ELIMINATION OF RECURSIVE CALLS USING
A SMALL TABLE OF "RANDOMLY" SELECTED
FUNCTION VALUES

J. HILDEN

Abstract.

This case study concerns the use of a table of selected function values to avoid
repeated function evaluations and, particularly, to speed up recursive ones. It is
assumed that the table cannot hold all repeatedly needed function values owing
to storage limitations ("small-table technique"). The programmer is then faced
with the problem of finding a table management policy that reduces repeat evalua-
tions to a minimum, a problem which must usually be tackled by heuristic means.
We test and discuss a spectrum of policies, most of which involve "random"
selection of function values for tabulation. Savings of 95% or more are easily
achieved, apparently even in the limit as the computational burden increases.
Policies involving table search proper are noted to be inferior. Sometimes intuiti-
vely reasonable policy refinements fail, so efficient policy selection must be based
on experiments of the kind presented here. Several other programming recommend-
dations are made, as well as suggestions for theoretical research.

Key Words: program optimization, look-up, search, hash tables, recursion,
small-table technique.

0. Introduction.

Sometimes it is impossible to employ the familiar device of table
look-up to speed up the evaluation of complicated recursive functions
because the required table of function values would become impracti-
cally large. By keeping a smaller table of selected values it may still
be possible to save computer time. The present case study tries to throw
light on how this selection ought to be carried out in the troublesome —
but typical — case in which the programmer cannot predict which func-
tion values most profitably could be kept in the table. Various table
management policies (or table policies, for short) are tried on a recursive
function definition of the type that abounds in combinatorics. The study
shows that much computing can be saved with any reasonable table
policy and a fairly small table. When it comes to refining the policy,
some intuitively promising policy variants do not seem to work as well
as their competitors, and conversely. This indicates that efficient use of
this programming technique must be based on experiments.

Received August 30, 1975.
1. Problem definition.

Let us map the subject area more carefully by listing four assumptions concerning the function and its intended use.

(a) A fair proportion of the function calls will have duplicate arguments. Otherwise there is no point in constructing a table.

(b) It is decided to fill in the table dynamically. This decision will typically be made tentatively in order to examine its consequences, the alternatives being a complete table computed in advance or no table at all.

Dynamical table construction means that each time the function is called the program looks for the value in the table; if it is not there the function is evaluated and the value stored. Many recursive combinatorial functions are ideal candidates for this scheme, as discussed in Barron’s book ([2]). The programming is straightforward, provided that a complete table can be kept in fast store. Complications arise when

(c) the range of argument values of current interest is so large that a complete table is out of question. To be precise, it is assumed that it is not possible to keep the function values in an array indexed by the argument, not even to store the actually computed subset in some kind of hash table or other fast search store.

In that event, one may contemplate keeping some of these values in a smaller table. Again we address ourselves to the worst case by assuming that

(d) it is impossible to select analytically those function values which would entail maximum savings. That is to say, good analytic table admission rules are not available — or are as hard to compute as the function itself.

If so, the table policy must be based on guesses or chosen empirically. We shall refer to the programming technique that employs a small table along these lines as the small-table technique.

2. The function.

The experience we are going to present derives from a study of the following function:

\( A(s,n,m) = \begin{cases} 0, & (s < v \lor s > \mu) \\ 1, & (s = v \lor s = \mu) \\ A(s-n-m,n-1,m) + A(s,n,m-1) \end{cases} \)

where \( s, n, m \) are integer arguments, \( n \) and \( m \) non-negative, and \( v \) and \( \mu \) are defined by

\( v \equiv n(n+1)/2, \quad \mu \equiv n(n+1)/2 + nm. \)