \geq 0$ is associated.

A path $\mu = [X_0, X_{i_1}, X_{i_2}, \ldots, X_n]$ must be found, such that

$c_{0i_1} + c_{i_1 i_2} + c_{i_2 i_3} + \ldots + c_{i_k i_n}$ is optimal.

We now consider the optimality principle enunciated by Richard Bellman: an optimal policy must be composed only of optimal sub-policies. This principle gives rise to the recursive formula:

$$f_k(x) = \operatorname{opt}_{p \in P} \{R_k(x, p) + f_{k-1}[x'(k, x, p)]\}$$

Where:

f_k - optimal total revenue obtainable from a system characterized by x, when k stages still remain to be performed.

P - set of all permissible policies

R_k - revenue from the k^{th} stage

x - state of the system at the k^{th} stage

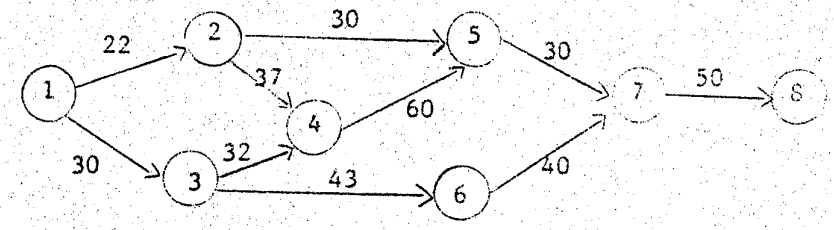
x' - state of the system at the $(k - 1)^{th}$ stage supposing policy p has been adopted.

Adapting this formula to the present problem it becomes:

$$f(X_i) = \underset{X_j \in S(X_i)}{\text{opt}} \left[c_{ij} + f(X_j) \right]$$

where $S(X_i)$ indicates the set of immediate successors of X_i.

Let us examine the example below in order to understand the intuitive meaning of this. Suppose our aim is to obtain a maximum value.



The value of f(1) is the maximum between 22 + f(2) and 30 + f(3). Now, f(2) will not include all possible paths between vertex 2 and vertex 8, but only the maximum among these.

The stage by stage optimization is the feature that reduces the number of comparisons thus reducing the dimension of an otherwise increasingly large combinatorial problem.

The condition that terminates the recursion is:

if $X_j = X_n$ (8, in the example)

then $c_{ij} + f(X_j)$ reduces to c_{ij}

Another choice for a termination is:

if $X_i = X_n$

then $f(X_i) = 0$

2 - A NON-RECURSIVE SOLUTION:

We shall present the Bellman-Kalaba algorithm, showing how it can be organized for computer solution. The search for the minimum will suffice.

At iteration 0, numbering the vertices from 0 to n, we pose:

$$f_i^{(0)} = c_{in} \quad , \quad i = 0, 1, 2, \ldots, n-1$$

$$f_n^{(0)} = 0$$

then we compute:

$$f_i^{(1)} = \min_{j \neq i} \left(f_j^{(0)} + c_{ij} \right) \qquad \begin{matrix} i = 0, 1, 2, \ldots, n-1 \\ j = 0, 1, 2, \ldots \end{matrix}$$

$$f_n^{(1)} = 0$$

then, successively:

$$f_i^{(m)} = \min_{j \neq i} \left(f_j^{(m-1)} + c_{ij} \right) \qquad \begin{matrix} i = 0, 1, 2, \ldots, n-1 \\ j = 0, 1, 2, \ldots \end{matrix}$$

$$f_n^{(m)} = 0$$

for all values of k = 1, 2, 3, ... ; we stop when:

$$f_i^{(m)} = f_i^{(m-1)} \quad , i = 0, 1, 2, \ldots , n$$

At this moment $f_0^{(m)}$ represents the value of the minimum path between vertices X_0 and X_n. It can be shown that if there are n + 1 vertices, at most n - 1 iterations will yield the minimum.

The algorithm is equivalent to the solution of the system of equations:

$$f_i = \min_{j \neq i} (f_j + c_{ij}) \quad , i = 0, 1, 2, \ldots , n - 1$$

$$f_n = 0$$

For computer solution, the input takes the form of three arrays nodes-i, nodes-j, values-c_{ij}, whose dimension is the number of edges.

The working area consists of two arrays values-f_i and nodes-ℓ (this one will indicate the path), both with the same dimension as the number of nodes.

The array values-f_i is initialized with a special high value, except for the element corresponding to f_n which is 0.

An iterative process is performed with the following steps:

a - each activity is considered, taking its nodes i and j;

b - if the value of f_j is the special high value no action is taken;

c - otherwise f_i is compared with $f_j + c_{ij}$;

d - if $f_i \geq f_j + c_{ij}$ no action is taken;

e - otherwise:

- f_i is replaced by $f_j + c_{ij}$
- μ_i is replaced by j
- a counter is increased by 1;

f - if after all activities have been considered the counter is not null, the process is repeated from step a after resetting the counter to zero;

g - otherwise the process is terminated.

The value of the minimum path from X_1 to X_n will be found in f_1.

The path itself can be traced by:

$$i \leftarrow \mu_1$$

$$i \leftarrow \mu_i \quad \text{until } i = n$$

Appendix I contains a listing of a simple FORTRAN II program to apply the algorithm.

3 - THE RECURSIVE SOLUTION:

The recursive solution is straightforward since it is the direct application of the recursive formula given at section 1.

The algorithm is indicated in a high level language notation which does not correspond exactly to any existing programming language:

```
f [i] : f ← high value
        S ← list of successors of i
        while S ≠ NIL do
              j ← car [S]
              if j = n then f ← min  f, c_ij
              else f ← min [f, c_ij +
v ← f [1]
```

The path itself can be obtained by a device analogous to the one that was described in the previous section.

Graphs can be represented as lists in the following way:

a - the graph is a list composed of n sub-lists where n is the number of vertices;

b - each sub-list standing for a vertex contains the label of the vertex as its first element ;

c - the remaining elements of a sub-list are the labels of the immediate successors of the vertex.

Moreover the values associated with each edge are given in a list with a corresponding structure: to each successor (in the list representing the graph) there corresponds the value of the edge going from the vertex to that successor (in the list of values). NIL will correspond to the vertices.

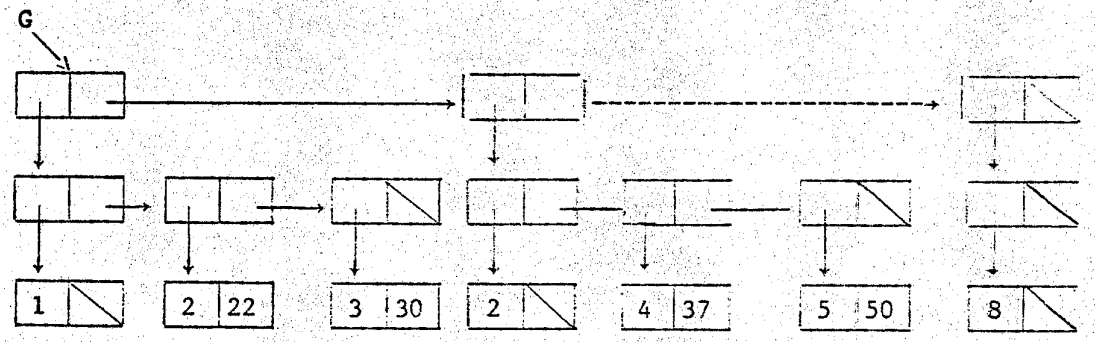
Taking the example of section 1, the network will be represented as:

G = ((1 2 3)(2 4 5)(3 4 6)(4 5)(5 7)(6 7)(8))

and the associated values as:

V = ((() 22 30)(() 37 30)(() 32 43)(() 60)(() 30)
(() 40)(() 50)(()))

A somewhat simplified scheme of the internal representation, after the elements of V have been attached to G, will be:



Appendix II contains a listing of the recursive solution as implemented in the LISP-FORTRAN IV language. This is an extension of FORTRAN IV to include, among other features, recursion and list processing.

Both appendices show the minimum and the maximum solution for the network we have being dealing with.

4 - BIBLIOGRAPHY:

- Bellman, R. - "Dynamic Programming"
  Princetor Univ. Press - 1957.

- Kaufmann, A.- "Méthodes et Modèles de la Recherche Operationnelle"
  tome 2 - Dunod - 1964.

- Barron,D. W.- "Recursive Techniques in Programming" -
  Mac Donald 1969.

APPENDIX I

PROGRAM

```
      DIMENSION IVATIV(50),NOC(50),IVNO(50),NOI(50),NOJ(50)
      READ100,NCASOS
      PUNCH100,NCASOS
      DO13N=1,NCASOS
      READ200,IS,NATIVS,NNOS
      PUNCH200,IS,NATIVS,NNOS
      READ300,(NOI(N),NOJ(N),IVATIV(N),N=1,NATIVS)
      PUNCH300,(NOI(N),NOJ(N),IVATIV(N),N=1,NATIVS)
      DO1N=1,NNOS
    1 IVNO(N)=-999*IS
      IVNO(NNOS)=0
    2 ICONT=0
      DO5N=1,NATIVS
      I=NOI(N)
      J=NOJ(N)
      IF(IABS(IVNO(J))-999)3,5,5
    3 IF((IVATIV(N)+IVNO(J)-IVNO(I))*IS)5,5,4
    4 IVNO(I)=IVATIV(N)+IVNO(J)
      NOC(I)=J
      ICONT=ICONT+1
    5 CONTINUE
      IF(ICONT)6,6,2
    6 IF(IS)8,7,7
    7 PUNCH400,1
      GO TO 9
    8 PUNCH500,1
    9 NDEST=1
   10 NORG=NDEST
      NDEST=NOC(NORG)
      DO11N=1,NATIVS
```

```
      IF(IABS(NOI(N)-NORG)+IABS(NOJ(N)-NDEST))11,12,11
 11 CONTINUE
 12 PUNCH600,NORG,NDEST,N,IVATIV(N)
    IF(NDEST-NNOS)10,13,10
 13 PUNCH700,IVNO(1)
    STOP
100 FORMAT(I3)
200 FORMAT(3I2)
 300 FORMAT(2I2,I3)
400 FORMAT(/////5X,4HCASO,1X,I3,5X,14HCAMINHO MAXIMO,
   , 3X,4HNO I,3X,4HNO J,3X,4HARCO,3X,5HVALOR)
500 FORMAT(5X,4HCASO,1X,I3,5X,14HCAMINHO MINIMO,
   , 3X,4HNO I,3X,4HNO J,3X,4HARCO,3X,5HVALOR)
600 FORMAT(37X,I2,5X,I2,5X,I2,5X,I3)
700 FORMAT(///40X,5HTOTAL,2X,I5)
    END
```

DATA

```
002
-11008
0102022
0103030
0204037
0205030
0304032
0306043
0405060
0507030
0607040
0708050
011008
0102022
0103030
0204037
0205030
0304032
0306043
0405060
0507030
0607040
0708050
```

RESULTS

CASO 1 CAMINHO MINIMO	NO I	NO J	ARCO	VALOR
	1	2	1	22
	2	5	4	30
	5	7	8	30
	7	8	10	50
			TOTAL	132

CASO 2 CAMINHO MAXIMO	NO I	NO J	ARCO	VALOR
	1	3	2	30
	3	4	5	52
	4	5	7	60
	5	7	8	30
	7	8	10	50
			TOTAL	202

APPENDIX II

PROGRAM

```
RECURSIVE F
RECURSIVE PATR
COMMON G,N,K
I=*1*
N=*8*
G=*((1 2 3)(2 4 5)(3 4 6)(4 5)(5 7)(6 7)(7 8)(8))*
V=*((() 22 30)(() 37 30)(() 32 43)(() 60)(() 30)(() 40)(() 50)
, (()))*
CALL PATR(V,G)
K=-1
Z=F(I)
K=1
Z=F(I)
STOP
END

RECURSIVE-INTEGER FUNCTION F(I)
RECURSIVE ASSOC
INTEGER FA,C,CIJ,VALUE,COND
INTEGER OPT
LOGICAL EQ
COMMON G,N,K
C(I,J)=VALUE(CAR(J))
OPT(I,N)=COND(IF(K.GT.0),THEN(MAX0(I,C)),
, IF(.TRUE.),THEN(MIN0(I,N)))
F=COND(IF(K.GT.0),THEN(0),IF(.TRUE.),THEN(99))
S=CDR(ASSOC(I,G))
CALL DO(NOTNIL(S),SET(J,CAR(S)),SET(FA,F),SET(CIJ,C(I,J)),
,                 TRACE(I,J,1),
,  o              COND(IF(EQ(J,N)),THEN(SET(F,OPT(FA,CIJ))),
,                       IF(.TRUE.),
,  o                    THEN(SET(F,OPT(FA,CIJ+F(J, F),CIJ,S))))
,                       ))
CALL TRACE(I,F,2)
RETURN
END
```

```
I=1 J=2

I=2 J=4

I=4 J=5

I=5 J=7

I=7 J=3

F(7)= 50

F(5)= 90

F(4)=140

I=2 J=5

I=5 J=7
```

```
I=7  J=8

F(7)= 50

F(5)= 80

F(2)=110

I=1  J=3

I=3  J=4

I=4  J=5

I=5  J=7

I=7  J=8
```

```
F(7)= 50

F(5)= 80

F(4)=140

I=3 J=6

I=6 J=7

I=7 J=8

F(7)= 50

F(6)= 90

F(3)=133

F(1)=132
```

TRACE (MAXIMUM)

```
I =1  J=2

I =2  J=4

I =4  J=5

I =5  J=7

I =7  J=8

F(7)=  50

F(5)=  80

F(4)=140

I =2  J=5
```

```
F(7)= 50

F(5)= 80

F(4)=140

I=3 J=6

I=6 J=7

I=7 J=8

F(7)= 50

F(6)= 90

F(3)=172

F(1)=202
```