The Elements of Dynamic Programming

Roland Backhouse
School of Computer Science
University of Nottingham

November 13, 2011

Abstract

So-called "dynamic programming" is a technique for constructing efficient solutions to certain optimisation problems and is often discussed in textbooks on algorithms and operations research. This paper presents a simple example of the use of the technique. All components of the technique are detailed with full mathematical rigour.

1 Introduction

So-called "dynamic programming" is commonly discussed in lectures and textbooks on algorithm design but the principles on which it is based are rarely given in a way that allows proper mathematical verification. Confusion is also sometimes caused by the fact that "dynamic programming" invokes several different problem-solving strategies which are not always properly distinguished. As an aid to better understanding, this paper is about the derivation of an algorithm to solve a simple optimisation problem. Because the problem is (deliberately) very simple we are able to explicate all the details of the (mathematically verifiable) derivation, leaving no stone unturned.

We begin in section 2 with the statement of the problem: a simple optimisation problem of maximising a so-called objective function over a constrained solution space. The development then proceeds in section 3 by examining the structure of the solution space: we identify an inductive structure which makes clear how the solution space is broken down into smaller subspaces. (Texts on dynamic programming often refer to "overlapping subproblems".) The same analysis is then applied in section 4 to the optimisation problem itself. This is where we invoke what is called "the principle of optimality" in texts on dynamic programming. The "principle of optimality" is typically a distributivity property; here it is just the property that addition distributes over maximum. In general, the objective function of an optimisation problem often involves "adding" (in some way or other) the value of the objective function on simpler subproblems and the "optimal" value is the

"maximum" (or minimum) value that the function can take. If "addition" distributes over "maximum" then it is the case that the optimal value is composed of optimal subsolutions.

The conclusion of section 4 is an inductive characterisation of the objective function: that is, a set of equations relating the optimal value of the objective function to the optimal value of the objective function on successively simpler subproblems. Section 5 then shows how to solve the equations by exploiting their inductive structure: first solve the basis equations and then use the values already known to determine other values in so-called "topological order". We also show how the optimal solution is found by combining the inductive characterisation of the set of putative solutions with the inductive characterisation of the optimal value.

2 Problem Statement and Abbreviations

Given is a one-dimensional array A of (real) numbers of length N (a natural number at least 0). The array is indexed by numbers i where $0 \leq i < N$. Using dummy S to range over sets of indices, the problem is to calculate

$$\langle \Uparrow S : S \subseteq \{i \mid 0 \leq i < N\} \wedge \langle \forall i :: i \notin S \vee i{+}1 \notin S \rangle : \mathsf{sum}.S \rangle$$

where

$$\mathsf{sum}.S \;=\; \langle \Sigma i : i \in S : A[i] \rangle \quad .$$

(Note that the array A is an implicit parameter of the function sum.)

For convenience, let us abbreviate the components of the problem specification. First, the relation sub is defined on sets S and numbers N by

$$[\; S \;\mathsf{sub}\; N \;\equiv\; S \subseteq \{i \mid 0 \leq i < N\} \;] \quad .$$

Second, the predicate nonCon is defined on sets S by

$$[\; \mathsf{nonCon}.S \;\equiv\; \langle \forall i :: i \notin S \vee i{+}1 \notin S \rangle \;] \quad .$$

(nonCon abbreviates "non-consecutive".) Third, the relation nc is defined on sets S and numbers N by

$$[\; S \;\mathsf{nc}\; N \;\equiv\; S \;\mathsf{sub}\; N \wedge \mathsf{nonCon}.S \;] \quad .$$

Finally, we define the function opt by

$$(1) \qquad [\; \mathsf{opt}.N \;=\; \langle \Uparrow S \;:\; S \;\mathsf{nc}\; N \;:\; \mathsf{sum}.S \rangle \;] \quad .$$

The problem is to derive an algorithm that evaluates $\mathsf{opt}.N$ for given number N.

The form of (1) is typical of an optimisation problem. The problem is parameterised, in this case by a number N and (implicitly) an array of numbers A. Given N, there is a

solution space: the range of the dummy S. The elements of the solution space are called *putative solutions*; typically, the putative solutions must satisfy some constraint. In this case the solution space is the set of sets of numbers that stand in the relation nc to N. The task is to determine a putative solution that optimises some function defined over the solution space; the function is often called the *objective function*. In this case, the function is sum and "optimise" means "maximise".

Note that typically what is required is an optimal putative solution rather than the optimal value of the objective function. When using a route-finder, for example, the user wants to know the route and would not be satisfied with just the information on how long the route is or how long the journey will take. Although (1) does not meet this specification, we will see how constructing the optimal solution can be combined with the evaluation of the optimal value.

3 Solution Space. Inductive Characterisation

The solution space in our problem is the set of sets S that stand in the relation nc to the given number N. The fact that N is a number suggests immediately that we try to express the relation nc by induction on N. That is, we determine nc in the base case when N is 0 and we determine nc in the case that N is $k{+}1$, for some k, in terms of its value for values of N at most k. (In fact, it turns out that the basis we need also includes the case that N is -1.)

Since nc is defined in terms of the relation sub and the predicate nonCon, we begin by finding inductive definitions of each of them.

3.1 The sub relation

First, note that

(2) $\quad [\ \{i \mid 0 \leq i < N\} = \emptyset \ \equiv\ N \leq 0\ ]$.

It follows that

(3) $\quad [\ \langle \forall S :: S \ \mathsf{sub}\ N \ \equiv\ S = \emptyset \rangle \ \equiv\ N \leq 0\ ]$.

Now we consider the inductive step. The first step is a simple case analysis:

$$
\begin{array}{ll}
 & S \ \mathsf{sub}\ (k{+}1) \\
= & \quad \{ \quad [\ k \in S \vee k \notin S\ ] \quad \} \\
 & (k \in S \ \wedge\ S \ \mathsf{sub}\ (k{+}1)) \ \vee\ (k \notin S \ \wedge\ S \ \mathsf{sub}\ (k{+}1))
\end{array}
$$

We deal with the two cases separately. First,

$$
\begin{array}{ll}
 & k \notin S \ \wedge \ S \ \mathsf{sub}\ (k{+}1) \\
= & \quad \{ \quad \text{definition of } \mathsf{sub} \quad \} \\
 & \langle \forall i : i \in S : i \neq k \rangle \ \wedge \ \langle \forall i : i \in S : 0 \leq i < k{+}1 \rangle \\
= & \quad \{ \quad \text{rearranging} \quad \} \\
 & \langle \forall i \ : \ i \in S \ : \ 0 \leq i < k{+}1 \wedge i \neq k \rangle \\
= & \quad \{ \quad [\ i < k{+}1 \ \equiv \ i < k \vee i = k \ ] \quad \} \\
 & \langle \forall i \ : \ i \in S \ : \ 0 \leq i < k \rangle \\
= & \quad \{ \quad \text{definition of } \mathsf{sub} \quad \} \\
 & S \ \mathsf{sub}\ k \ .
\end{array}
$$

Second,

$$
\begin{array}{ll}
 & k \in S \ \wedge \ S \ \mathsf{sub}\ (k{+}1) \\
= & \quad \{ \quad [\ k \in S \ \equiv \ \langle \exists T : k \notin T : S = T \cup \{k\} \rangle \ ] \quad \} \\
 & \langle \exists T : k \notin T : S = T \cup \{k\} \rangle \ \wedge \ S \ \mathsf{sub}\ (k{+}1) \\
= & \quad \{ \quad \text{distributivity of } \wedge \text{ over } \vee, \text{ trading} \quad \} \\
 & \langle \exists T \ : \ k \notin T \wedge S = T \cup \{k\} \ : \ S \ \mathsf{sub}\ (k{+}1) \rangle \\
= & \quad \{ \quad \text{Leibniz, definition of } \mathsf{sub} \quad \} \\
 & \langle \exists T \ : \ k \notin T \wedge S = T \cup \{k\} \ : \ T \ \mathsf{sub}\ (k{+}1) \wedge \{k\} \ \mathsf{sub}\ (k{+}1) \rangle \\
= & \quad \{ \quad [\ k \notin T \ \wedge \ T \ \mathsf{sub}\ (k{+}1) \ \equiv \ T \ \mathsf{sub}\ k \ ] \\
 & \qquad\quad [\, \{k\} \ \mathsf{sub}\ (k{+}1) \,] \quad \} \\
 & \langle \exists T \ : \ k \notin T \wedge S = T \cup \{k\} \ : \ T \ \mathsf{sub}\ k \rangle \\
= & \quad \{ \quad [\ k \notin T \Leftarrow T \ \mathsf{sub}\ k \ ], \text{ trading} \quad \} \\
 & \langle \exists T : S = T \cup \{k\} : T \ \mathsf{sub}\ k \rangle \quad .
\end{array}
$$

We conclude that

$$
(4) \qquad [\ S \ \mathsf{sub}\ (k{+}1) \ \equiv \ S \ \mathsf{sub}\ k \ \vee \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{sub}\ k \rangle \ ] \quad .
$$

3.2 The nonCon predicate

Now we consider the nonCon predicate. Suppose $S \ \mathsf{sub}\ k$. Then

$$
\begin{array}{ll}
 & \mathsf{nonCon}.(S \cup \{k\}) \\
= & \quad \{ \quad \text{definition, trading} \quad \} \\
 & \langle \forall i : i \in S \cup \{k\} : i{+}1 \notin S \cup \{k\} \rangle
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \quad \text{range splitting} \quad \} \\
& \langle \forall i : i{\in}S : i{+}1 \notin S{\cup}\{k\} \rangle \ \ \wedge \ \ \langle \forall i : i{\in}\{k\} : i{+}1 \notin S{\cup}\{k\} \rangle \\
= & \{ \quad \text{1st conjunct: trading;} \\
& \qquad \text{2nd conjunct: one-point rule} \quad \} \\
& \langle \forall i : i{+}1 \in S{\cup}\{k\} : i{\notin}S \rangle \ \ \wedge \ \ k{+}1 \notin S{\cup}\{k\} \\
= & \{ \quad \text{1st conjunct: range splitting, one-pt. rule;} \\
& \qquad \text{2nd conjunct: } [\ S \ \mathsf{sub} \ k \Rightarrow k{+}1 \notin S{\cup}\{k\} \ ] \quad \} \\
& \langle \forall i : i{+}1 \in S : i{\notin}S \rangle \ \ \wedge \ \ k{-}1 \notin S \\
= & \{ \quad \text{trading, definition} \quad \} \\
& \mathsf{nonCon}.S \ \ \wedge \ \ k{-}1 \notin S \ \ .
\end{array}
$$

That is,

$$[\ S \ \mathsf{sub} \ k \ \wedge \ \mathsf{nonCon}.(S{\cup}\{k\}) \ \ \equiv \ \ S \ \mathsf{sub} \ k \ \wedge \ \mathsf{nonCon}.S \ \wedge \ k{-}1 \notin S \ ] \ \ .$$

But (by an easy calculation)

$$[\ S \ \mathsf{sub} \ k \ \wedge \ k{-}1 \notin S \ \equiv \ S \ \mathsf{sub} \ k{-}1 \ ] \ \ .$$

We conclude that

(5) $$[\ S \ \mathsf{sub} \ k \ \wedge \ \mathsf{nonCon}.(S{\cup}\{k\}) \ \ \equiv \ \ S \ \mathsf{sub} \ k{-}1 \ \wedge \ \mathsf{nonCon}.S \ ] \ \ .$$

3.3 The relation nc

Now we are ready to construct an inductive definition of the relation nc.
Basis: Suppose $N \leq 0$. Then

$$
\begin{array}{ll}
& S \ \mathsf{nc} \ N \\
= & \{ \quad \text{definition} \quad \} \\
& S \ \mathsf{sub} \ N \ \ \wedge \ \ \mathsf{nonCon}.S \\
= & \{ \quad \text{assumption } (N \leq 0) \text{ and (3); } \mathsf{nonCon}.\emptyset \quad \} \\
& S = \emptyset \ \ .
\end{array}
$$

That is,

(6) $$[\ (S \ \mathsf{nc} \ N \ \equiv \ S = \emptyset) \ \Leftarrow \ N \leq 0 \ ] \ \ .$$

Induction step.

$$
\begin{array}{ll}
 & S \ \mathsf{nc}\ k{+}1 \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & S \ \mathsf{sub}\ k{+}1 \ \wedge \ \mathsf{nonCon}.S \\
= & \quad \{ \quad (4) \quad \} \\
 & (S \ \mathsf{sub}\ k \ \vee \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{sub}\ k \rangle) \\
 & \wedge \quad \mathsf{nonCon}.S \\
= & \quad \{ \quad \text{distributivity of } \wedge \text{ over } \vee \quad \} \\
 & (S \ \mathsf{sub}\ k \ \wedge \ \mathsf{nonCon}.S) \\
 & \vee \quad \langle \exists T : S = T \cup \{k\} : T \ \mathsf{sub}\ k \wedge \mathsf{nonCon}.S \rangle \\
= & \quad \{ \quad \text{1st disjunct: definition of } \mathsf{nc} \\
 & \qquad \text{2nd disjunct:} \\
 & \qquad\quad \langle \exists T : S = T \cup \{k\} : T \ \mathsf{sub}\ k \wedge \mathsf{nonCon}.S \rangle \\
 & \qquad = \quad \{ \quad \text{Leibniz} \quad \} \\
 & \qquad\quad \langle \exists T : S = T \cup \{k\} : T \ \mathsf{sub}\ k \wedge \mathsf{nonCon}.(T \cup \{k\}) \rangle \\
 & \qquad = \quad \{ \quad (5) \quad \} \\
 & \qquad\quad \langle \exists T : S = T \cup \{k\} : T \ \mathsf{sub}\ k{-}1 \wedge \mathsf{nonCon}.T \rangle \\
 & \qquad = \quad \{ \quad \text{definition} \quad \} \\
 & \qquad\quad \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc}\ k{-}1 \rangle \quad \} \\
 & S \ \mathsf{nc}\ k \ \vee \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc}\ k{-}1 \rangle \quad .
\end{array}
$$

We conclude that

$$
(7) \qquad [\ S \ \mathsf{nc}\ k{+}1 \ \equiv \ S \ \mathsf{nc}\ k \ \vee \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc}\ k{-}1 \rangle \ ] \quad .
$$

3.4 Solutions as Paths in a Graph

The inductive definition of nc can be represented by paths in a graph. See the graph in fig. 1. (The "0"s labelling the upper edges should be read as $\emptyset$.) Note that it is easier to see how the nodes and edges in the graph relate to (7) by rewriting $S \ \mathsf{nc}\ k$ as $\langle \exists T : S = T \cup \emptyset : T \ \mathsf{nc}\ k \rangle$. In this way, (7) becomes

$$
[\ S \ \mathsf{nc}\ k{+}1 \ \equiv \ \langle \exists T : S = T \cup \emptyset : T \ \mathsf{nc}\ k \rangle \ \vee \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc}\ k{-}1 \rangle \ ] \quad .
$$

Each terminating path in the graph defines a set formed by taking the union of all the sets on the edges that form the path. The set of all sets S satisfying $S \ \mathsf{nc}\ k$ is then the set of all sets derived in this way from a terminating path starting at node k.

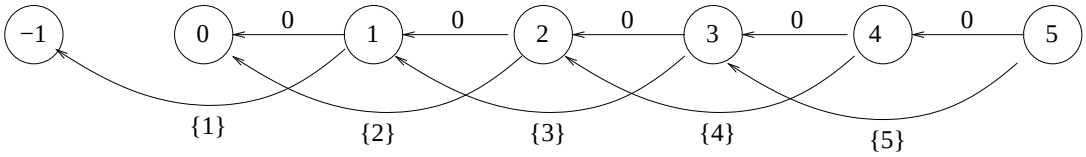

Figure 1: Representing the solution space.

Another way of expressing the inductive definition of nc is as a non-deterministic program. Suppose the task is to generate non-deterministically a putative solution for the case when $N=k$ (i.e. a set S such that S nc k). We can do this for $N=-1$ and $N=0$; by (6) the only choice for S is $\emptyset$. For greater values of k, suppose putative solutions S and T have been found for the cases that $N=k$ and $N=k-1$, respectively; then, by (7) a putative solution for $N=k+1$ is obtained by either choosing S or by choosing $T\cup\{k\}$. This is formulated in the algorithm below. The non-deterministic choice is implemented by a non-deterministic conditional statement: either branch of the statement may be chosen because both guards are true.

$\{ \quad 0 \leq N \quad \}$

$S,T,k \;:=\; \emptyset,\emptyset,0$;

$\{$ **Invariant:** $\quad 0 \leq k \leq N \;\wedge\; S$ nc $k \;\wedge\; T$ nc $k-1 \quad \}$

do $k \neq N \rightarrow$ if true $\rightarrow\; T := S$

$\Box$ true $\rightarrow\; S,T \;:=\; T\cup\{k\}\,,S$

fi ;

$k := k+1$

od

$\{ \quad S$ nc $N \quad \}$

In terms of the graph shown in fig. 1, the algorithm performs a so-called *topological search* of the nodes of the graph. That is, nodes are visited once all their successors have been visited.

Our task now is to transform the non-deterministic choice into a choice of a solution that optimises the objective function. (The choice will still retain an element of non-determinism because there may be several optimal solutions.)

This is a first example of a path-finding algorithm. We shall see several instances of path-finding problems involving different "path algebras". For more background on these topics, see the final chapter of [Bac11].

4 The Objective Function

We now turn to the objective function. The goal is to derive an inductive definition of opt by exploiting the inductive definition of nc that we have just found.

Once again we consider the components in the definition of opt separately.

4.1 Sum

It following calculations are very straightforward.

$$
\begin{array}{ll}
 & \mathsf{sum}.\emptyset \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \langle \Sigma i : i \in \emptyset : A[i] \rangle \\
= & \quad \{ \quad \text{empty range} \quad \} \\
 & 0 \quad .
\end{array}
$$

Suppose $k \notin T$. Then

$$
\begin{array}{ll}
 & \mathsf{sum}.(T \cup \{k\}) \\
= & \quad \{ \quad \text{range splitting} \quad \} \\
 & \mathsf{sum}.T + \mathsf{sum}.\{k\} - \mathsf{sum}.(T \cap \{k\}) \\
= & \quad \{ \quad \text{definition, } [\ k \notin T \ \equiv \ T \cap \{k\} = \emptyset\ ] \quad \} \\
 & \mathsf{sum}.T + \langle \Sigma i : i \in \{k\} : A[i] \rangle - \mathsf{sum}.\emptyset \\
= & \quad \{ \quad \text{one-point rule, empty range} \quad \} \\
 & \mathsf{sum}.T + A[k] \quad .
\end{array}
$$

4.2 opt

Now we are ready for the inductive definition of opt. The basis is numbers N at most 0. Suppose $N \leq 0$. Then

$$
\begin{array}{ll}
 & \mathsf{opt}.N \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \langle \Uparrow S \ : \ S \ \mathsf{nc} \ N \ : \ \mathsf{sum}.S \rangle \\
= & \quad \{ \quad (6) \quad \} \\
 & \langle \Uparrow S : S = \emptyset : \mathsf{sum}.S \rangle \\
= & \quad \{ \quad \text{one-point rule, } \mathsf{sum}.\emptyset = 0 \quad \} \\
 & 0 \quad .
\end{array}
$$

Induction step.

$$
\begin{array}{ll}
 & \mathsf{opt}.(k{+}1) \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \langle \Uparrow S : S \ \mathsf{nc} \ k{+}1 : \mathsf{sum}.S \rangle \\
= & \quad \{ \quad (7) \quad \} \\
 & \langle \Uparrow S \ : \ S \ \mathsf{nc} \ k \ \vee \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc} \ k{-}1 \rangle \ : \ \mathsf{sum}.S \rangle \\
= & \quad \{ \quad \text{range disjunction and definition of } \mathsf{opt} \quad \} \\
 & \mathsf{opt}.k \uparrow \langle \Uparrow S \ : \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc} \ k{-}1 \rangle \ : \ \mathsf{sum}.S \rangle \quad .
\end{array}
$$

We continue the calculation with the second term.

$$
\begin{array}{ll}
 & \langle \Uparrow S \ : \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc} \ k{-}1 \rangle \ : \ \mathsf{sum}.S \rangle \\
= & \quad \{ \quad \text{trading (in preparation for range disjunction)} \quad \} \\
 & \langle \Uparrow S \ : \ \langle \exists T : T \ \mathsf{nc} \ k{-}1 : S = T \cup \{k\} \rangle \ : \ \mathsf{sum}.S \rangle \\
= & \quad \{ \quad \text{range disjunction} \quad \} \\
 & \langle \Uparrow T \ : \ T \ \mathsf{nc} \ k{-}1 \ : \ \langle \Uparrow S : S = T \cup \{k\} : \mathsf{sum}.S \rangle \rangle \\
= & \quad \{ \quad \text{one-point rule} \quad \} \\
 & \langle \Uparrow T : T \ \mathsf{nc} \ k{-}1 : \mathsf{sum}.(T \cup \{k\}) \rangle \\
= & \quad \{ \quad [\ T \ \mathsf{nc} \ k{-}1 \Rightarrow k \notin T \ ]), \text{ section } 4.1 \quad \} \\
 & \langle \Uparrow T : T \ \mathsf{nc} \ k{-}1 : \mathsf{sum}.T + A[k] \rangle \\
= & \quad \{ \quad \text{distributivity of addition over max} \\
 & \qquad \text{(the ``principle of optimality'')} \quad \} \\
 & \langle \Uparrow T : T \ \mathsf{nc} \ k{-}1 : \mathsf{sum}.T \rangle \ + A[k] \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \mathsf{opt}.(k{-}1) \ + \ A[k]
\end{array}
$$

Combining the calculations, we have thus derived the equations:

(8) $\quad [\ \mathsf{opt}.N = 0 \ \Leftarrow \ N \leq 0 \ ]$

and, for $0 \leq k$

(9) $\quad [\ \mathsf{opt}.(k{+}1) \ = \ \mathsf{opt}.k \uparrow (\mathsf{opt}.(k{-}1) \ + \ A[k]) \ ] \quad .$

A crucial step in the above calculation is the penultimate step above in which the distributivity of addition over maximum is used to factor out "$+ A[k]$" from the computation of

$$\langle \Uparrow S \ : \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc} \ k{-}1 \rangle \ : \ \mathsf{sum}.S \rangle \quad .$$

In accounts of dynamic programming, this is called the *principle of optimality*. The principle is often phrased in words as "optimal solutions are composed of optimal subsolutions". The formal application of the principle is a combination of breaking the solution space down into subspaces and then exploiting a distributivity property of the operation used to evaluate individual spaces (in this case sum) over the optimisation operator (in this case maximum).

The solution of equations (8) and (9) can also be represented as a path-finding problem — this time as finding a longest path in a graph.

Fig. 2 illustrates the general structure of the graph. The length of a path is the sum of the labels of the edges forming the path. Calculation of $\mathsf{opt}.k$ for a given value of k is equivalent to finding a longest terminating path in the graph starting at the node labelled k.

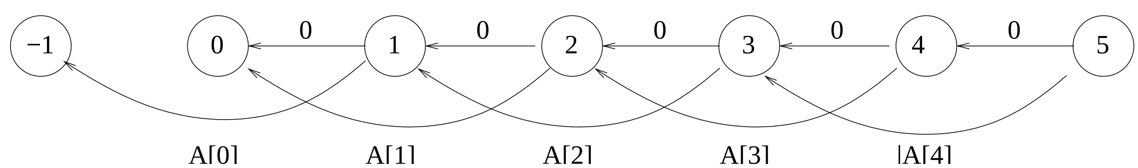

Figure 2: Longest-path Problem

Once again it is easier to see how the nodes and edges in the graph relate to (9) by rewriting it as

$$[\ \mathsf{opt}.(k{+}1) \quad = \quad (\mathsf{opt}.k + 0)\uparrow(\mathsf{opt}.(k{-}1) + A[k])\ ] \quad .$$

5 Topological Search

The final component of "dynamic programming" is the use of topological search to solve an acyclic system of equations.

For this problem, the method is straightforward: solve the equations for $\mathsf{opt}.k$ in increasing order of k. The algorithm is shown below.

$\{\quad 0 \le N \quad\}$

$k,m,n\ :=\ 0,0,0\ ;$

$\{\ \ \textbf{Invariant:}\quad 0 \le k \le N\ \wedge\ m = \mathsf{opt}.k \wedge n = \mathsf{opt}.(k{-}1)\quad\}$

$\mathsf{do}\ k \ne N \to \quad \mathsf{if}\quad m \ge n{+}A[k]\ \to\ n := m$

$\qquad\qquad\qquad\ \ \Box\quad m \le n{+}A[k]\ \to\ m,n := n{+}A[k]\,,m$

$\qquad\qquad\qquad\ \ \mathsf{fi}\ \ ;$

$\qquad\qquad\qquad\ \ k := k{+}1$

od

$\{ \quad S \ \mathsf{nc} \ N \ \wedge \ m = \mathsf{sum}.S = \mathsf{opt}.N \quad \}$

This program only calculates the optimal value and does not compute an optimal solution. To record an optimal solution, we combine the calculation of $\mathsf{opt}.N$ with the calculation of putative solutions given in section 3.3. The combination has five variables k, m, n, S and T. The set S "witnesses" the fact that m equals $\mathsf{opt}.k$ in the sense that S stands in the relation nc to k and $m = \mathsf{sum}.S$. Similarly, T witnesses the fact that n equals $\mathsf{opt}.(k-1)$.

$\{ \quad 0 \leq N \quad \}$

$k,m,n,S,T \ := \ 0,0,0,\emptyset,\emptyset \ ;$

$\{$ **Invariant:**

$\quad 0 \leq k \leq N$

$\wedge \quad S \ \mathsf{nc} \ k \ \wedge \ m = \mathsf{opt}.k = \mathsf{sum}.S$

$\wedge \quad T \ \mathsf{nc} \ k-1 \ \wedge \ n = \mathsf{opt}.(k-1) = \mathsf{sum}.T \quad \}$

$\mathsf{do} \ k \neq N \rightarrow \quad \mathsf{if} \quad m \geq n+A[k] \rightarrow n,T := m,S$

$\quad \square \quad m \leq n+A[k] \ \rightarrow \ m,n,S,T \ := \ n+A[k] \, , m \, , T \cup \{k\} \, , S$

$\quad \mathsf{fi} \ ;$

$\quad k := k+1$

od

$\{ \quad S \ \mathsf{nc} \ N \ \wedge \ m = \mathsf{sum}.S = \mathsf{opt}.N \quad \}$

Note that there is still an element of non-determinism in the algorithm: if $m = n+A[k]$ a non-deterministic choice is made between assigning $T \cup \{k\}$ to S or leaving the value of S unchanged. This reflects the non-determinism in the problem specification.

6 Efficiency

The algorithm we have developed very obviously takes time proportional to N. Indeed, the number of maximum and addition operations used is directly related to the number of edges in the graph of size N illustrated by fig. 3. An alternative method of calculating $\mathsf{opt}.N$ is to use equations (8) and (9) in a so-called "recursive" solution of the problem. This is equivalent to enumerating all putative solutions. It is interesting to calculate just how many addition and maximum operations this would entail. This can also be expressed as a path-finding problem: it is the problem of calculating the number of terminating paths starting from node N in the graph illustrated by fig. 3.

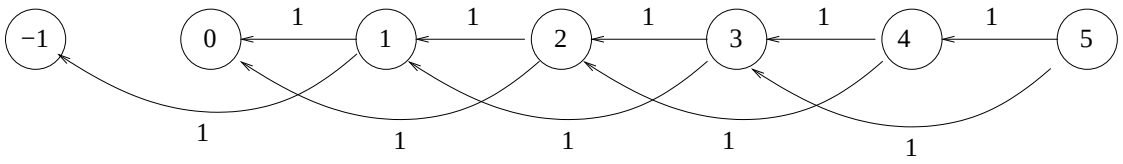

Figure 3: Counting Paths

To see why this is the case, consider again the equations (6) and (7). Let $\mathsf{SolSpaceSize}.k$ denote the number of putative solutions when the parameter N has the value k. That is,

(10) $\quad [\ \mathsf{SolSpaceSize}.k \ = \ \langle \Sigma S : S \ \mathsf{nc}\ k : 1 \rangle\ ]$.

Then, immediately from (6), we have:

(11) $\quad [\ \mathsf{SolSpaceSize}.N = 1 \ \Leftarrow \ N \leq 0\ ]$.

Moreover, for all k such that $0 \leq k$,

$$
\begin{array}{ll}
 & \mathsf{SolSpaceSize}.(k+1) \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \langle \Sigma S : S \ \mathsf{nc}\ (k+1) : 1 \rangle \\
= & \quad \{ \quad (7) \quad \} \\
 & \langle \Sigma S : S \ \mathsf{nc}\ k \ \vee \ \langle \exists T : S = T \cup \{k\} : T \ \mathsf{nc}\ k-1 \rangle \ : 1 \rangle \\
= & \quad \{ \quad \text{range splitting and one-point rule} \quad \} \\
 & \langle \Sigma S : S \ \mathsf{nc}\ k : 1 \rangle \ + \ \langle \Sigma T : T \ \mathsf{nc}\ k-1 : 1 \rangle \\
= & \quad \{ \quad \text{definition} \quad \} \\
 & \mathsf{SolSpaceSize}.k \ + \ \mathsf{SolSpaceSize}.(k-1) \ .
\end{array}
$$

That is,

(12) $\quad [\ \mathsf{SolSpaceSize}.(k+1) \ = \ \mathsf{SolSpaceSize}.k \ + \ \mathsf{SolSpaceSize}.(k-1)\ ]$.

(In fig. 3, the edges are labelled with the number 1. The relation between the edge labels and (12) becomes clear by writing the right side of the equation as

$$\mathsf{SolSpaceSize}.k \times 1 \ + \ \mathsf{SolSpaceSize}.(k-1) \times 1 \ .$$

The path algebra for counting paths is a combination of multiplication and addition. If there are m paths from node u to node v and n paths from node v to node w then there are $m \times n$ paths from node u to node w. Moreover, where there is a choice of paths, the number of paths is obtained by adding the counts for each possible choice.)

The solution to equations (11) and (12) is well-known; it is the so-called *Fibonacci function*. The rate of growth of the Fibonacci function for increasing values of the parameter k is not as fast as 2^k, but almost. In any case, it is exponentially increasing. The improvement in efficiency obtained by using a topological search to evaluate opt is therefore very substantial. (The same can be said for evaluating the Fibonacci function for a given argument k. The so-called "top-down recursive" program uses an exponential number of additions whereas the "bottom-up" topological-search algorithm uses a linear number of additions.)

7 Conclusion

So-called "dynamic programming" is an important topic in operations research that is best understood as a combination of different problem-solving principles. This note is about a (deliberately) very simple example in which the components of dynamic programming are clearly separated. It also provides a detailed illustration of the use of the quantifier calculus.

Optimisation problems are about optimising some objective function that is defined over a set of putative solutions satisfying some constraint. The first task is to break the solution space down into subspaces. The next task is to exploit the subspace structure of the solution space in combination with distributivity properties of the components of the objective function to similarly break down the evaluation of the objective function. If the first two steps result in an *acyclic* system of equations, the equations can be solved in topological order. The (best- and worst-) case running time of the resulting algorithm is then linear in the number of terms in the system of equations that are to be solved. Finally, if care is taken to relate the structure of the solution space to the evaluation of the objective function, it is easy to incorporate the calculation of an optimal solution into the calculation of the optimal value of the objective function. Using, as we have done, a non-deterministic algorithm to describe the construction of putative solutions helps to clarify the process.

Acknowledgement The problem presented here is a simplified version of the "Planning a Company Party" problem given as an exercise in [CLR90]. I learnt of the exercise from [BdM97].

References

[Bac11] Roland Backhouse. *Algorithmic Problem Solving.* John Wiley & Sons, 2011.

[BdM97] Richard S. Bird and Oege de Moor. *Algebra of Programming.* Prentice-Hall International, 1997.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Electrical Engineering and Computer Science Series, MIT Press, 1990.