Theoretical Computer Science 14 (1981) 79-90
North-Holland Publishing Company

OPTIMAL ALGORITHMS FOR SENSITIVITY ANALYSIS IN ASSOCIATIVE MULTIPLICATION PROBLEMS*

Arnon ROSENTHAL**
Department of Computer and Communication Sciences, The University of Michigan, Ann Arbor, MI 48109, U.S.A.

Communicated by R. Karp
Received November 1978
Revised November 1979

Abstract. We consider efficient ways of determining the sensitivity of a product to changes in individual factors. The task is motivated by several interesting combinatorial and numeric problems which can be given a unified formulation as the problem of finding the (associative) product of N objects. Both deterministic and probabilistic changes to the factors are considered. Algorithms for two kinds of deterministic variation schemes are considered. Nontrivial lower bounds are obtained which demonstrate the algorithms to be optimal. For probabilistic choice of the parameter to be varied, it is shown that optimal ordered binary search trees or Huffman trees determine the optimal strategies. A number of unsolved problems are posed.

1. Introduction

We consider a number of apparently different applications for which parameter variation studies are performed. By 'parameter variation', we mean that each individual input will be changed, and the effect of this change on the problem solution noted.

Parameter variation studies (or equivalently, *sensitivity analyses*) are necessary in many optimization and other computational problems. When the input data is known to be unreliable, it is desirable to study how the optimal value changes if some of the data is altered. Such sensitivity analysis procedures may exploit duality [12] or linearity [6] in the problem domain, or special properties of certain algorithms for graphical computations [3, 4, 13], or of certain branch and bound algorithms [5]. Nonserial associativity is exploited in [10, 11]. [15] exploits the simple structure of regular languages.

We consider one-variable-at-a-time sensitivity analysis on a very simple kind of computation: multiplying N quantities $Z_1, \ldots, Z_N$ in some arbitrary associative multiplication structure (i.e. semigroup). Given new values $Z'_1, \ldots, Z'_N$, define

* This work was partially supported by a grant from Horace H. Rackham Graduate School and by National Science Foundation MCS77-01753.
** Current address: Sperry Research Center, Sudbury, MA 01776, U.S.A.

variants by: $\text{Variant}_k \equiv (\prod_{i<k} Z_i) Z'_k (\prod_{i>k} Z_i)$. Variant_0 will denote $\prod_{i=1}^{N} Z_i$. The *Type I sensitivity analysis problem is*: Compute $\{\text{Variant}_k \mid k = 0, 1, \ldots, N\}$.

Note that there is no scalar measure of sensitivity; rather, we simply output all the variants. The product operation is assumed only to be associative.

2. Applications

(1) *Error correction*: If $a_1, \ldots, a_N$ is an input string of a finite automaton with transition function δ, δ_{a_i} can be defined as the transition function corresponding to character a_i. (That is, if q is a state of the machine, then $\delta_{a_i}(q) = (q, a_i)$.)

We can then extend our definition so that $\delta_{a_j \cdots a_k}$ is the transition function corresponding to the string $a_j \cdots a_k$. Our goal is to compute the transition function corresponding to the input string $a_1 a_2 \cdots a_N$. The semigroup element Z_i is the one-step transition function δ_{a_i}.

The parameter variation problem is to study how the function $\delta_{a_1 \cdots a_N}$ is changed if a_k is changed to a'_k. The scheme could be useful in error correction attempts for regular languages. For instance, an error correcting interpreter could quickly find the 'most likely' single-character errors. Attempts to insert semicolons as terminators, to interpolate multiple characters, or to transpose adjacent pairs of characters require only a small modification of our procedures. See [15] for a more powerful but more complex approach to this problem.

(2) Consider optimization problems expressible as *serial discrete* decision processes, solvable by either forward or backward dynamic programming. A somewhat simplified version may be expressed as follows: Given N known functions, $f_1, \ldots, f_N$, each a function of two variables with finite ranges, compute the function

$$h(x_1, x_{N+1}) \equiv \min_{\substack{\{\text{values of} \\ x_2, x_3, \ldots, x_N\}}} [f_1(x_1, x_2) + f_2(x_2, x_3) + \cdots + f_N(x_N, x_{N+1})].$$

The parameter variation problem is to explore the effect of varying each f_i. Each Z_i represents a function f_i. $Z_i Z_{i+1}$ is the function g given by

$$g(x_i, x_{i+2}) = \min_{\{x_{i+1}\}} [f_i(x_i, x_{i+1}) + f_{i+1}(x_{i+1}, x_{i+2})].$$

The next three problems may be transformed into instances of the above optimization problem. The state variables of the usual dynamic programming recurrences map to the variables x_i above. The examples are included because they are more familiar than the general optimization problem.

(3) *Assignment of tasks to machines*: Suppose a task can be decomposed into N sequential stages, and to do stage i at location j costs $p_i(j)$. Moving the output of stage i at location u to location v costs $c_i(u, v)$. We wish to choose $\{x_i \triangleq \text{location of task } i \mid i = 1, \ldots, N\}$ to minimize the total processing costs plus communication costs. That is, minimize

$$\mathscr{X} = \sum_{i=1}^{N-1} [p_i(x_i) + c_i(x_i, x_{i+1})] + p_N(x_N).$$

The parameter variation problem is to explore the effect on min $\mathscr{X}$ of altering each $[(p_i(\cdot) + c_i(\cdot, \cdot)]$.

(4) Consider the classical *resource allocation* problem of allocating limited resources among competing projects so as to maximize profit. The problem may be formalized as: Given functions $r_j(\cdot)$, $j = 1, \ldots, N$, for each value of TOTAL, $0 \leq \text{TOTAL} \leq \text{LIMIT}$ (for some known LIMIT), choose the single stage allocations $Y_1, Y_2, \ldots, Y_N$ to maximize $\sum_1^N r_j(Y_j)$ subject to $\sum Y_j = \text{TOTAL}$. We change variables with $x_j \equiv \sum_1^{j-1} Y_i$ (so $Y_j = x_{j+1} - x_j$).

The functions r_j are the inputs that are to be varied, and the maximum value of $\sum_1^N r_j(Y_j)$ is the output. $f_j(x_j, x_{j+1}) \equiv r_j(x_{j+1} - x_j)$. x_1 is 0 by definition. $H(0, x_{N+1})$ tells the best return achievable with x_{N+1} resource units. The parameter variation problem is to explore the effect of altering each r_j.

(5) *Equipment replacement*: A machine's operating cost changes with time. At the start of each time period, one may exchange a machine of any age for a machine of any other age. (A cash payment may be involved in the trade.) Functions f_i which tell the trading plus operating cost during period i for each possible trading action are known. The problem is to choose a replacement strategy to minimize costs over some given number of time periods.

(6) The *group knapsack* problem [12], (from relaxation methods for integer programs) is: minimize cx ($x \geq 0$, integer vector) subject to $Ax \equiv b$ (mod 1). Several parameter variation problems may be defined. One may study the effect (on min cx) of changing columns A_j to alternate values A_j', and also change b to b', while still imposing the congruence constraint. The semigroup element Z_j is the jth column vector A_j (in the ring).

(7) *Multiplication of N matrices*: Given equal sized square matrices $M_1, \ldots, M_N$, find the product $M_1 M_2 \cdots M_N$. The parameter variation problem is to explore the effect of changing each M_i.

(8) *Convolution of N functions*: The probability density function of the sum of N random variables is formed by the convolution of the individual densities. In one application (noted by D. Shier) the individual random variables represent sources of measurement error, and the probability density function for total error is desired. The parameter variation problem is to find the effect of altering the individual densities. The semigroup element Z_i is the density function of the ith random variable.

3. A simple, yet optimal algorithm for Type I sensitivity analysis

Devising sensitivity analysis algorithms is difficult only in the specific settings. (The equipment replacement problem was our starting place.) Algorithm design is much easier when one abstracts to a semigroup setting.

The most direct solution to the sensitivity analysis problem is to compute each variant separately. $N-1$ multiplications will be needed for each k, for a total of $O(N^2)$. With a little care, the running time can be reduced to $O(N)$.

Define left partial products L_i for $1 \leqslant i \leqslant N$, $L_i \triangleq \prod_{j \leqslant i} Z_j$. Also, define right partial products $R_i \triangleq \prod_{j \geqslant i} Z_j$. We adopt the convention that multiplying any X by the empty terms L_0 or R_{N+1} results in X but costs nothing.

Algorithm 1.

Step 1. Multiply the terms Z_i right to left, obtaining

$$\{R_i \mid i = N-1, \ldots, 1\}.$$

Step 2. Multiply the terms Z_i left to right, obtaining

$$\{L_i \mid i = 2, \ldots, N-1\}.$$

Step 3. For $k = 1$ until N: compute Variant$_k$:= $L_{k-1} Z'_k R_{k+1}$.

Complexity. Exactly $(N-1)+(N-2)+(2N-2) = 4N-5$ multiplications are performed. $(N-2)$ intermediate results must be stored. (We may combine Steps 2 and 3; R_k will be discarded when L_k is computed.)

It seems desirable to relate the multiplication count to running time in each application. For 'number of algebraic multiplications' to be a useful complexity measure, the time for multiplications must be roughly constant. For certain applications which were deliberately omitted from our list (e.g. the 0–1 knapsack problem, polynomial multiplication), an elementary term Z_k (which in knapsack problems has only two entries) requires much less time to multiply than a partial product. In this kind of situation, it may be best to compute the variants directly from their definitions. $O(N^2)$ multiplications will be required, but each multiplication will include a simple term Z_i as one operand.

4. Optimality and near-uniqueness

We will obtain a lower bound on the algorithm's worst case performance by considering its behavior in a situation without useful input-dependent relations, e.g. the free semigroup with unequal inputs. Algorithms operating in this setting are essentially straight line programs, performing a sequence of semigroup multiplications. To provide a handier description, we model an arbitrary straight line program on given size input by a circuit. Inputs to the circuit are the values $Z_1, \ldots, Z_N$, $Z'_1, \ldots, Z'_N$. Each gate forms the product of its two inputs. The multiplier outputs may be routed to any number of other multipliers but no directed cycles are permitted. The circuit's outputs are the desired variants. A circuit is optimal if it contains the minimum possible number of multipliers.

Theorem 1 (below) shows that Algorithm 1 is optimal. Theorem 2 shows that all optimal algorithms are nearly the same as Algorithm 1.

Consider any optimal circuit, and for any $k > 0$ let P_k denote a path in the circuit from Z'_k to Variant$_k$.

Lemma 1. *Any optimal circuit may be transformed into an optimal circuit in which*
(i) L_{k-1} *and* R_{k+1} *are available as multiplier outputs,*
(ii) *the output values of multipliers not on* P_k *in the original circuit remain unchanged.*

Proof. First, a transformation will be defined. We begin by deleting P_k from the circuit. Next, insert new multipliers in such a way that all the orphaned multipliers off P_k which were inputs to P_k from left [right] are multiplied together left to right [right to left]. Now, multiply the grand left product by Z'_k and the result by the grand right product. It is shown below that the transformed circuit still solves the sensitivity analysis problem (i.e. computes all the variants), has no more multipliers than the original circuit, still uses a minimum number of multiplications, and satisfies conditions (i) and (ii).

We begin with some observations. Since the original circuit was optimal, it could not compute any superfluous outputs. For instance, outputs which included the same term twice or two 'primed' terms would not be present in an optimal circuit. Thus the subgraph consisting of all edges on directed paths leading to Variant$_k$ must be a tree, with Z'_k the only primed input. Each multiplier has 2 inputs, so all these trees are binary. All binary trees on N leaves have the same number of multipliers (i.e. internal nodes).

It is readily seen that multipliers on P_k are used only to compute Variant$_k$ and that no multipliers off P_k have been altered (although the destination of some outputs may have changed). Thus, variants other than Variant$_k$ are unaffected by the transformation. Now, the grand product formed on the left includes all terms to the left, and hence is L_{k-1}. Similarly, the grand right product is R_{k+1}. Multiplying these with Z'_k as specified produces Variant$_k$. Hence, the circuit still computes all the variants. This argument has also verified that conditions i and ii of the lemma still hold.

The tree below Variant$_k$ in the new circuit still has N leaves, so the number of multipliers there is unchaged. The rest of the circuit is unaltered. Thus the new circuit has the same number of multipliers as the old, and is optimal

Lemma 2. *Paths* $P_{k'}$ *and* $P_{k''}$ *are vertex disjoint, for* $k' \neq k''$. *In an optimal circuit, applying the transformation to* $P_{k'}$ *will affect multiplier outputs only on* $P_{k'}$. *(Hence the transformation may be applied for different values of k, without interference.)*

Proof. In an optimal circuit, quantities which are not useful for any required output cannot be computed. All multiplier outputs on $P_{k'}$ [$P_{k''}$] include $Z'_{k'}$ [$Z'_{k''}$]. No useful output includes both $Z'_{k'}$ and $Z'_{k''}$. Hence $P_{k'}$ and $P_{k''}$ are disjoint.

The transformation changes the values only for multipliers on P_k, or off P_k and receiving input from a vertex on P_k. If the second case arose, the product would include Z'_k but would not be used for computing Variant$_k$, and hence could not be used in computing any required output. Thus, the second case does not arise.

Theorem 1. *Every algorithm for the sensitivity analysis problem uses at least* $4N-5$ *multiplications in the worst case* (*i.e. Algorithm 1 is optimal*).

Proof. By Lemma 2, we may start with an arbitrary optimal circuit and apply the transformation for every k, $k=1,\ldots,N$. One obtains an optimal circuit which has available all the variants, plus L_{k-1}, $L_{k-1}Z'_k$, and R_{k+1} for all k (assuming suitable interpretations for the cases of $k=1$ and N). But these are all the multiplier outputs of Algorithm 1; hence, by Lemma 1 this optimal circuit has at least as many multiplications as Algorithm 1, so Algorithm 1 is optimal.

A product $Z_i\cdots Z_j$ is called *internal* if $i\neq 1$ and $j\neq N$.

Lemma 3. *No optimal circuit may compute any internal product.*

Proof. If a sequence of transformations from Lemma 1 is applied to an arbitrary optimal circuit, no internal products are altered. However, the resulting circuit, that of Algorithm 1, has no internal products. Therefore, no optimal circuit has internal products.

Lemma 4. *If an optimal circuit does not contain* L_i, *it does not contain* L_j, $i<j<N$.

Proof. An internal product is required to compute L_j, $i<j<N$ without L_i. But by Lemma 3 an optimal circuit may not contain an internal product.

The following theorem implies that any optimal circuit must be essentially the same as the circuit for Algorithm 1. The circuits will, in fact, differ only in the association of the last two multiplications along the path to each Variant$_k$ ($k>0$), and in the value of i used in the multiplication which computes Variant$_0=L_iR_{i+1}$.

Theorem 2. *Every optimal circuit computes* $\{L_i\mid i=2,\ldots,N-2\}$, $\{R_i\mid i=3,\ldots,N-1\}$ *and for each* i $(i=2,\ldots,N-1)$ *one of* $L_{i-1}Z'_i$ *or* Z'_iR_{i+1}. *No optimal circuit computes* $Z_iZ'_j$, Z'_iZ_j, *or* Z_iZ_j *unless* $j=i+1$ *and either* $i=1$ *or* $j=N$.

Proof. Consider an optimal circuit which does not contain L_i, $i\leq N-2$. By Lemma 4, the circuit does not contain L_{i+1} either. Now apply transformations independently to

P_{i+1} and P_{i+2}, by Lemma 2. A new multiplier with output L_i is created as a result of the transform on P_{i+1}. As a result of the transform on P_{i+2}, a new multiplier with output L_{i+1} is created. Since the transforms are independent, L_{i+1} does not use input from any vertex created by the transform on P_i. But, by Lemma 3, L_{i+1} must receive L_i as input. Hence, the transformations have created an optimal circuit with two multipliers having output L_i, which we observed in Lemma 1 was a contradiction. Therefore, it must be that the optimal circuit did contain L_i, $i \leq N-2$. It is easy to show that as long as L_i is available, it will be suboptimal to compute $Z_iZ'_{i+1}$ since at the same cost one can compute $L_iZ'_{i+1}$.

The argument about $\{R_i\}$ and $\{Z'_iZ_{i+1}\}$ is analogous.

5. Type II sensitivity analysis

In this section, a number of results for other sensitivity analysis problems will be presented. All the problems (including the one above) can be considered as the execution of a series of commands of the form CHANGE Z_i to Z'_i, RESTORE Z_i to its previous value, and compute PRODUCT. (The command handler may wish to store some intermediate products.) In the sensitivity analysis problem presented above, each CHANGE command was immediately followed by PRODUCT and RESTORE, each Z_k was CHANGEd exactly once, and the entire sequence was known in advance. There are several other interesting kinds of problems having different restrictions and different advance information.

5.1. Type II sensitivity analysis in commutative semigroups

In a hill-climbing optimization procedure, one might not RESTORE Z'_i to its original value before altering Z_{i+1}. If RESTOREs are omitted, another interesting (open) problem could be to optimally handle arbitrary sequences of CHANGEs. We consider in this section only the case, where CHANGEs are made successively to Z_1, $Z_2, \ldots$ (In commutative semigroups, if the sequence of CHANGEs is known in advance and no term is CHANGEd twice, one may reorder terms to achieve this situation.)

As before, let Variant_0 denote $\prod_{i=1}^{N} Z_i$, but let $\text{Variant}_k \triangleq (\prod_{j\leq k} Z_{j'})(\prod_{j>k} Z_j)$. The sensitivity analysis problem of this section is again to compute $\{\text{Variant}_k \mid k = 0, \ldots, N\}$.

A simple algorithm is optimal for this problem also. Let $R_k \triangleq \prod_{i\geq k} Z_i$ (as before), and let $L'_k \triangleq \prod_{i\leq k} Z'_i$.

Algorithm 2.

Step 1. Compute $\{R_k \mid k = N-1, N-2, \ldots, 1\}$ by multiplying right to left.

Step 2. Compute $\{L'_k \mid k = 2, \ldots, N\}$ by multiplying left to right.

Step 3. For $k = 1$ until N compute $\text{Variant}_k := L'_kR_{k+1}$ (by convention multiplying by the nonexistent R_{N+1} has no effect or cost).

This algorithm requires $3N$-3 multiplications, and storage for N-2 intermediate results.

Theorem 3 (below) is stronger than Theorem 2 in two ways. The circuit is compared with circuits for the apparently easier problem set in systems where multiplication commutes. And there is no hedging on uniqueness. In proving the necessary lemmas below, assume inductively that Theorem 3 holds through $N-1$. (The statement is trivial for 1 and 2.) It will be useful to define the deletion of input Z_N from an algorithm (i.e. circuit). To delete Z_N, eliminate all multiplications of the form YZ_N (i.e. where Z_N is an input). When (YZ_N) is subsequently used, use Y instead.

The deletion of Z'_N is defined similarly. Note that only multipliers which receive Z_N or Z'_N as inputs are removed.

Let SENS(N) denote the sensitivity analysis problem for products of N quantities, and consider *any* optimal circuit (called CIRC(N)) for SENS(N). Let CIRC($N-1$) denote the circuit obtained by deleting Z_N and Z'_N from CIRC(N). (CIRC($N-1$) is appropriately named; Theorem 3 and Lemma 5 will imply that it is the optimal circuit.)

Lemma 5. (i) CIRC($N-1$) *solves* SENS($N-1$).

(ii) *At least three multipliers were eliminated by the deletions, two involving Z_N as an input, and one involving Z'_N.*

Proof. (i) In SENS($N-1$), the required outputs differ from those of SENS(N) only in that Z_N or Z'_N should not be part of the product. The deletion of Z_N and Z'_N accomplish this.

(ii) Z'_N was clearly involved in at least one multiplication, the one which formed Variant$_N$. This multiplication is deleted. Z_N multiplies some primed [respectively unprimed] terms in the course of computing Variant$_{N-1} = (\prod_{i=1}^{N-1} Z'_i)Z_N$ [Variant$_0 = \prod_{i=1}^{N} Z_N$]. Hence at least two multiplications are deleted when Z_N is deleted.

Lemma 6. *An optimal circuit* CIRC(N) *for* SENS(N) *has exactly* $3N$-3 *multipliers.*

Proof. CIRC(N) cannot have more multipliers than the circuit for Algorithm 2, which has $3N-3$. Now, by inductive hypothesis, Algorithm 2, which uses $3(N-1)-3$ multiplications, is optimal for SENS($N-1$). Hence, CIRC($N-1$) (obtained by deleting Z_N and Z'_N from CIRC(N)) has at least $3(N-1)-3$ multipliers. Since at least three multipliers were removed by the deletion, CIRC(N) has at least $[3(N-1)-3]+3 = 3N-3$ multipliers.

Lemma 7. *In every optimal* CIRC(N), *Z'_N multiplies just L'_{N-1}, and Z_N multiplies just L'_{N-1} and Z_{N-1}.*

Proof. In an optimal circuit, there can be only one multiplier which computes a given output. Let M denote the unique multiplier in CIRC($N-1$) whose output is L'_{N-1}.

Suppose $\text{Variant}_N = L'_{N-1}Z'_N$ is not computed in CIRC(N) by multiplying the output of M by Z'_N. Consider then the multiplier in CIRC(N) whose output is $L'_{N-1}Z'_N$. After Z'_N and Z_N are deleted to form CIRC($N-1$), this multiplier has output L'_{N-1} in CIRC($N-1$), contradicting the uniqueness of M in CIRC($N-1$). Hence, Variant_N must be computed in CIRC(N) by multiplying L'_{N-1} by Z'_N.

An identical argument will show that Variant_{N-1} must be computed by multiplying L'_{N-1} by Z_N.

Two of the new multipliers have been fixed. There remains only the multiplication of Z_N by unprimed terms.

In CIRC($N-1$), the paths which compute Variant_0 and Variant_{N-2} share Z_{N-1} but no other inputs or multipliers, since one has primed terms, and the other unprimed. (Recall that $N>2$.) But for CIRC(N) to solve SENS(N), both of these computations must include Z_N. This can be done using one additional multiplier only if Z_{N-1} multiplies Z_N and the output is used in the later computations.

Theorem 3. *For the type II sensitivity analysis problem, in commutative semigroups, Algorithm 2 is optimal and its circuit is the unique optimal circuit.*

Proof. The theorem will now be proved by induction on N. The assertion is easily verified for SENS(1) and SENS(2). Assume $N>2$ and that the theorem holds up to N−1. Lemma 5 has shown that Algorithm 2 solves SENS($N-1$). The proof of Lemma 6 has completed the inductive proof of the assertion that Algorithm 2 is optimal. Since every optimal CIRC(N) has $3N-3$ multipliers, and every CIRC($N-1$) obtained from an optimal CIRC(N) by deletion has at most $[3N-3]-3 = 3(N-1)-3$ multipliers, every CIRC($N-1$) so obtained is an optimal circuit for SENS($N-1$). CIRC($N-1$) is unique, by inductive hypothesis.

By Lemma 7, the three deleted multipliers must all be in specific places in CIRC(N), i.e. there is only one optimal circuit for SENS(N). By Lemma 6, Algorithm 2, which uses $3N-3$ multiplications, is optimal.

5.2. On line computation

In previous examples, we knew in advance what variables were to be CHANGEd, i.e. what variants were to be computed. Suppose now that after the initial product is computed, a user at a terminal gives commands 'CHANGE Z_i and compute PRODUCT' without any prespecified ordering. Products must be computed immediately, i.e. on line. We will assume that $N-2$ intermediate results may be stored (all the intermediates from one computation). How may the number of multiplications be kept small? An elegant solution is obtained by reformulating the problem using binary trees.

Left-to-right multiplication is just one way of assocating (i.e. parenthesizing) the elements Z_i to form the product. Any association may be represented by a binary tree, with the values Z_i at the leaves and each internal node representing the product of its two sons. The original product $\prod Z_i$ is computed according to some binary tree, and the intermediate results are stored. Now, suppose Z_k is CHANGEd to Z'_k. It will be necessary to recompute values in the tree along the path from Z_k to the root; the number of new multiplications equals the length of that path.

5.2.1. Optimal worst case

To minimize the worst-case time, the multiplication pattern should use a uniform binary tree. With such a tree, at worst $\lceil \log_2 N \rceil$ CHANGEs will be needed. By contrast, the tree corresponding to left-to-right multiplication has worst-case path length $N-1$.

$\lceil \log_2 N \rceil$ is optimal, as long as only one set of intermediate results (i.e. only one tree) is allowed to be stored. We conjecture that keeping more than one tree would be counterproductive in practice, as the time to update all trees after unrestored CHANGEs would probably exceed the time saved.

5.2.2. Optimal expected number of multiplications

Let the weight of the ith leaf node be the frequency with which Z_i is CHANGEd. To minimize expected number of multiplications, we need the ordered binary tree with minimum weighted external path length. T.C. Hu has a complicated $O(N \log N)$ algorithm for this problem [7].

In many applications, multiplication commutes, so it is undesirable to enforce an ordering on the leaves of the tree. To minimize the expected number of multiplications with reordering permitted, one uses a Huffman tree (which is found by a very simple $O(N \log N)$ algorithm [7]).

5.3. Open problems

5.3.1. Off-line computation

Suppose the sequence of all future commands is known in advance, but is not of the simple forms considered in Sections 3 or 5.1. This problem is open. A greedy strategy may be appropriate.

5.3.2. Unknown stationary random process

No knowledge of the relative frequencies is available, but the process generating CHANGEs is known to be stationary, so that relative frequencies can be estimated. This is also open. The results in [1] about adaptive trees may be relevant.

5.3.3. Multiple changes

All problems considered so far have command lists with at most one CHANGE between PRODUCT commands. Bentley has adapted techniques from selection problems to deal with multiple CHANGEs [2].

5.3.4. Commutativity

If multiplication commutes, then the lower bound argument of Theorem 1 is no longer valid, since it assumes Z'_k must be multiplied separately on the left and right.

We know neither a way to resurrect the lower bound nor a way to speed the algorithms by exploiting commutativity.

If multiplication commutes and has an inverse (i.e. the semigroup is an Abelian group), then it is sufficient to multiply $\prod^N Z_i$ by $Z'_k Z_k^{-1}$ to CHANGE Z_k to Z'_k. To solve our original problem, $(N-1)+2N=3N-1$ multiplications plus N inverse computations are needed. This simple procedure will probably be superior if inverse calculations are easy.

5.4. Circuit depth and parallelism

If the computation is to be done in parallel, one might wish a circuit of minimum depth. Techniques developed in [8] appear applicable.

5.5. Time-space tradeoffs

The naive $O(N^2)$ algorithm requires storage for only one intermediate product; a slightly modified version of the naive scheme uses $\frac{1}{2}(N-1)(N+4)$ multiplications and two intermediate results. Our time-optimal method used $4N-5$ multiplications and $N-2$ locations. An interesting research question would be to find the time-optimal algorithm as a function of allotted storage. [2] and [14] report schemes requiring $O(N \log^2 N)$ time and $O(\log N)$ storage.

6. Summary

The study of associative multiplication problems has been motivated. For several kinds of sensitivity analysis in associative multiplication problems we have presented algorithms and demonstrated the algorithms' optimality. A number of open questions have been posed.

References

[1] B. Allen and I. Munro, Self organizing binary search trees, *J. ACM* **25** (1978) 526–535.
[2] J. Bentley, Personal communication, Carnegie-Mellon University.
[3] G.A. Cheston, Incremental algorithms in graph theory, Technical Report 91, Department of Computer Science, University of Toronto (1976).
[4] F.Y. Chin and D. Houck, Algorithms for updating minimum spanning trees, *J. Comput. System Sci.* **16** (3) (1978) 333–334.
[5] A.M. Geoffrion and R. Nauss, Parametric and postoptimality analysis in integer linear programming, *Management Sci.* **23** (5) (1977) 453–466.
[6] S. Goto and A. Sangiovanni-Vincentelli, A new shortest path updating algorithm, *Networks* **8** (1) (1978) 341–372.
[7] D. Knuth, *The Art of Computer Programming* (Addison-Wesley, Reading, MA, 1973).
[8] R. Ladner and M. Fischer, Parallel prefix computation, International Conference on Parallel Processing (1977).
[9] K. Murty, *Linear and Combinatorial Programming* (Wiley, New York, 1976).

[10] A. Rosenthal, Sensitivity analysis in nonserial optimization problems, to appear.
[11] A. Rosenthal, Decomposition algorithms and sensitivity analysis for probabilistic circuits and fault trees, to appear.
[12] H. Salkin, *Integer Programming* (Addison–Wesley, Reading, MA, 1975).
[13] P. Spira and A. Pan, On finding and updating spanning trees and shortest paths, *SIAM J. Comput.* 4 (3) (1975) 375–380.
[14] N. Swami, Personal communication, University of Illinois.
[15] R. Wagner, Order-*n* correction for regular languages, *Comm. ACM* **17** (5) (1974) 265–268.