A GENERALIZATION OF DIJKSTRA'S ALGORITHM *

Donald E. KNUTH
Computer Science Department, Stanford University, Stanford, California 94305, U.S.A.

Received 26 July 1976

Shortest paths, AND/OR graphs, context-free grammars, simultaneous minimax equations, decision trees, dynamic programming

E.W. Dijkstra [3] has introduced an important algorithm for finding shortest paths in a graph, when the distances on each arc of the graph are nonnegative. The purpose of this note is to present an algorithm which generalizes Dijkstra's in essentially the same way that tree structures generalize linear lists, or that context-free languages generalize regular languages.

1. The problem in general

Let R_+ be the nonnegative real numbers extended with the value $+\infty$. We shall say that a function $g(x_1, ..., x_k)$ from R_+^k into R_+ is a *superior function* if it is monotone nondecreasing in each variable and if

$$g(x_1, ..., x_k) \geqslant \max(x_1, ..., x_k) \quad \text{for all } x_1, ..., x_k .$$

When $k = 0$ the function g is simply a constant element of R_+, and it is trivially considered to be a superior function. When $k = 2$ several familiar functions $g(x, y)$ such as $\max(x, y)$ and $x + y$ and $\max(1, x) \cdot \max(1, y)$ are all superior. Any functional composition of superior functions is a superior function of the variables occurring in the composition.

A *superior context-free grammar* is a context-free grammar in which all productions are of the general form

$$Y \to g(X_1, ..., X_k) ,$$

where $Y, X_1, ..., X_k$ are nonterminal symbols, g is a terminal symbol corresponding to a superior function (possibly a different superior function for each production), and the parentheses and commas are also terminal symbols. If $k > 0$, there are $k - 1$ commas; if $k = 0$ the right-hand side of the production is simply written "g", a terminal symbol which corresponds to a nonnegative real constant.

For example, the following productions define a superior context-free grammar on the nonterminal symbols $A\,B\,C$ and the terminal symbols $(\,),a\,b\,c\,d\,e\,f$:

$$A \to a , \qquad B \to c(A) , \qquad C \to e ,$$
$$A \to b(B, C) , \qquad B \to d(A, C, A) , \qquad C \to f(B, A) .$$

Here a, b, c, d, e, f are supposed to correspond to superior functions, and we may for example define

$$a = 4 , \qquad d(x, y, z) = x + \max(y, z) ,$$
$$b(x, y) = \max(x, y) , \qquad e = 9 ,$$
$$c(x) = x + 1 , \qquad f(x, y) = \tfrac{1}{2}(x + y + \max(x, y)) .$$

For every nonterminal symbol Y of a superior context-free grammar over the terminal alphabet T we let

$$L(Y) = \{\alpha \mid \alpha \in T^* \text{ and } Y \to^* \alpha\}$$

be the set of terminal strings derivable from Y. Every string α in $L(Y)$ is a composition of superior functions, so it corresponds to a uniquely-defined nonnegative real number which we shall call $\mathrm{val}(\alpha)$. Thus, in the above example we have

$$L(A) = \{a, b(c(a), e), b(c(a), f(c(a), a)), b(c(b(c(a), e)), e), ...\}$$

* This research was supported in part by National Science Foundation grant MCS 72-03752 A03, by the Office of Naval Research contract N00014-76-C-0330, and by IBM Corporation. Reproduction in whole or in part is permitted for any purpose of the United States Government.

and the corresponding numerical values are

$\{4, 6, 7, 10, ...\}$.

The problem we shall solve is to compute the *smallest* values corresponding to these languages, namely

$$m(Y) = \min \{\text{val}(\alpha) | \alpha \in L(Y)\},$$

for each nonterminal symbol Y.

2. Applications of the general problem

(A) Consider $n+1$ cities $\{c_0, c_1, ..., c_n\}$ connected by a network of roads. For each road from c_i to c_j, introduce the production

$$C_i \to g_{ij}(C_j),$$

and the corresponding superior function

$$g_{ij}(x) = d_{ij} + x,$$

where $d_{ij} \geq 0$ is the length of the road. Then add one more production

$$C_0 \to 0,$$

where 0 corresponds to the constant function zero. In this superior context-free grammar, the elements of $L(C_i)$ correspond to the paths from c_i to c_0, and the corresponding values will be the lengths of those paths. For example, if there is a path from c_3 to c_4 to c_2 to c_0, one of the elements of $L(C_3)$ will be $g_{34}(g_{42}(g_{20}(0)))$, and its corresponding value is $d_{34} + d_{42} + d_{20} + 0$. Therefore $m(C_i)$ is the length of the *shortest path* from c_i to c_0, for all i.

The algorithm we shall develop for the general problem reduces to Dijkstra's algorithm in this speical case.

(B) Given a context-free grammar, replace each production $Y \to \theta$ in which the nonterminal symbols of string θ are $X_1, ..., X_k$ from left to right (including repetitions) by

$$Y \to g_\theta(X_1, ..., X_k),$$

where

$$g_\theta(x_1, .., x_k) = x_1 + ... + x_k$$

+ (the number of terminal symbols in θ).

Then $m(Y)$ is the length of the shortest string derivable from Y in the given grammar. Alternatively if we let

$$g_\theta(x_1, ..., x_k) = \max(x_1, ..., x_k) + 1,$$

then $m(Y)$ is the minimum height of a parse tree for a string derivable from Y in the given grammar.

Finally, if we let

$$g_\theta(x_1, ..., x_k) = \max(x_1, ..., x_k).$$

we have $m(Y) = 0$ or ∞ according as $L(Y)$ is nonempty or empty. The algorithm we shall develop for the general problem reduces to the classical emptiness-testing algorithms of Bar-Hillel, Perles, and Shamir [2] or Greibach [5] in this special case.

(C) Consider the AND/OR graphs which arise in artificial intelligence applications as in Nilsson [9], section 4–5. In this case we use one nonterminal symbol for each "problem" to be solved. If some problem-reduction operator shows that problem Y could be solved if we could solve all of problems $X_1, ..., X_k$, then we introduce the production

$$Y \to g(X_1, ..., X_k),$$

where $g(x_1, ..., x_k) = x_1 + ... + x_k +$ (cost of solving Y given solutions to $X_1, ..., X_k$). Then $m(Y)$ is the minimum cost of a solution to Y, provided that the cost of solving common subproblems is replicated (all problem solutions are considered independent). Our algorithm will therefore find a smallest AND/OR graph in this sense; but it is of limited utility for A.I. applications because it deals with the set of *all* "easy" subproblems, and this set is usually too large.

A special case of the algorithm to be described here has been published by Martelli and Montanari. They deal only with AND/OR graphs whose functions $g(x_1, ..., x_k)$ have the form $x_1 + ... + x_k + c$ with $c > 0$; furthermore they restrict their discussion to acyclic AND/OR graphs. The algorithm below works also in the presence of cycles, and in this sense it is more general than the usual "dynamic programming" approach.

(D) given $2n+1$ probabilities $p_1, ..., p_n, q_0, ..., q_n$ which sum to 1, construct the context free grammar with nonterminal symbols $C_{i,j}$ for $0 \leq i \leq j \leq n$, where the productions are

$C_{i,i} \to 0$ for $0 \leq i \leq n$,

$C_{i,j} \to g_{ij}(C_{i,k-1}, C_{k,j})$ for $0 \leq i < k \leq j \leq n$;

and let

$$g_{ij}(x, y) = x + y + p_{i+1} + \ldots + p_j + q_i + \ldots + q_j .$$

Then $m(C_{0,n}) + p_1 + \ldots + p_n$ is the expected number of comparisons in an *optimum binary search tree* define by the given probabilities, in the sense of [6, 434–435]. The general algorithm we shall describe here is not competitive with the special one in [6]; but it appears to be useful in connection with similar problems, such as that of constructing optimum programs for decision tables.

(E) The author has successfully used this approach to generalize the optimum code-generation algorithm of Aho and Johnson [1], treating the case of machines with asymmetric registers.

3. The general algorithm

Given a superior context-free grammar, the following algorithm determines $m(Y)$ for all nonterminal Y. The algorithm operates on elements $\mu[Y]$ and $\nu[Y]$ for each Y; initially all these elements are undefined, but when the algorithm terminates $\mu[Y]$ will equal $m(Y)$ for all Y. The set D represents those Y for which $\mu[Y]$ has been defined.

1) Set D to the empty set.
2) If all nonterminals are in D, stop.
3) For each nonterminal $Y \notin D$, compute

$$\nu[Y] = \min \{g(\mu[X_1], \ldots, \mu[X_k]) \mid Y \to g(X_1, \ldots, X_k) \text{ is a production and } \{X_1, \ldots, X_k\} \subseteq D\} .$$

(If this set is empty, $\nu[Y]$ is infinite.)
4) Choose $Y \notin D$ such that $\nu[Y]$ is minimum, and set $\mu[Y] \leftarrow \nu[Y]$.
5) Include Y in D, and return to step 2).

For example, consider the grammar given in sect. 1. The computation proceeds as in table 1, viewed in step 3). Finally $\mu[C] \leftarrow 7$.

Actually the values $\nu[Y]$ do not need to be computed explicitly, they have been introduced here only for convenience in stating and proving the algorithm. By maintaining a priority queue of the current values of $\mu[Y]$ with $Y \notin D$, the running time of this algorithm is bounded by a constant times $m \log n + t$, where there are m productions and n nonterminals, and the total length of all productions is t. Further refinements

Table 1

D	$\mu[A]$	$\mu[B]$	$\mu[C]$	$\nu[A]$	$\nu[B]$	$\nu[C]$
$\emptyset$	–	–	–	4	∞	9
$\{A\}$	4	–	–	–	5	9
$\{A,B\}$	4	5	–	–	–	7

are also possible, since all productions with Y on the left-hand side can be deleted from memory as soon as $\mu[Y]$ has been defined.

4. Proof of the algorithm

We must show that $\mu[Y] = m(Y)$ whenever step 5) is performed, since the relation

$$(*)\ X \in D \text{ implies } \mu[X] = m(X) ,$$

will then be invariant throughout the algorithm.

It is clear that $\nu[Y] \geqslant m(Y)$ in step 3); for if $\nu[Y] < \infty$, (*) implies that

$$\nu[Y] = \text{val}(g(\alpha_1, \ldots, \alpha_k)), \text{ where}$$

$$Y \to g(X_1, \ldots, X_k) \to^* g(\alpha_1, \ldots, \alpha_k) ,$$

and the terminal strings $\alpha_1, \ldots, \alpha_k$ satisfy $\text{val}(\alpha_i) \geqslant m(X_i)$ for all i. Therefore the algorithm will be valid unless $\mu[Y] > m(Y)$ at some occurrence of step 5). In that case there will exist a terminal string α such that $Y \to^* \alpha$ and $\text{val}(\alpha) < \mu[Y]$.

Assume that α is the shortest terminal string such that $Z \to^* \alpha$ for some nonterminal $Z \notin D$ and such that $\text{val}(\alpha) < \mu[Y]$, at some occurrence of step 5). Then $\alpha = g(\alpha_1, \ldots, \alpha_k)$ for some $\alpha_1, \ldots, \alpha_k$, where the grammar contains the production $Z \to g(X_1, \ldots, X_k)$ and $X_i \to^* \alpha_i$ for all i. If $\{X_1, \ldots, X_k\} \subseteq D$ we have $\text{val}(\alpha_i) \geqslant m(X_i) = \mu[X_i]$ for all i, by definition of $m(X_i)$ and (*), hence $\text{val}(\alpha) \geqslant g(\mu[X_1], \ldots, \mu[X_k])$, by the monotonicity of g in each variable. But $g(\mu[X_1], \ldots, \mu[X_k]) \geqslant \nu[Z]$ by step 3), and $\nu[Z] \geqslant \nu[Y] = \mu[Y]$ by step 4), contradicting $\text{val}(\alpha) < \mu[Y]$. Therefore $X_i \notin D$ for some i. But now $\text{val}(\alpha_i) \leqslant \text{val}(\alpha)$ by the superiority of g, hence α_i is a shorter string having the stated properties of α; this contradiction completes the proof.

It is easy to prove that the algorithm defines the $\mu[Y]$ in increasing order of their value, since the new

element Y introduced into D by steps 4) and 5) cannot make any of the ν's less than $\mu[Y]$ in the next occurrence of step 3). The usual proofs of Dijkstra's algorithm rely on this monotonicity property; but the above proof shows that it isn't really a crucial fact, even when the algorithm has been substantially generalized.

5. Another application

If the functions g of a superior context-free grammar satisfy the additional condition of strict inequality,

$$g(x_1, ..., x_k) > \max(x_1, ..., x_k),$$

we shall prove that there is a *unique* solution to the set of simultaneous equations

$$f(Y) = \min \{g(f(X_1), ..., f(X_k)) \mid Y \to g(X_1, ..., X_k) \text{ is a production}\},$$

for all nonterminal Y,

namely $f(Y) = m(Y)$ for all Y.

In the first place, $m(Y)$ is a solution: For if $Y \to g(X_1, ..., X_k)$ is any production, then $g(m(X_1), ..., m(X_k))$ is infinite or equal to val (α) for some terminal string α, where

$$Y \to g(X_1, ..., X_k) \to^* g(\alpha_1, ..., \alpha_k) = \alpha$$

for some $\alpha_1, ..., \alpha_k$. Thus $g(m(X_1), ..., m(X_k)) \geq m(Y)$ by definition of $m(Y)$. Conversely if $m(Y) = \text{val}(\alpha)$ then α has the form $g(\alpha_1, ..., \alpha_k)$, where there is a production $Y \to g(X_1, ..., X_k)$; and

$$g(m(X_1), ..., m(X_k)) \leq g(\text{val}(\alpha_1), ..., \text{val}(\alpha_k)) = \text{val}(\alpha) = m(Y).$$

In the second place, the solution is unique: Suppose f_1 and f_2 are distinct solutions to the simultaneous minimization equations above. Let Y be a nonterminal such that $f_1(Y) \neq f_2(Y)$ and $\min(f_1(Y), f_2(Y))$ is as small as possible. Without loss of generality assume that $f_1(Y) < f_2(Y)$; then $f_1(X) = f_2(X)$ for all X such that $f_1(X) < f_1(Y)$. There must exist a production

$Y \to g(X_1, ..., X_k)$ such that

$$g(f_1(X_1), ..., f_1(X_k)) = f_1(Y) < f_2(Y) \leq g(f_2(X_1), ..., f_2(X_k)).$$

But the strict inequality condition above implies that $f_1(X_i) < f_1(Y)$ for $1 \leq i \leq k$, hence $f_1(X_i) = f_2(X_i)$ for all i by our construction, and this is impossible.

Note that something like the strict inequality condition is necessary to guarantee uniqueness, because of the following example grammar:

$A \to a \qquad a = 5,$

$A \to b(B), \qquad b(x) = x,$

$B \to c(A), \qquad c(x) = x.$

The simultaneous minimization equations are

$$f(A) = \min(5, f(B)), \qquad f(B) = f(A),$$

hence the solutions are

$$f(A) = f(B) = x \quad \text{for } 0 \leq x \leq 5.$$

The application discussed here arises for example in the study of "levels" in a flowchart, as defined by Floyd [4] and Mont-Reynaud [8].

Acknowledgments

I wish to thank Edsger W. Dijkstra and Nils J. Nilsson for their helpful comments on the first draft of this paper.

References

[1] A.V. Aho and S.C. Johnson, Optimal code generation for expression trees, J. Assoc. Comput. Mach. 23 (1976) 488–501.

[2] Y. Bar-Hillel, M. Perles and E. Shamir, On formal properties of simple phase structure grammars, Z. f. Phonetic, Sprachwissenschaft und Kommunikationsforschung 14 (1961) 143–172, CR 1450 (September-October 1963) 213–214. Reprinted in Yehoshua Bar-Hillel, Language and Information (Addison–Wesley, Reading, MA, 1964) 116–150.

[3] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.

[4] R.W. Floyd, Flowchart levels (preliminary draft), unpublished manuscript (July 1965).

[5] Sheila Greibach, Inverses of phase structure generators, Math. Linguistics and Automatic Translation Report NSF-11, Harvard University (June 1963).
[6] Donald E. Knuth, The Art of Computer Programming, Vol. 3, Sorting and Searching (Addison-Wesley, Reading, MA, 1973).
[7] A. Martelli and U. Montanari, Additive AND/OR graphs, Advance Papers of the Third International Joint Conference on Artificial Intelligence (1973) 1-11.
[8] B. Mont-Reynaud, personal communication (July 1976).
[9] Nils J. Nilsson, Problem-Solving Methods in Artificial Intelligence (McGraw-Hill, New York, 1971).