J. Symbolic Computation (1998) **25**, 285–293

The Geobucket Data Structure for Polynomials[†]

THOMAS YAN[‡]

Department of Computer Science, Cornell University, Ithaca, New York 14853, U.S.A.

The *geobucket* data structure is a suitable intermediate representation of polynomials for performing large numbers of polynomial additions in the face of interspersed lead-term extractions. A sum involving N terms has worst-case running time $O(N \log N)$, matching or surpassing the performance of lists and binomial heaps. This makes the geobucket a good choice for performing reductions in Gröbner basis computations.

1. Introduction

A common representation of polynomials is a sorted list of terms; this structure supports linear-time binary addition of polynomials and constant-time extraction of lead terms. Although adequate for small calculations, the list representation usually behaves poorly for large aggregate sums, such as those arising during Gröbner basis computations (Buchberger, 1965, 1985), whose inner loop reduces a polynomial p as follows:

```
while p ≠ 0 and not done do
    Extract the lead term of p
    Perhaps add another polynomial summand to p
```

where the polynomial summands added to p depend on the extracted lead terms but are not known initially. A frequent feature of reductions is for partial sums (accumulated in p) to be(come) much larger than summands; to avoid poor performance in such situations, an appropriate representation must be chosen for partial sums.

In November 1992, using *geobuckets* (defined in Section 3) instead of lists for partial sums allowed us to compute a Gröbner basis for commuting 4-by-4 matrices ("the 4-by-4 example", henceforth; see Appendix) at least 32 times faster—under 43 hours instead of over 8 weeks.[§] This dramatic speedup is due to better treatment of large size ratios between summands and the corresponding partial sums: one reduction had 1.7×10^5

[†] NDSEG fellowship DAAL03-90-G-0158, ONR grant N00014-92-J-1973, and NSF grant CCR-9503319.

[‡] E-mail: tyan@cs.cornell.edu

[§] With some guidance from Mike Stillman, we did the computation attributed to him in Remark 2 of Hreinsdóttir (1994); we were perhaps the first to compute such a basis.

0747–7171/98/030285 + 09 $25.00/0 sy970176

summands, with partial sums of at least 1.2×10^5 terms, but the average size of summands was only 46 terms. By adding only polynomials of comparably bounded size, the *geobucket* data structure achieves $O(N \log N)$ worst-case asymptotic running time for a sum containing N terms interspersed with up to N lead-term extractions (each term being extracted at most once). Although binomial heaps have the same worst-case running time (Vuillemin, 1978; Brown, 1978), our experiments indicate geobuckets are a better choice.

The paper is organized as follows: terminology and cost model (Section 2), definition of geobuckets (Section 3), worst-case time and space analysis (Section 4), empirical results and conclusions (Section 5), and benchmark specifications (Appendix).

2. Preliminaries

Fix a polynomial ring. A *monomial* is a product of indeterminates. A *term* $a \cdot \alpha$ is a product of non-zero coefficient a and monomial α. *Like* terms have the same monomial. A *polynomial* f is a finite sum of non-like terms; its *size* $\#f$ is the number of terms. Polynomial *addition* is done by combining like terms, as expected; we refer to combination of like terms as *cancelation* since it eliminates at least one term from the sum.

Fix a (total) *monomial order*; this induces a partial order on terms: like terms are incomparable. A non-zero polynomial f whose largest term is $a \cdot \alpha$ has *lead term* $\mathrm{LT}(f) = a \cdot \alpha$, *lead coefficient* $\mathrm{LC}(f) = a$, and *lead monomial* $\mathrm{LM}(f) = \alpha$. *Lead-term extraction* removes the lead term, leaving the remaining terms in polynomial $\mathrm{tail}(f) = f - \mathrm{LT}(f)$.

Note: for polynomials, "large" and "small" refer solely to size and never to lead terms.

We are interested in the time and space complexity of adding n polynomials $f_1, \ldots, f_n$ interspersed with up to N lead-term extractions performed on the partial sums, where $N = \#f_1 + \cdots + \#f_n$ is the total number of terms.† With no extractions, the ith *partial sum* $p_i = f_1 + \cdots + f_i$ is the intermediate sum up to and including f_i; with extractions, p_i includes the effects of all extractions performed before adding f_i. For simplicity, assume all f_i are non-zero, so $n \leq N$ and hence n is $O(N)$.

We measure space in the number of terms and time in the number of monomial operations, primarily monomial comparisons. We ignore the number of coefficient additions because it is dominated by the number of monomial operations and is the same, up to a factor of two, for all algorithms considered.‡

A standard polynomial representation is a list of terms, sorted in decreasing order: space usage is optimal since cancelation eliminates as many like terms as possible, lead-term extraction takes constant-time, and binary addition $f_1 + f_2$ essentially merges the two lists, taking at most $\#f_1 + \#f_2$ comparisons. So a sum $f_1 + \cdots + f_n$ with no cancelation where all summands are of size 1 is therefore essentially insertion sort, which has $O(N^2)$ worst-case behavior. Geobuckets are partly motivated by the superior $O(N \log N)$ performance of merge sort, which merges lists of comparable size. To see how this pays off, consider the worst-case cost of computing a sum $f_1 + f_2 + f_3$ with

† Spatial locality of memory accesses is important for large calculations. Although our analysis and experimental data do not address this, geobuckets should have good spatial locality because they are built upon lists, which can be implemented as arrays, and because the lists are sequentially accessed and are fairly large for the most part.

‡ This justification suffices for coefficients of fixed size. Otherwise, although the number of *coefficient* additions is bounded, we do not know of bounds on the number of *primitive* (machine) operations.

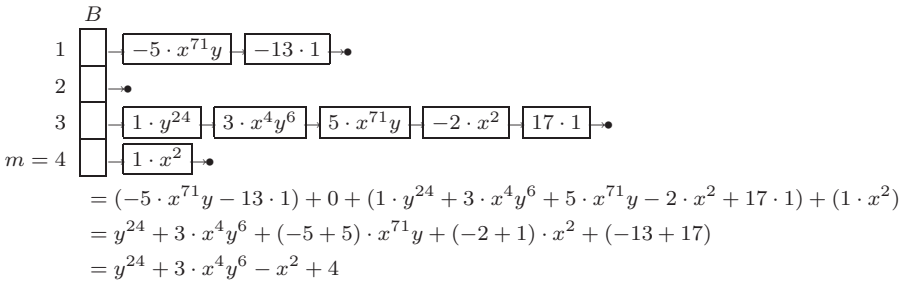

Figure 1. Sample geobucket ($d = 2$, $m = 4$). Coefficients are integers, and monomials $x^i y^j$ are ordered by $x^i y^j > x^{i'} y^{j'}$ if $j > j'$ or both $j = j'$ and $i > i'$.

$\#f_1 \gg \#f_2 = \#f_3 = 1$. Computing left-to-right takes $2\#f_1 + 1$ comparisons, but right-to-left takes only $\#f_1 + 2$ comparisons: after f_2 and f_3 are cheaply combined, the sorted terms in their sum share the cost of merging with f_1 in a single pass. Thus, the key to geobuckets is (re)grouping summands.

Note: unless we specify otherwise, a polynomial f is represented as a list.

3. Geobucket Data Structure and Operations

We define the geobucket and give implementations for creation, addition, canonicalization, and lead-term extraction, which simultaneously tests for zero. See Figure 2 for pseudo-code.

Fix a constant d, $1 < d \in \Re$; a good choice is determined in Section 4. For the rest of the paper, logarithms $\log(\cdot)$ are base d.

Definition 3.1. *A* geobucket *is an array $B[1..m]$ of* buckets *$B[i]$ of* capacity *d^i: bucket $B[i]$ is a polynomial of size $\#(B[i]) \leq d^i$. The* canonical polynomial *represented by geobucket B is the sum $B[1] + \cdots + B[m]$ of its buckets.*

See Figure 1 for an example and note the following: there is some "internal cancelation" of terms; bucket $B[1]$ is full with $d^1 = 2^1 = 2$ terms; bucket $B[2]$ is empty; bucket $B[3]$ is only partly full, but its contents of five terms exceeds the capacity $d^2 = 2^2 = 4$ of $B[2]$; and bucket $B[4]$ is way under capacity.

To create the resizable array $B[1..m]$ of buckets, allocate enough room from the start and maintain the number m of buckets in use. Below we show that $m \leq \lceil \log N \rceil$. Since N is bounded by the amount of memory available and at most $\lceil \log N \rceil$ buckets are needed (see below), one can effectively compute a small constant upper bound on the space required. For example, given $2^{32-2} = 2^{30}$ addressable words, if $d = 4$, then at most $1 + \log 2^{30} = 16$ buckets are needed.†

To add a polynomial f to geobucket B, add it into the smallest bucket allowed. The

† This approach treats N as bounded, which violates the spirit of asymptotic analysis and can waste excessive space if numerous small geobuckets are in use. These are unlikely to be practical concerns: $\log N$ is small, and in Gröbner basis computations, reductions are done sequentially, so it suffices to preallocate a single geobucket. However, this nitpick is easily addressed using the simple "array doubling" approach Hopgood (1968) suggested for hash table overflow.

Creation of $B = 0$

```
Allocate array B
Initialize B's entries to 0
m := 0
```

Addition of $B[1..m]$ with polynomial f

```
(We assume f may be modified.)
i := max(1, ⌈log(#f)⌉)
if i ≤ m then
    f := f + B[i]
    while i ≤ m and #f > d^i do
        f := f + B[i + 1]
        B[i] := 0
        i := i + 1
m := max(m, i)
B[i] := f
```

Canonicalization of $B[1..m]$ into polynomial f

```
f := 0; for i := 1 to m do f := f + B[i]
```

Extraction of Lead Term from $B[1..m]$

```
repeat
    j := 0
    for i := 1 to m do if B[i] ≠ 0 then
        if j = 0 or LM[i] > LM[j] then j := i
        else if LM[i] = LM[j] then
            LC[j] := LC[j] + LC[i]; B[i] := tail[i]
until j = 0 or LC[j] ≠ 0
if j = 0 then B = 0, so there is no lead term
else lt := LT[j]; B[j] := tail[j]; return lt
```

where LT[i] = LT($B[i]$).
LM[i], LC[i], and tail[i] are similarly defined.

Figure 2. Geobucket code.

size of f and the contents of the bucket are both bounded by the bucket's capacity, so in this way only polynomials of comparably bounded size are added. When the contents of a bucket exceeds its capacity, it *overflows*: its contents are added into the next bucket, which may also overflow, etc.; a sequence of overflows is called a *cascade*. Note that when the code shown in Figure 2 adds a polynomial f into a bucket $B[i]$—both during the initial addition into a bucket and during cascades—the following condition holds:

$$\#f > d^{i-1} \text{ and } \#(B[i]) \le d^i. \tag{3.1}$$

New buckets are created only when f exceeds the capacity of the largest bucket. Since a bucket capable of holding all $N \le d^{\lceil \log N \rceil}$ terms cannot have its capacity exceeded, the largest bucket ever used has index at most $\lceil \log N \rceil$, i.e.

$$m \le \lceil \log N \rceil < 1 + \log N. \tag{3.2}$$

To obtain the canonical polynomial represented by B, return the sum $B[1] + \cdots + B[m]$ of the buckets, computed from small to large buckets.

To extract the lead term, linearly search through the lead terms of non-empty buckets, canceling like terms as they are discovered, so at least one term is always removed from a non-empty geobucket. Note that the code continues to search for like or larger terms LT[i] if cancelation yields a zero coefficient LC[j] = 0. The choice of whether to stop or continue this loop on i is somewhat arbitrary, and both choices are encompassed in our analysis in Section 4.

The name *geobucket* comes from the use of buckets with capacities in a geometric series, an arrangement reminiscent of other data structures and algorithms, including binomial heaps and other tree structures (Vuillemin, 1978; Brown, 1978), memory allocation (Knuth, 1973), and doubling a hash table when it overflows (Hopgood, 1968).

4. Worst-Case Analysis

THEOREM 4.1. (SPACE COMPLEXITY) *In the worst case, $O(N)$ space is used, even if only $O(1)$ space is required for the list representation.*

PROOF. Let $f_1, \ldots, f_{n-1}$ be polynomials of sizes $d, d^2, \ldots, d^{n-1}$ with distinct monomials.

Set $f_n = -(f_1 + \cdots + f_{n-1})$. Summing these places each f_i into a distinct bucket, so no cascades and hence no cancelation is performed, although the canonical polynomial is 0.

THEOREM 4.2. (COST OF ADDITION) *The total number of monomial comparisons incurred by additions is at most* $(d+1)(1+\log N)N$.

PROOF. The cost $(\#f + \#(B[i]))/\#f$ per term in f of adding a polynomial f into a bucket $B[i]$ is at most $1+d$ comparisons because $\#(B[i]) < d \cdot \#f$ by (3.1). Cascades move terms only from smaller to larger buckets, so each term can be added into each bucket at most once, and by (3.2) there at most $1+\log N$ buckets.

THEOREM 4.3. (COST OF CANONICALIZATION) *The cost of canonicalization is at most N monomial comparisons beyond the cost of addition.*

PROOF. The contents of bucket $B[m-i]$ are added at most $i+1$ times, so each term is compared at most $i+1$ times, but the analysis of addition already provides for moving a term in bucket $B[m-i]$ by i places up to bucket $B[m]$ and performing $d+1 > 2$ comparisons at each step along the way. When $i \geq 1$, this more than covers the $i+1 \leq 2i < (d+1)i$ comparisons required for canonicalization. This leaves us with just the costs of inspecting the terms in $B[m-i]$ when $i = 0$; this is obviously at most N.

THEOREM 4.4. (COST OF LEAD-TERM EXTRACTION) *Lead-term extractions perform at most $N \log N$ monomial comparisons.*

PROOF. Since there are at most $1+\log N$ buckets, each linear search performs at most $\log N$ comparisons. Each linear search of a non-empty geobucket removes at least one term, so at most N non-trivial linear searches can be performed.

COROLLARY 4.5. (TIME COMPLEXITY) *The number of monomial comparisons incurred from performing addition, interspersed with up to N lead-term extractions and terminated by at most one canonicalization, is at most*

$$(d+1)(1+\log N)N + N\log N + N = (d+2)N\log N + (d+2)N,$$

so the worst-case asymptotic time complexity is $O(N \log N)$.

PROOF. During lead-term extraction and polynomial addition, each coefficient sum is preceded by a monomial comparison. Creating and resizing the resizable array are constant time operations. The linear cost of polynomial addition dominates the cost of array access—both the cost of sequential access and the cost of computing $\lceil \log(x) \rceil$. Thus, monomial comparisons asymptotically dominate all other costs.

4.1. CHOICE OF d

When N is large, the worst-case cost as measured by monomial comparisons is dominated by the term $(d+2)N\log N$, which is basically the cost of cascades. Applying the identity $\log x = \ln x / \ln d$ and pulling out the $N \ln N$ factor leaves us with $(d+2)/\ln d$ to minimize. The minimum occurs at $d \simeq 4.3$. If we attempt to minimize just the cost of cascades, then we get $d \simeq 3.6$. Empirically searching for d as a power of 2, we found that

Table 1. Profiles of benchmarks.

GB	No. red's	Ave. $\#f$	ΣN	Σn	Max sum N	Max sum n	Penetration (%)
$u5$	114	12	19 435	1 595	460	34	77–97
$u6$	389	26	432 756	16 408	3 454	109	73–89
$u7$	2220	107	58 494 253	544 550	146 213	986	75–87
$f5$	981	20	579 311	29 554	3 456	226	60–63
$f6$	1982	25	3 900 731	156 736	25 691	1 173	71–73
$f7$	3199	29	17 601 842	598 145	99 399	3 838	75–75
$f8$	4057	35	48 500 006	1 403 807	314 996	10 207	75–75

$d = 4$ worked best for the 4-by-4 example; this is a gratifying correspondence between an actual run and our worst-case analysis.

5. Empirical Results

We computed various Gröbner bases using different polynomial representations and collected time and space data: total number of monomial comparisons, total time spent on polynomial manipulation, cumulative space usage, and amount of "penetration". We now give more detailed descriptions and explain how data were collected.

5.1. explanation of data

The benchmarks are cyclic roots of unity of degrees 5 through 7 ($u5$, $u6$, $u7$) and partial runs of the 4-by-4 example ($f5$, $f6$, $f7$, $f8$). See Appendix for definitions of the ideals. Bases for these homogeneous ideals are computed monotonically, from lower to higher degrees; run fd is the 4-by-4 example stopped after degree d is completed. For each benchmark, we computed the total number $\sum N$ of terms, the number $\sum n$ of summands, the average size $\sum N / \sum n$ of summands, total number of reductions, the maximum number N of terms in any reduction and the corresponding number n of summands, and lower and upper bounds on penetration (see below). See Table 1.

We took measurements for lists (L), geobuckets (G), and binomial heaps of polynomials (H'_P) with "aggressive cancelation"; see Yan (1996) for descriptions and data for this and other heap variants. We include binomial heaps because they are an obvious alternative: their $O(1)$ *lazy meld* (heap union) and $O(\log N)$ *deletemin* (maximal element extraction) give them an $O(N \log N)$ worst-case running time, too (Vuillemin, 1978; Brown, 1978).

Monomial comparisons are counted using a global counter.

Runs were on a Sun 4/670 under Solaris 2.5 with 256 Mb of real memory, so paging was not a major concern. Repeated runs indicate that times in Table 3 are accurate to about two significant digits. To estimate the time P spent on polynomial operations, we consider the total time to be the sum $O + P$ of P with the time O spent on everything else. An underestimate of O gives a conservative overestimate P that is better than just using $O + P$. Suppose we have total times $O + L$, $O + G$, $O + H'_P$, and $O' + L + G$ for lists, geobuckets, heaps, and a dual run duplicating the same computation on both lists and geobuckets, respectively. We expect $O' > O$ due to poorer cache behavior and higher runtime costs, e.g. memory management, so the difference $(O+L)+(O+G)-(O'+L+G)$

Table 2. Monomial comparisons (millions). The parenthesized numbers are the corresponding ratios with respect to geobuckets.

Rep	$u5$	$u6$	$u7$	$f5$	$f6$	$f7$	$f8$
G	0.035	0.78	115	1.82	16.1	89	277
L	0.037 (1.0)	0.90 (1.2)	164 (1.4)	4.43 (2.4)	79.5 (4.9)	800 (9.0)	4330 (15.6)
H'_P	0.056 (1.6)	1.32 (1.7)	188 (1.6)	3.84 (2.1)	33.2 (2.1)	180 (2.0)	552 (2.0)

Table 3. Estimated time (seconds) for polynomial operations. The parenthesized numbers are the corresponding ratios with respect to geobuckets. At the bottom is a line O of estimated overheads; see the text for an explanation about $f8$.

Rep	$u5$	$u6$	$u7$	$f5$	$f6$	$f7$	$f8$
G	0.18	5.2	1016	12.2	132	907	3 368
L	0.15 (0.8)	3.6 (0.7)	1100 (1.1)	19.7 (1.6)	376 (2.8)	4481 (4.9)	29 523 (8.8)
H'_P	0.44 (2.4)	11.7 (2.2)	2546 (2.5)	30.4 (2.5)	286 (2.2)	1713 (1.9)	6 774 (2.0)
O	0.54	9.8	1084	19.4	97	235	235∗

gives us an underestimate of O. For $f8$, since this gives $O < 0$ and $f7$ is a subcomputation, we use $f7$'s estimate, but using $O = 0$ would not change the numbers much.

Fix a representation and let $\#p_i$ be the number of terms used to represent the ith partial sum p_i. We measure cumulative space usage as the cumulative number $S = \sum_{i=1}^{n} \#p_i$ of terms surviving each addition. The ratio of the cumulative space used by two different representations shows how much more/less space is used on average.

When using lists for partial sums, we can measure how deeply the terms of summands "penetrate" into the partial sums as the ratio C/W of the number C of actual comparisons and the number W of comparisons theoretically performed in the worst case. In the worst case, addition inspects all terms. The resulting comparisons are those performed on terms surviving each addition—i.e. cumulative space S—and those performed on terms canceling to 0—at most $N/2$ comparisons because each cancelation/comparison deletes two terms. Since S and $S + N/2$ bound W, we can bound the amount C/W of penetration by $C/(S + N/2)$ and C/S, which we use instead of measuring W because the lower bound already exhibits substantial penetration.

5.2. CONCLUSIONS

Table 1 shows "penetration" of at least 60% in all benchmarks when using lists, so we expect geobuckets to perform at least comparably fast to lists, if not much better. This is indeed the case, as can be seen in both Table 2 and also Table 3. These tables mostly agree on relative performance among the data structures, and the ratios for time and the ratios for number of comparisons are somewhat related. This validates our decision to count monomial operations as a measure of time. Note that heaps have disappointing performance. While they do perform better than lists on partial runs of the 4-by-4 example, they perform rather worse than lists on cyclic roots of unity.

The increased speed of geobuckets over lists is at the cost of about 70% extra space in our benchmarks (Table 4). Heaps, however, again perform much worse, using almost

Table 4. Cumulative space usage (millions of terms). The bracketed numbers are the corresponding ratios with respect to lists—not geobuckets.

Rep	$u5$	$u6$	$u7$	$f5$	$f6$	$f7$	$f8$
L	0.038	1.0	189	7.1	109	1059	5 781
G	0.053 [1.4]	1.5 [1.4]	374 [2.0]	10.6 [1.5]	170 [1.6]	1756 [1.7]	9 751 [1.7]
H'_P	0.184 [4.9]	10.7 [10.6]	8 974 [47.4]	17.6 [2.5]	308 [2.8]	3622 [3.4]	22 888 [4.0]

50 times extra space for $u7$. This suggests that geobuckets tend to use considerably less space than heaps, especially in the cases that lists work well. Yan (1996) describes geobucket variations that might further decrease space usage.

The data from our experiments show that while heaps might have some speed advantages over lists, their profligate use of space is a concern. Geobuckets, however, offer significant speedups over both lists and heaps, at a reasonably modest cost in space, so they are the best choice of the three.

Acknowledgements

Many thanks to David Gries, Ben Keller, Greg Morrisett, Mike Stillman, Tim Teitelbaum, Todd Wilson, Rich Zippel, and two anonymous referees for many useful comments.

Appendix. Polynomial Ideals used for Benchmarks

The field of coefficients for all benchmarks is the set of integers modulo 17.

The generators for the 4-by-4 example are the 16 entries of $AB - BA$, where

$$A = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \qquad \text{and} \qquad B = \begin{bmatrix} q & r & s & t \\ u & v & w & x \\ y & z & A & B \\ C & D & E & F \end{bmatrix}.$$

The monomial order used is lexicographic on $F > \cdots > A > z > \cdots > a$. This is better than graded reverse lexicographic, but not as good as the orders used in Hreinsdóttir (1994). (We tried using the first order given in Hreinsdóttir (1994) and found that using geobuckets gave speedups of roughly two or three times over lists.)

The generators for the (homogenized) cyclic roots of unity of degree d are $p_d = x[0] \cdots x[d-1] - x[d]^d$ and the d polynomials $p_k = \sum_{i=0}^{d-1} \prod_{j=0}^{k} x[(i+j) \bmod d]$, for k, $0 \le k < d$. The monomial order used is graded reverse lexicographic on $x[0] > \cdots > x[d]$.

References

Brown, M. R. (1978). Implementation and analysis of binomial queue algorithms. *SIAM Journal of Computing* **7**, 298–319.

Buchberger, B. (1965). *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal.* PhD thesis, Universitat Innsbruck, Institut fur Mathematik, German.

Buchberger, B. (1985). Gröbner bases: An algorithmic method in polynomial ideal theory. In: Bose, N. K. (ed.), *Multidimensional Systems Theory*, chap. 6, pp. 184–232. Dordrecht: Reidel.

Hopgood, F. (1968). A solution to the table overflow problem for hash tables. *Computer Bulletin* **11**(4), 297–300.

Hreinsdóttir, F. (1994). A case where choosing a product order makes the calculations of a Groebner basis much faster. *J. Symbolic Comput.* **18**, 373–378.

Knuth, D. E. (1973). *The Art of Computer Programming, volume I: Fundamental Algorithms*, chap. 2 (2nd edn). Reading, MA: Addison-Wesley.

Vuillemin, J. (1978). A data structure for manipulating priority queues. *Communications of the ACM* **21**(4), 309–315.

Yan, T. (1996). The geobucket data structure for polynomials. Tech. Rep. CUCS TR96-1607, Department of Computer Science, Cornell University. Available electronically at `http://cs-tr.cs.cornell.edu/Dienst/UI/2.0/Describe/ncstrl.cornell%2fTR96-1607`.

Originally received 18 September 1996
Accepted 20 May 1997