MATHEMATICS OF COMPUTATION, VOLUME 27, NUMBER 123, JULY, 1973

Implementation and Analysis of the Todd-Coxeter Algorithm

By John J. Cannon, Lucien A. Dimino, George Havas and Jane M. Watson

Abstract. A recent form of the Todd-Coxeter algorithm, known as the lookahead algorithm, is described. The time and space requirements for this algorithm are shown experimentally to be usually either equivalent or superior to the Felsch and Haselgrove-Leech-Trotter algorithms. Some findings from an experimental study of the behaviour of Todd-Coxeter programs in a variety of situations are given.

1. Introduction. The Todd-Coxeter algorithm [20] (TC algorithm) is a systematic procedure for enumerating the cosets of a subgroup H of finite index in a group G, given a set of defining relations for G and words generating H. At the present time, Todd-Coxeter programs represent the most common application of computers to group theory. They are used for constructing sets of defining relations for particular groups, for determining the order of a group from its defining relations, for studying the structure of particular groups and for many other things.

As an example of the use of the algorithm, consider the following family of defining relations, Men(n), due to Mennicke:

$$\begin{aligned}\text{Men}(n) = \text{gp}\langle a, b, c, d, e \mid\ & a^4 = b^2 = c^2 = d^2 = e^2 = abcd \\ & = ede^{-1}a^{-1}da = ea^{-1}bcbae^{-1}b^{-1}a^{-1}c^{-1}ab^{-1} \\ & = ecace^{-1}a^{-1}b^{-1}a^{-1}b^{-1}a = (bc)^n = 1\rangle.\end{aligned}$$

Using Todd-Coxeter programs to find the index, in Men(n), of the dihedral subgroup $\langle b, c\rangle$ of order $2n$, we have found that |Men(1)| = 16, |Men(2)| = 256, |Men(3)| = 2,688, |Men(4)| = 36,864 and |Men(5)| = 551,040. The determination of |Men(5)| was done using secondary storage and occupied an IBM 360/50 for 79.5 hours.

It is over 10 years since the first published description of a Todd-Coxeter program appeared (Felsch [8]). Advances in the understanding of the underlying processes and the use of more sophisticated implementation techniques now enable us to produce much faster programs. It thus appears appropriate to give an account of a recent implementation of the TC algorithm.

In order to gain a better understanding of the algorithm, we have undertaken an extensive investigation of its behaviour. In this paper, we present a summary of our findings concerning

Received March 28, 1972.
AMS (MOS) subject classifications (1970). Primary 20F05; Secondary 20-04.
Key words and phrases. Todd-Coxeter algorithm, generators and relations, group theory.

(a) the comparative behaviour of the three most commonly programmed forms of the algorithm, and

(b) the behaviour of the algorithm with respect to different classes of defining relations and different choices of subgroup.

In Section 2, we discuss various forms of the TC algorithm, while Section 3 contains a formal description of the lookahead TC algorithm. Section 4 contains the results of our investigation of the TC algorithm.

A general account of earlier TC programs, together with some applications, can be found in Leech [11].

2. Algorithm Design. Suppose

$$G = \mathrm{gp}\langle g_1, \cdots, g_r \mid R_1(g_1, \cdots, g_r) = \cdots = R_s(g_1, \cdots, g_r) = 1\rangle$$

and let $H = \langle h_1, \cdots, h_t\rangle$ be a subgroup of G, where the h_i's are words in the g_j's. If $[G : H] = n$, the right cosets of H in G will be denoted by the integers 1 to n. The coset consisting of H itself will be always represented by the integer 1. We shall write $(i)g_j$ to denote the coset obtained upon multiplying coset i on the right by generator g_j.

Definition 1. If m is the number of cosets of H currently defined and r is the number of generators in the given presentation for G, then the *partial coset table* T is an $m \times 2r$ array whose (i, j)th element is defined as follows. Suppose i is a coset of H and $s_j = g_l$ or g_l^{-1} for some generator g_l. Then, $T(i, j) = k$, if $(i)s_j$ is known to be coset k (i.e., if $(i)s_j = k$), and $T(i, j) = 0$, if $(i)s_j$ is unknown. Thus, the columns of T correspond to the generators of G and their inverses, while the rows of T correspond to the currently defined cosets.

In practice, if a generator is an involution, then T contains a single column for both the generator and its inverse. If $1, \cdots, m$ are precisely the complete set of cosets of H and no entry of T is zero, then T is called the *coset table* for H in G. Since it will be clear from context when we are discussing a partial coset table rather than a coset table, we shall use the term coset table in both cases.

Let $w = s_1 \cdots s_k$, where $s_i = g_l$ or g_l^{-1} for some generator g_l, be a word and let i be a coset. Assuming that the coset $(i)s_1 \cdots s_j$ is defined, the coset $(i)s_1 \cdots s_j s_{j+1}$ is said to be *defined* if and only if $T((i)s_1 \cdots s_j, s_{j+1})$ is nonzero, in which case

$$(i)s_1 \cdots s_j s_{j+1} = T((i)s_1 \cdots s_j, s_{j+1}).$$

If w is the empty word, then $(i)w$ is defined to be i. Sometimes, we shall indicate that $(i)w$ is not defined, by writing $(i)w = 0$.

We shall usually denote a word in the generators of G by a subscripted w and a single generator or its inverse by a subscripted s.

Definition 2. Let R be a relator in G and let i be some coset of H. We now proceed to define the process of *applying relator R to coset i* with respect to some partial coset table T by defining a transformation of T corresponding to each of the following four possibilities:

(a) $R = w$ and $(i)w = i$. Here, T is left unchanged.

(b) $R = w_1 s w_2$, where w_1 is a subword of R, s is not the identity and w_2 is a subword of R right adjacent to s, such that $(i)w_1 = j$, $(i)w_2^{-1} = k$ but $T(j, s) = T(k, s^{-1}) = 0$. Note that w_1 or w_2 may be empty. We then set $T(j, s) = k$ and $T(k, s^{-1}) = j$. This is called a *deduction*.

(c) $R = w_3w_4$, where w_3 is a subword of R and w_4 is a subword of R right adjacent to w_3, such that $(i)w_3 = j$ and $(i)w_4^{-1} = k$ with $j \neq k$. Note that one of w_3 or w_4 may be empty and that w_3 is chosen so that its length is maximal. This means that j and k represent the same coset and we say that j and k are *coincident*, or alternatively that one of the cosets j and k is *redundant*. We write $j \sim k$. In this case, one of the cosets j and k must be deleted from T and this is carried out by a process called the coincidence procedure which will be specified later.

(d) $R = w_5w_6w_7$, where w_5 and w_7 are nonempty subwords of R such that $(i)w_5 = (i)w_7^{-1} = 0$. In this case, T is left unchanged.

In cases (a), (b) and (c), we say that the *R-cycle at coset i is complete*, while in case (d), we say that the *R-cycle at coset i is incomplete.*

In terms of the concepts just introduced, we may summarize the action of the TC algorithm as follows. First, cosets are defined so that each generator h_i of H forms a cycle at coset 1. Then, each given relator R_i of G is applied to each coset. If some R_i-cycles are incomplete, then, at some stage, new cosets must be introduced. The process terminates when every R_i-cycle is complete at every coset currently defined.

For further background information, including a description of the basic algorithm for processing coincidences, the reader is referred to the paper of Leech [11]. Proofs that various versions of the algorithm terminate in the case of finite index have been given by Trotter [21], Mendelsohn [17] and Dimino [7].

The central problem in programming the TC algorithm is finding a satisfactory rule for introducing new cosets. As the range of application of a Todd-Coxeter program is thus far limited by the amount of storage required to hold the partial coset tables generated during an enumeration, one tries to define cosets in such a way that as few redundant cosets as possible are introduced.

It is easily seen that the application of the TC algorithm to certain presentations will necessarily require the introduction of redundant cosets. For example, the group

$$G = \text{gp}\langle a \mid a^p = a^q = 1, \, p \text{ and } q \text{ prime}, \, p \neq q \rangle$$

is trivial but $\min(p, q)$ cosets must be defined before the relation $a = 1$ are discovered. Indeed, given any integer m, one can produce a presentation for the trivial group which requires the definition of at least m cosets before the TC algorithm is able to deduce that the group is trivial. Then, by adding such a presentation of the trivial group to some presentation of a group G, we can produce an arbitrarily bad presentation for G.

Thus, it is fruitless to expect to be able to produce a version of the TC algorithm which performs well even for all presentations of a single group. However, the majority of presentations which arise in practice turn out to be reasonable from the point of view of the TC algorithm and so we concentrate our efforts on producing forms of the algorithm which operate as efficiently as possible on this class of presentations.

The two most popular strategies for introducing new cosets are the Felsch method [8], which defines fewer redundant cosets at the expense of execution time, and the Haselgrove-Leech-Trotter method [11], [21] which usually defines more redundant cosets but which typically runs faster than the Felsch method. The Haselgrove-Leech-Trotter method was developed by Haselgrove in 1953 and later adapted by

Leech, Trotter and others. For brevity, we shall refer to it simply as the HLT method or HLT algorithm.

Suppose that the definition $(i)s_j = k$ has just been made. The Felsch procedure is to apply all significantly different cyclic permutations of relators beginning with s_j, to coset i. This process is repeated with any deductions, $(i')s_j' = k'$, which may have been discovered, until all possible consequences of the original definition have been discovered. Only at this stage, will a new coset be introduced, if necessary, and then by defining it so that the first vacant position in the coset table is filled. In the HLT method, relators are applied to the cosets in the order in which the cosets were introduced. If for some coset i and relator R, the R-cycle at coset i is incomplete, sufficient new cosets are immediately introduced so as to complete the R-cycle at i.

Another form of the TC algorithm has been discussed by Mendelsohn [17]. However, it is much too inefficient to be considered for machine implementation and so it will not be discussed further.

In order to be able to compare various forms of the TC algorithm, we introduce some measures which are associated with the enumeration of the cosets of a subgroup H in a group G by a specific form of the TC algorithm. Suppose

I is the index of H in G;

M is the maximum number of cosets defined at any instant during the enumeration;

T is the total number of cosets defined in the enumeration;

t is the execution time;

$\tau = M/I$;

$\epsilon = T/I$.

We shall indicate the form of the TC algorithm to which one of the above measures refers by a subscript: F for the Felsch procedure and H for the Haselgrove-Leech-Trotter procedure.

The number τ can be interpreted as measuring the amount of space required to perform the enumeration, over and above that required to store the coset table for H in G. As noted above, coset enumerations with respect to certain presentations will necessarily involve redundancies, so that it is not possible to design a coset enumeration algorithm having $\tau = 1$ for all possible coset enumerations. So, current efforts are directed towards producing fast programs for which τ is close to 1 for large classes of presentations. It is obvious that usually $\tau_H \geqq \tau_F$ and it is observed in practice that for a large proportion of presentations τ_H is significantly greater than τ_F.

Of the machine versions of the TC algorithm which have thus far been proposed, the Felsch method usually defines the fewest cosets as it tries to extract the maximum amount of information each time a new coset is defined. However, a form of the algorithm has been developed which performs the majority of enumerations very quickly but which rarely requires significantly more space than the Felsch algorithm in order to complete an enumeration. As this form of the Todd-Coxeter, which we shall call the *lookahead method*, gives the best all round performance, we shall describe it in some detail in this and the next section.

A type of lookahead was used by Leech in 1959 (Leech [11, p. 265, last paragraph]) but this form of the TC algorithm did not really begin to develop until Guy [9] wrote his lookahead TC program for the ATLAS at Cambridge in 1967.

The lookahead method operates in two distinct phases: a *defining phase* and a *lookahead phase*. As long as the number of cosets defined at any instant is less than a specified number M_L, the algorithm remains in the defining phase. In this phase, the enumeration proceeds by the HLT method. When, however, the number of cosets defined exceeds the limit M_L, the algorithm switches to the lookahead phase. Here, relators are applied to cosets as before, but if a relator cycle is incomplete at some coset, no new cosets are defined to complete the cycle. The aim is to discover a large number of deductions and coincidences without introducing any new cosets. If the enumeration is still incomplete at the end of the lookahead phase and if sufficient storage space is available, we return to the definition phase in which we remain until either the enumeration completes or the number of cosets defined again passes some preset limit. Thus, the algorithm alternates between the defining phase and the lookahead phase.

Let us denote a lookahead form of the TC algorithm by the letter L. If the lookahead phase is not used, then $\tau_L = \tau_H$. When lookahead is used, τ_L varies with the value of M_L supplied. Let us denote the values of τ_L and ϵ_L, when M_L is the minimum possible for the enumeration to complete, by $\hat{\tau}_L$ and $\hat{\epsilon}_L$, respectively. Experimentally, it is found that $\hat{\tau}_L \simeq \tau_F$. While ϵ_L is often much greater than ϵ_F, this is not felt to be an important consideration because of the faster processing of cosets in a lookahead program as compared to a Felsch program.

To summarize, the lookahead form of the TC algorithm can usually perform coset enumerations faster than other machine algorithms used at present.

On the other hand, if storage space is at a premium, a lookahead program will not require significantly more storage in order to complete an enumeration than a Felsch program. Indeed, the lookahead algorithm often requires less space to complete an enumeration than the Felsch algorithm!

There are a number of possible ways of arranging the lookahead. Guy [9], for example, divides his available space up into a number of blocks and applies lookahead before allowing the coset table to extend into a new block. We shall refer to this technique as *bumping*. If the optimum block size is chosen, this technique will result in extremely rapid enumerations. On the other hand, a poor choice of block size can result in inefficient enumerations. The other possibility, of course, is to let the coset table exhaust the available space before applying lookahead. Also, when the program is in the lookahead phase, it can return to the defining phase as soon as a single coincidence has been discovered (*incremental lookahead*) or only after all relators have been applied to all cosets currently defined (*complete lookahead*).

If all cosets have been processed in the lookahead phase of an incremental lookahead program, the lookahead begins again with the first coset not yet processed in the defining phase. A single application of lookahead, of either kind, to every coset not yet processed in the defining phase is called a *lookahead pass*. Generally, both incremental and complete lookahead programs are arranged so that they terminate when either the enumeration completes or a lookahead pass fails to discover a single coincident coset. In the case of complete lookahead, a considerable amount of time can sometimes be saved in situations where the enumeration does not complete by specifying that execution is to terminate if less than a certain number of coincidences are discovered during an application of lookahead.

We have programmed both the complete and incremental forms of the look-

ahead algorithm. Comparing their behaviour on a series of test examples, we have discovered that the following hold:

(i) The two techniques have equivalent power in the sense that the minimum amount of store necessary to complete a given enumeration is the same for both.

(ii) The execution time for the two techniques generally differs by less than 15 per cent. In some cases, the difference favours incremental lookahead and, in other cases, it favours complete lookahead. The reason for this is that if, at the time lookahead is called, sufficient cosets have already been defined for the enumeration to complete, then incremental lookahead often defines further unnecessary cosets before reaching a "critical" coincidence, while complete lookahead would discover this coincidence before defining any further cosets. On the other hand, if insufficient cosets are defined at the time lookahead is called, then it is slightly more economical to define new cosets as soon as space is freed by the discovery of a coincidence.

Since we have so far been unable to distinguish between complete and incremental lookahead on the basis of performance, we have chosen to describe in detail a complete lookahead TC algorithm in the next section because the algorithm is slightly simpler. The lookahead statistics given in Section 4 refer to this algorithm. An implementation of an incremental lookahead program is described in Dimino [7].

We conclude this section by mentioning other possible strategies which deserve further investigation. The lookahead algorithm can be modified slightly to allow the lookahead phase to use relators (e.g. redundant relators) which are not used in the defining phase. More fundamentally, in the HLT algorithm, instead of applying all the relators to a particular coset at the same time, one could allow the application of long relators to cosets to lag some distance behind the application of the short relators. (See Section 4.2.)

3. A Lookahead Todd-Coxeter Algorithm. In this section, we describe in some detail an implementation of a lookahead version of the Todd-Coxeter algorithm. The ANSI FORTRAN text of the program is available from the authors.

3.1. *Data Structures.* Considerable gains can be achieved in the run time efficiency of a Todd-Coxeter program by careful design of the data structures. The four main components are

(a) generators for H and relators for G,

(b) coset table,

(c) active row list,

(d) coincidence queue.

(a) *Generators for H and Relators for G.* These two sets of words are stored in a one-dimensional array SGREL in the following way. The generators of G and their inverses are mapped sequentially into the integers beginning at 3, except, if a generator is involutory, then both it and its inverse are assigned the same integer. The involutory relator is not stored as it is not needed. The representation of generator g_i of G will be denoted $\mathbf{g}_i$ and that of g_i^{-1} will be $\mathbf{g}_i^{-1}$. Then, each word is stored as a string in the $\mathbf{g}_i$, $\mathbf{g}_i^{-1}$. Prefixed to each string is the length of the string.

Example. The presentation $s^3 = t^2 = (s^{-1}tst)^2 = 1$ is stored as the one-dimensional array

3, 3, 3, 3, 8, 4, 5, 3, 5, 4, 5, 3, 5

where $s \rightarrow 3$, $s^{-1} \rightarrow 4$, $t \rightarrow 5$, $t^{-1} \rightarrow 5$.

In addition to this array, a table of inverses INVCOL is stored. The entries of INVCOL are defined by the rule INVCOL $(s_i) = s_i^{-1}$ where s_i runs through the generators of G and their inverses.

(b) *Coset Table.* As stated in Definition 1, the partial coset table T contains a column for each generator and the inverse of each noninvolutory generator. In practice, the columns of T are indexed by the $\mathbf{g}_i$ since the first two columns of the two-dimensional array SPACE in which T is stored are used for the active coset list. Having a single column for involutory generators, clearly results in considerable space saving and also some time saving since the relation $g_i^2 = 1$ need not be processed, as it comes implicitly from the column sharing by g_i and g_i^{-1}. The space saving is most important since the range of application of a TC program depends critically upon the amount of core storage available rather than upon execution time. On word machines it is often necessary to pack as many coset table entries as possible into each machine word. [The coset table entries are actually pointers to other cosets (rows) of the table. The name assigned to a coset is the number of the corresponding row of T. (Thus, on a CDC 6000 series machine, for example, three coset table entries may be packed into each machine word.)]

(c) *Active Row List.* When coincidences are discovered during an enumeration, certain rows of the coset table become inactive and, hence, available for reuse. Early programs collected this free space from time to time by moving all the active rows together at one end of the store while simultaneously renumbering the active cosets and coset table entries. This, however, is a time consuming procedure and so more recent programs link together the active rows of the coset table in a list structure. One cannot avoid linking the active rows by merely marking inactive rows and adding them to a list of free rows, available for reuse, for it is necessary to know which active rows have been processed in the defining phase, and which have been processed in the lookahead phase, etc.

The linking together of the active rows of T is effected by means of the *active row list* which is stored in a two-dimensional array A (occupying the first two columns of the array SPACE), where the ith row of A corresponds to the ith row of T. If the ith row of T is active, $A(i, 1)$ points to the previous active row while $A(i, 2)$ points to the next active row. The backward chaining is necessary since one must link around a row of T when it becomes inactive. (The contents of $A(i, 1)$ and $A(i, 2)$ are actually array indices, but we shall use the term pointer for brevity.)

Initially, the rows of A are unlinked, and the rows of T are available sequentially for the next coset to be defined. When an active row of T becomes inactive, the corresponding row of A is removed from the active row list and, instead, linked to the front of FREEL, the free row list. Once all of the sequentially available rows of T are used, rows may be recovered from FREEL and reused.

(d) *Coincidence Processing.* The discovery of a coincidence usually leads to the discovery of further coincidences which must be stored to await processing. The allocation of space to store these coincidences presents a considerable problem since, for example, in cases of total collapse, the number of coincidences which must be stored may approach the number of cosets currently defined. However, it turns out that there is always sufficient space in the array A to store all possible coincidences.

Suppose $i < j$ are cosets found to be coincident. As row j of T is to be removed

from the active row list, the two corresponding fields of A are spare. So a flag to indicate that coset j has been found redundant is placed in $A(j, 2)$, together with a pointer to row i, while a pointer to the next member of the queue Q of coincidences awaiting processing is placed in $A(j, 1)$.

Coincidences are added to Q in the following way. Suppose we deduce $i \sim j$. We examine $A(j, 2)$ to see if row j is already marked as coincident. If j is marked as coincident with k (say), we reset j to k and go back and examine the new $A(j, 2)$. We do the same with i, and finally reach $i' \sim j'$ where neither i' nor j' is already coincident. If $i' = j'$, then the original coincidence $i \sim j$ yields no new information so nothing need be added to Q. Otherwise, choosing $i'' = \min(i', j')$, $j'' = \max(i', j')$, we mark j'' as coincident with i'' and add j'' to Q. Finally, when the coincidence is fully processed $A(j, 2)$ is linked to the free space list.

In this way, coincidences may be queued and processed so that no extra space is required.

3.2. *Coset Enumeration Algorithm.* We now give a formal description of the algorithm.

The algorithm first applies the nontrivial generators of H (if any) to coset 1, and then begins applying the defining relators to the cosets from 1 on until either the enumeration is complete or the available space is exhausted. If the latter occurs, lookahead is performed and, provided some coincidences are discovered, control is then returned to the defining phase. If no coincidences are found in lookahead, the enumeration terminates with the coset table incomplete.

A coset i is said to be *closed* if all the relator cycles at coset i are complete. The action of the defining phase is to systematically close the defined cosets from 1 onwards in the order defined by the active row list. If lookahead discovers a closed coset i, the coset's position in the active row list is altered so that coset i becomes the last coset closed in the defining phase. This sometimes saves quite a lot of unnecessary work in the defining phase while the extra overhead is negligible.

The coset enumeration program consists of two central subroutines: ENUM(ERATE) and COINC(IDENCE), together with a driving program TODCOX.

TODCOX. This routine initializes the various data structures and performs the function of control routine.

(1) [Initialize] Input the generators for H and the defining relators for G; Set up pointers and tables; Allocate storage for the adjustable array SPACE.

(2) [Perform enumeration] Call ENUM.

(3) [Next problem] Go to (1) for next problem if desired, otherwise exit.

ENUM(NMAX, NCOL, SPACE). This routine systematically enumerates the cosets of H in G. The routine operates in three modes:

(a) Apply subgroup generators to coset 1, mode SG.

(b) Apply relators in defining phase, mode DP.

(c) Apply relators in lookahead phase, mode LA.

When the enumeration is complete, or alternatively when the storage available is exhausted, the index of H in G or an overflow message is printed with some execution statistics. Control then returns to TODCOX.

IFRONT is the last coset reached in the forward scan.

IBACK is the last coset reached in the backward scan.

K points to beginning of current word in SGREL.

L points to end of current word in SGREL.

M points to position in current word in forward scan.

N points to position in current word in backward scan.

CLOSED is a flag used to indicate if a coset is closed in the lookahead phase.

(1) [Initialize] Initialize pointers and counters; If there are no nontrivial subgroup generators go to (8); Else set mode type and parameters to SG.

(2) [Initialize scan of next word] K ← pointer to first symbol of word; L ← pointer to last symbol of word; IFRONT, IBACK ← coset currently being processed; M ← K.

(3) [Forward scan] I ← T(IFRONT, SGREL(M)); If I = 0, N ← L and go to (4); (forward scan ends); Else IFRONT ← I; M ← M + 1; If M > L go to (7); (forward scan reaches end of word); Else go to (3).

(4) [Backward scan] I ← T(IBACK, INVCOL(SGREL(N))); If I = 0, go to (5); (backward scan can go no further); Else IBACK ← I; N ← N − 1; If N < M go to (7); (backward scan meets forward scan); Else go to (4).

(5) [Relator incomplete] If N = M + 1, go to (6); (deduction); Else if mode is LA go to (10); (do not define new cosets in lookahead phase); If no cosets available go to (9); (space exhausted so commence lookahead); Else I ← next coset available; Initialize new row and update statistics; T(IBACK, INVCOL(SGREL(N))) ← I; T(I, SGREL(N)) ← IBACK; (define new coset); Go to (4); (return to backward scan).

(6) [Only one gap so complete cycle with deduction]

$$T(IBACK, INVCOL(SGREL(N))) \leftarrow IFRONT;$$

(deduction); If T(IFRONT, SGREL(N)) ≠ 0, IFRONT ← T(IFRONT, SGREL(N)) and go to (7); Else T(IFRONT, SGREL(N)) ← IBACK; (another deduction); IFRONT ← IBACK.

(7) [Relator complete] If IFRONT ≠ IBACK call COINC; (process nontrivial coincidence); If more words left in set for this phase go to (2); If mode is LA go to (11).

(8) [Start or continue defining phase] If mode is SG reset mode and parameters to DP; If no more cosets to be processed go to (13); (enumeration complete); Get next coset to be processed; Reset pointers to first word of set; Go to (2).

(9) [No space left in DP] Output overflow message; Reset mode to LA.

(10) [Incomplete scan in lookahead] CLOSED ← . FALSE .; (note coset is not closed); If more words left in set go to (2).

(11) [End of set of words in LA] If CLOSED = . TRUE ., link current coset to become last closed coset to avoid reprocessing in DP; If no more cosets to be processed go to (12); Else reset pointers to first word of set; CLOSED ← . TRUE .; Go to (2); (continue lookahead).

(12) [End of LA] Output message to signify end of LA; If no space available return; (enumeration terminates); Else reset current coset to last closed coset and mode to DP; Reset pointers to first word of set; Go to (2); (return to DP).

(13) [Enumeration complete] Output index of H in G and statistics; Return.

COINC (NMAX, NCOL, SPACE, KX, KY). Given a pair of coincident cosets by ENUM, this routine discovers all consequences and modifies the coset table

accordingly. The modifications which have to be made to the coset table are described in Leech [11].

NCOL is the number of columns of T.

KX < KY are the original pair of coincident cosets.

KA < KB are a pair of coincident cosets.

JA, JB run through the entries of rows KA, KB respectively.

I indexes the columns of T.

(1) [Initialize] KA ← KX; KB ← KY; Link around row KB; (remove row from active row list).

(2) [Prepare to process new coincidence] Link KB onto the free row list; Decrease active coset count; I ← 0; (set column counter).

(3) [Compare Ith entries of rows KA and KB] I ← I + 1; If I > NCOL, go to (7); JB ← T(KB, I);

(a) If JB = 0, go to (3); (no new information).

(b) If JB = KB, JB ← KA; Go to (4).

(c) If JB ≠ 0 and JB ≠ KB, T(JB, INVCOL(I)) ← 0; Go to (4). (T(JB, INVCOL(I)) initially contains KB, which we delete at this stage rather than replace by KA to avoid having two occurrences of KA in the same column.)

(4) [Continue comparison] JA ← T(KA, I);

(a) If JA = 0, T(KA, I) ← JB and go to (6); (deduction).

(b) If JA = KB, T(KA, I) ← KA; JA ← KA; Go to (5); (possible new coincidence).

(c) If JA ≠ 0 and JA ≠ KB, go to (5); (possible new coincidence).

(5) [Queue coincidence] Add the pair of coincident cosets JA, JB to the coincidence queue if they are not already there; Link around the row of T corresponding to the higher numbered of the two cosets.

(6) [Assign inverse T entry if currently undefined] If T(T(KA, I), INVCOL(I)) = 0, T(T(KA, I), INVCOL(I)) ← KA; Go to (3).

(7) [Fetch next coincidence from queue] If no more coincidences in queue, return; Else extract new KA, KB from queue; Go to (2).

4. The Behaviour of the Todd-Coxeter Algorithm. The Todd-Coxeter algorithm often displays a great variation in behaviour when applied in apparently similar situations. At present, there is little understanding of the reasons for this and so we have undertaken an experimental study of the behaviour of the algorithm. In this section, we shall summarize some of the more interesting findings from these studies, first, to help new users to get the most out of their coset programs and, second, in the hope that these examples may form the starting point for theoretical studies of the algorithm.

Apart from execution time, the most important parameter associated with an in-core coset enumeration is the amount of space required to store any intermediate partial coset table over and above that required to store the final coset table, i.e., the parameter τ. The reason for focussing attention on τ rather than ϵ is that many attempted enumerations fail, even though sufficient space is available to comfortably store the final coset table, because τ is significantly greater than one. Thus, it is of considerable interest to examine those enumerations having large τ's.

For this reason, we call an enumeration *F-pathological* (or simply *pathological*)

if τ_F is significantly greater than one. As a guide line we shall call an enumeration pathological if $\tau_F \geqq 1{\cdot}1$. The reason for using the Felsch algorithm here is that it most closely approximates the hand method for enumerating cosets and also it has been observed that if $\tau_F > \tau_A$ for one of the major machine TC algorithms, A, currently in use, then τ_F is usually close to τ_A. In a study by one of us (Watson) of nearly 300 enumerations, it was found that $M_F = I$ (i.e., $\tau_F = 1$) in more than 80 per cent of the cases. (These enumerations were chosen so that no relator had a proper subword equal to the identity.)

While this definition of pathological suffers from the drawback that there undoubtedly exist enumerations which are pathological with respect to this definition but which are not "absolutely pathological", it does serve the purpose of identifying those enumerations which present difficulties to current coset enumeration algorithms.

The enumeration of the cosets of subgroup $\langle h_1, \cdots, h_r \rangle$ in the group G will be denoted by $G \mid \langle h_1, \cdots, h_r \rangle$. If the subgroup is the identity, we write $G \mid E$. It will be seen later in this section that the order in which the relators of G are applied to a coset can have a significant effect on HLT and lookahead type programs. Thus, unless otherwise stated, it will be assumed that relators are applied to a coset in order of increasing relator length with relators having the same length being processed in the order in which they are written down. It should be noted that different implementations of a particular form of the TC algorithm (e.g. the Felsch algorithm) may lead to slight variations in M and T. As M_L and T_L are not unique, we shall denote the smallest value of M_L, for which an enumeration will complete, by $\hat{M}_L$ and the corresponding value of T_L by $\hat{T}_L$. Similarly, the corresponding execution time will be denoted by $\hat{t}_L$.

In this section, we use standard notation when writing generators and relations so that if w_1 and w_2 are words in the generators of a group G, we write $[w_1, w_2]$ for $w_1^{-1}w_2^{-1}w_1w_2$ and ${w_1}^{w_2}$ for $w_2^{-1}w_1w_2$. Also, if $\mathcal{P}$ is a set of relations and R is some word in the same generators as the words of $\mathcal{P}$, we write $\mathcal{P} + R$ for the relations $\mathcal{P}$, $R = 1$.

4.1. *Comparative Behaviour of Todd-Coxeter Programs.* In Tables 3 and 4, we compare the performance of the Felsch, lookahead and HLT forms of the TC algorithm. The HLT statistics were obtained by giving the lookahead program of Section 3.2 sufficient storage so that lookahead would not be used.* The Felsch algorithm was implemented using the same data structures and coincidence subroutine as the lookahead program of Section 3.2.

Table 3 contains a number of nonpathological enumerations. It can be seen that in, each case, $M_F = I$, and, except for $(4, 6|2, 12) + [a^{-1}, b]^3 \mid E$, $\hat{M}_L$ does not differ by more than one from I. Note the wide variation in τ_H, ranging from 1.00 for Weyl $B_6 \mid E$ to 7.57 for $\mathrm{PSL}_3(4) \mid \langle a \rangle$.

Table 4 contains a number of pathological enumerations. In eight out of twelve examples, $\hat{M}_L$ is less than M_F. In the case of $\mathrm{PSL}_2(11) \mid E$, $\tau_F = 1.61$, while $\hat{\tau}_L = 1.00$. The Neumann, Campbell and Macdonald presentations give rise to extremely pathological enumerations. The reason for this appears to be as follows. When the TC algorithm is doing an enumeration $G \mid H$, it must in effect find all the relations

* Note that, using the program of Section 3, M_H and T_H may vary slightly with the available space whenever T_H exceeds the number of cosets which can be stored in the available space.

TABLE 1

Definition of some sets of defining relations

Designation	*Order of group*	*Ref*	*Remarks*	*Defining relations*
E_1	1		Due to B. H. Neumann	$t^{-1}rtr^{-2} = r^{-1}srs^{-2} = s^{-1}tst^{-2} = 1.$
Cox	3,000	[4]		$a^6 = b^6 = (ab)^2 = (a^2b^2)^2 = (a^3b^3)^5 = 1.$
$B_{2,4}$	4,096	[6a]	2-generator Burnside group of exponent 4	$a^4 = b^4 = (ab)^4 = (a^{-1}b)^4 = (a^2b)^4 = (ab^2)^4 = (a^2b^2)^4 = (a^{-1}bab)^4 = (ab^{-1}ab)^4 = 1.$
S_7	5,040	[6]	Symmetric group S_7	$a^7 = b^2 = (ab)^6 = [a, b]^3 = [a^2, b]^2 = [a^3, b]^2 = 1.$
$PSL_2(11)$	660	[5]	Simple	$a^{11} = b^2 = (ab)^3 = (a^4ba^{-5}b)^2 = 1.$
$PSL_2(17)$	2,448	[6b]	Simple	$a^9 = b^2 = (ab)^4 = (a^2b)^3 = 1.$
$PSL_3(4)$	20,160	[18]	Simple	$a^5 = b^3 = (ab)^4 = (a^{-1}ba^{-1}b^{-1}ab)^3 = ba^2ba^{-2}b^{-1}a^{-2}b^{-1}a^2bab^{-1}a^{-1} = 1.$
$M_{11}^{(1)}$	7,920	[6a]	Mathieu simple group M_{11}	$a^{11} = b^5 = c^4 = (a^4c^2)^3 = (bc^2)^2 = (abc)^3 = b^{-1}aba^{-4} = c^{-1}bcb^{-2} = 1.$
$M_{11}^{(2)}$	7,920	[6b]	Mathieu simple group M_{11}	$a^{11} = b^5 = c^4 = (ac)^3 = c^{-1}bcb^{-2} = b^{-1}aba^{-4} = 1.$
J_2	604,800	[16]	Hall-Janko simple group	$a^3 = b^3 = c^3 = abab^{-1}a^{-1}b^{-1} = (ca)^5 = (cb)^5 = (cb^{-1}cb)^2 = a^{-1}baca^{-1}bac^{-1}a^{-1}b^{-1}ac^{-1} = aba^{-1}caba^{-1}c^{-1}ab^{-1}a^{-1}c^{-1} = 1.$
J_3	50,232,960	[10]	Higman-Janko-McKay simple group	$a^2 = c^2 = b^{15} = (ac)^3 = (bc)^2 = abab^{-4}ab^3 = s^2 = t^2 = (sa)^2 = (sc)^2 = (at)^2 = (bt)^3 = b^5tb^{-5}t = sbsb^{-4} = (ct)^4s = (b^2st)^3 = (b^{-2}ctb^4ct)^2 = b^2tb^{-1}abtb^{-2}a = b^{-2}ab^{-3}ctab^2ctb^3ab^3ctactb^7ab^4ct = 1.$
J_3^*	150,718,880	[16]	Maximal covering group of J_3	Generated by a, b, c, s, t and z with a, b, c, s and t subject to the same relations as in J_3 except that the last relator has z^{-1} appended and the following additional relators are added: $z^3 = [a, z] = [b, z] = [c, z] = [s, z] = [t, z] = 1.$
Neu	40,320	[19]		$a^3 = b^3 = c^3 = (ab)^5 = (a^{-1}b)^5 = (ac)^4 = (ac^{-1})^4 = ab^{-1}abc^{-1}acac^{-1} = (bc)^3 = (b^{-1}c)^4 = 1.$
Weyl B_6	46,080	[6]	Weyl group of Lie algebras B_6 and D_6	$a^2 = b^2 = c^2 = d^2 = e^2 = f^2 = (ab)^3 = (ac)^2 = (ad)^2 = (ae)^2 = (af)^2 = (bc)^3 = (bd)^2 = (be)^2 = (bf)^2 = (cd)^3 = (ce)^2 = (cf)^2 = (de)^3 = (df)^2 = (ef)^4 = 1.$

TABLE 2
Definition of families of defining relations

Designation	*Reference*	*Defining Relations*
$G^{p,q,r}$	Coxeter [5]	$a^p = b^q = c^r = (ab)^2 = (bc)^2 = (ca)^2 = (abc)^2 = 1.$
$(l, m \mid n, k)$	Coxeter [5]	$a^l = b^m = (ab)^n = (a^{-1}b)^k = 1.$
$(l, m, n; q)$	Coxeter [5]	$a^l = b^m = (ab)^n = [a, b]^q = 1.$
$((l, m, n; p))$	Coxeter [5]	$a^2 = b^2 = c^2 = (ab)^l = (bc)^m = (ca)^n = (abc)^p = 1.$
(l, m, n)		$a^l = b^m = (ab)^n = 1.$
$G(\alpha, \beta)$	Macdonald [15]	$b^{-1}a^{-1}bab^{-1}aba^{-\alpha} = a^{-1}b^{-1}aba^{-1}bab^{-\beta} = 1.$
Cam(n)	Campbell [3]	$r^{n-1}srsr^{-n}s^{-1} = s^{n-1}rsrs^{-n}r^{-1} = 1.$

which hold in G modulo H. Presumably then, the enumeration will be pathological when some of the relations modulo H can only be deduced by lengthy algebraic argument from the given defining relators for G. This is illustrated by the following example due to John Leech. The relations of $(8, 7 \mid 2, 3)$ imply $(a^2b^4)^6 = 1$, but this is very difficult to prove (Leech and Mennicke [14], Leech [11]). From Table 4 it can be seen that the enumeration $(8, 7 \mid 2, 3) \mid \langle a^2, a^{-1}b \rangle$ is pathological with $M_F = 1302$. However, M_F is only 448 for the enumerations

$$(8, 7 \mid 2, 3) + (a^2b^4)^6 \mid \langle a^2, a^{-1}b \rangle \quad \text{and} \quad (8, 7 \mid 2, 3) \mid \langle a^2, a^{-1}b, (a^2b^4)^6 \rangle.$$

The Macdonald presentations $G(\alpha, \beta)$ are highly pathological for most values of α and β. If the orders of the generators a and b are added as relations then there is a dramatic reduction in the degree of difficulty of the enumerations, suggesting that it is much easier for current versions of the TC algorithm to deduce relations in G from this extended relation set. Further, it appears to be a difficult task to deduce the orders of a and b in $G(\alpha, \beta)$ by hand, except in a few special cases.

The Macdonald groups are also the only family of groups we know which give rise to enumerations for which $\tau_H \ll \tau_F$. Thus the enumerations

$$G(\alpha, \beta) \mid \langle [a, b], [b, a^{-1}], [a^{-1}, b^{-1}], [b^{-1}, a] \rangle,$$

where $\alpha = -2, -1, 3, 4$ and β is odd, become extremely difficult for the Felsch algorithm as β increases but remain quite easy for the HLT algorithm. For example, if $\alpha = 3$, $\beta = 21$, then the index is 40 and $M_F = 16{,}063$, $T_F = 16{,}067$ while $M_H = 84$, $T_H = 91$. This is an example of a situation where the HLT method of introducing new cosets is far superior to the Felsch method. This was verified by Alford who produced a Felsch program which used the HLT method of introducing new cosets. Applying this program to the above enumerations resulted in their completion with M_F equal to the index. Unfortunately, when this program was applied to other groups, enumerations became much harder than when the usual Felsch method of introducing new cosets was used.

In the case of Cam(3), the addition of the orders of the generators r and s as extra relations does not significantly decrease the difficulty of the enumeration

TABLE 3

Comparative behaviour of Todd-Coxeter programs with respect to nonpathological enumerations
(Times are in UNIVAC 1108 *seconds using a packed version of the program of Section* 3)

Enumeration	I	M_F	$\hat{M}_L$	M_H	T_F	$\hat{T}_L$	T_H	τ_F	$\hat{\tau}_L$	τ_H	t_F	$\hat{t}_L$	t_H
$G^{3,7,-9} \mid E$	504	504	505	775	504	855	1,222	1.00	1.00	1.54	2.56	2.83	2.21
$(7, 7 \mid 2, 3) \mid E$	1,092	1,092	1,093	1,498	1,121	1,484	1,856	1.00	1.00	1.37	3.07	2.68	2.64
Cox $\mid E$	3,000	3,000	3,000	10,353	3,000	6,654	15,029	1.00	1.00	3.45	21.3	26.7	21.1
$(30, 30 \mid 3, 10) + a^3b^3 \mid E$	3,000	3,000	3,000	14,290	3,000	5,871	22,482	1.00	1.00	4.76	16.6	48.8	33.4
$\mathrm{PSL}_3(4) \mid \langle a \rangle$	4,032	4,032	4,033	30,537	4,655	8,134	34,439	1.00	1.00	7.57	49.8	41.7	43.3
$B_{2,4} \mid E$	4,096	4,096	4,097	12,576	5,022	6,561	15,024	1.00	1.00	3.00	61.7	50.3	29.3
$S_7 \mid E$	5,040	5,040	5,040	5,048	5,082	6,026	6,074	1.00	1.00	1.00	38.7	15.9	14.1
$(4, 6 \mid 2, 12) + [a^{-1}, b]^3 \mid E$	5,184	5,184	5,382	13,424	5,187	10,892	27,847	1.00	1.04	2.59	24.9	75.0	35.8
$(2, 3, 11; 4) \mid E$	6,072	6,072	6,073	8,268	6,101	9,725	17,522	1.00	1.00	1.36	19.7	22.0	24.0
$J_3 \mid \langle a, b, c, s \rangle$	6,156	6,156	6,157	>28,703	6,870	13,986	>35,932	1.00	1.00	>4.66	268	289	—
$J_3^* \mid \langle a, b, c, s \rangle$	18,468	18,468	18,468	37,900	19,351	41,163	115,349	1.00	1.00	2.05	886	890	—
Weyl $B_6 \mid E$	46,080	46,080	—	46,094	46,080	—	74,382	1.00	—	1.00	—	—	—

TABLE 4

Comparative behaviour of Todd-Coxeter programs with respect to pathological enumerations
(*Times are in* UNIVAC 1108 *seconds using a packed version of the program of Section* 3)

Enumeration	I	M_F	$\hat{M}_L$	M_H	T_F	$\hat{T}_L$	T_H	τ_F	$\hat{\tau}_L$	τ_H	t_F	$\hat{t}_L$	t_H
$E_1 \mid E$	1	588	695	1,649	588	758	1,705	588	695	1,649	10.5	2.39	2.81
$(2, 5, 7; 2) \mid E$	1	254	224	344	257	227	362	254	224	344	1.33	0.54	0.64
$G^{3,7,17} \mid \langle ab, c\rangle$	1	1,471	1,381	2,764	1,471	2,315	3,903	1,471	1,381	2,764	29.1	9.54	7.50
$\mathrm{PSL}_2(11) \mid E$	660	1,066	661	1,188	1,118	824	1,495	1.61	1.00	1.80	9.04	2.32	2.33
$(2, 3, 7; 7) \mid E$	1,092	1,590	2,286	6,132	1,648	2,880	8,826	1.46	2.09	5.61	6.83	10.2	11.7
$M_1^{(1)} \mid \langle a\rangle$	720	980	721	3,975	1,223	1,349	5,694	1.36	1.00	5.83	14.6	18.0	8.14
$(8, 7 \mid 2, 3) \mid \langle a^2, a^{-1}b\rangle$	448	1,302	1,241	2,253	1,306	1,422	2,602	2.91	2.77	5.02	8.53	3.45	3.96
$\mathrm{Neu} \mid \langle a, c\rangle$	240	4,439	4,553	29,272	4,740	7,158	42,013	18.5	18.9	122	240	70.0	—
$\mathrm{Cam}(3) \mid E$	120	1,638	2,189	4,375	1,660	2,206	4,396	13.6	18.2	36.4	23.7	3.81	5.77
$G^{3,7,16} \mid E$	21,504	75,547	—	116,550	75,548	—	256,946	3.51	—	5.42	—	—	—
$G(2, 4) \mid E$	3	6,812	2,973	10,881	6,864	3,255	12,393	2,271	991	3,627	163	7.45	15.6
$G(2, 6) \mid E$	5	19,597	4,194	18,558	19,627	4,582	20,138	3,919	839	3,712	850	13.4	24.0
$G(3, 3) \mid E$	16	109,538	29,007	>63,232	110,105	31,993	>71,031	6,850	1,813	>3,950	—	125	—

Cam(3) | E. However, the addition of the relation $(rs)^3 = 1$ causes M_F to drop to 120. The critical missing relations in the case of the group of Neumann are currently unknown.

On the basis of the evidence presented in Tables 3 and 4, it appears that the lookahead algorithm can usually perform an enumeration in about the same space as required by the Felsch algorithm and occasionally in significantly less space. We now compare the algorithms with respect to execution time. We first note that the ratio t_A/t_B of execution times for TC algorithms A and B can vary a certain amount from machine to machine. From Table 3, it can be seen that the Felsch and lookahead ($\hat{t}_L$) execution times are roughly comparable for nonpathological enumerations. The Felsch algorithm tends to be faster when the relators consist of words raised to fairly large powers (e.g., (30, 30 | 3,10) + a^3b^3 | E). In the case of pathological enumerations, Table 4 indicates that t_F is typically three times $\hat{t}_L$ (and t_H). How then does the lookahead algorithm compare with the HLT algorithm? We first note that if $M_L = M_H$ then the algorithms are the same so that times are identical in this case. As M_L is successively reduced until its minimum value $\hat{M}_L$ is reached the following behaviour of the corresponding execution time t_L is observed.

As M_L is reduced to a value slightly less than M_H so that lookahead has to be used, t_L increases slightly. Soon however, t_L begins to decrease with decreasing M_L until it reaches a minimum and then begins to increase again. The value of t_L continues to increase until $\hat{M}_L$ is reached, with the rate of increase becoming larger as one gets closer to $\hat{M}_L$. The final value of t_L, $\hat{t}_L$, ranges from a value significantly less than t_H to a value of two or three times greater than t_H on rare occasions. Two examples illustrating the behaviour of t_L are given in Table 5. In this table, l refers to the number of times lookahead is used. In Table 5(a), we see that for the enumeration (8, 7 | 2, 3) | $\langle a^2, a^{-1}b\rangle$, t_L first increases slightly and then decreases until it is about 75 per cent of t_H and finally in the space of 30 cosets increases to a value about 20 per cent greater than t_H. Table 5(b) summarizes the behaviour of t_L for the enumeration $M_{11}^{(1)}$ | $\langle a\rangle$. (Note that the relators of $M_{11}^{(1)}$ are not processed in order of increasing length.) Here, $\hat{t}_L$ is more than twice t_H.

The reasons for this variation in t_L are quite clear. The initial slight increase is due to the first use of lookahead. As M_L is reduced, T_L also gets smaller so that the total number of cosets that have to be processed decreases. However, as M_L approaches $\hat{M}_L$, the lookahead phase has to be used substantially more so that the average work done per coset roughly balances the reduction in the number of cosets introduced. To summarize, then, unless a value of M_L close to $\hat{M}_L$ is used, the time t_L will either be significantly less than t_H or close to t_H. If a value of M_L close to $\hat{M}_L$ is used, then, in some cases, t_L will be significantly greater than t_H.

Generally speaking, the Felsch algorithm is competitive with the HLT and lookahead algorithms with regard to execution time, in situations where the defining relators consist mainly of words raised to reasonable powers.

In practice, when using the lookahead program, one rarely chooses a value of M_L close to $\hat{M}_L$ (since one does not know $\hat{M}_L$ in advance), so that its execution time performance is usually much better than indicated in Tables 3 and 4. For example, if in the J_3^* enumeration of Table 3, M_L is set equal to 22,956, then t_L drops from 890 seconds to 472 seconds.

4.2. *Effect of Permuting Relators and Permuting the Letters of Relators on the*

TABLE 5(a)

Variation of time with maximum table size for $(8, 7 \mid 2, 3) \mid \langle a^2, a^{-1}b \rangle$, *with relators in the order presented*

(*Times are given in* CDC 6600 *seconds*)

M_L	2,176	2,100	1,900	1,700	1,500	1,300	1,250	1,240	1,225	1,220
T_L	2,626	2,531	2,288	2,009	1,744	1,498	1,448	1,461	1,441	1,437
t_L	0.927	0.934	0.880	0.822	0.766	0.712	0.704	0.898	0.898	1.093
l	0	1	1	1	1	1	1	2	2	3

TABLE 5(b)

Variation of time with maximum table size for $M_{11}^{(1)} \mid \langle a \rangle$

M_L	4,201	3,900	3,600	3,300	3,000	2,615	2,500	2,300	2,100	1,900
T_L	6,331	5,755	5,318	4,635	4,312	3,862	3,728	3,508	3,308	3,710
t_L	2.015	2.016	1.957	1.928	1.894	1.881	1.912	1.941	1.980	2.191
l	0	1	1	1	1	1	1	1	1	1
M_L	1,800	1,600	1,400	1,300	1,200	1,100	1,000	900	800	750
T_L	3,373	2,412	2,189	2,538	2,079	1,995	1,862	1,769	1,470	1,405
t_L	2.191	2.242	2.296	2.362	2.691	2.863	3.140	3.349	4.426	5.038
l	2	2	2	2	3	4	5	5	8	10

HLT *Method.* It has been found that the order in which the relators of a presentation are processed by the HLT algorithm can have a significant effect on the number of redundant cosets introduced. As a general rule, it is found that the best performance is achieved when the relators are processed in order of increasing length. Intuitively, the reason for this is that short relators are less likely to introduce redundant cosets than long ones. New information deduced when applying the short relators to coset i sometimes means a reduction in the number of new cosets which must be defined when applying the long relators to coset i.

We illustrate this effect with several related presentations for the group $((5, 4, 7; 3))$, which collapses to the identity, in Table 6. Recall that $((l, m, n; p)) = \mathrm{gp}\langle a, b, c \mid a^2 = b^2 = c^2 = (ab)^l = (bc)^m = (ca)^n = (abc)^p = 1 \rangle$. This can be written in the alternative form

$$((l, m, n; p))^* = \mathrm{gp}\langle x, y, z \mid x^m = y^n = z^p = (xy)^l = (yz)^2 = (zx)^2 = (xyz)^2 = 1 \rangle.$$

In the table, T_H denotes the total number of cosets defined when the relators are applied to a coset in the order in which they are written down above, and T_H^0, the total number of cosets when the relators are applied in order of increasing length. The enumerations are all performed over the trivial subgroup. In case 2, we see that the number of cosets defined has dropped by 19 per cent.

It should be noted that applying relators in order of increasing length does not always lead to the fewest cosets being defined. However, in most situations where we have found that it is not the optimal ordering, we have observed that the difference in total cosets defined between the optimal arrangement of relators and the ordered arrangement is insignificant. Thus, the best strategy is to set the relators up so that they are processed in order of increasing length.

TABLE 6

Effect of ordering relators

	Enumeration	T_H	T_H^0	$(T_H - T_H^0)/T_H$
1	((7, 5, 4; 3))* | E	27,082	22,615	0.16
2	((4, 5, 7; 3))* | E	16,232	13,199	0.19
3	((5, 4, 7; 3))* | E	15,510	12,681	0.18
4	((5, 4, 7; 3)) | E	6,012	5,760	0.04
5	((4, 5, 7; 3)) | E	6,018	6,036	0.00

TABLE 7

Effect of permuting the letters of the relators of $G(2, 2)$ *on* T_H

	0	1	2	3	4	5	6	7	8
0	459	516	611	850	293	422	514	703	355
1	460	664	824	935	518	497	757	997	592
2	493	532	521	915	490	629	2219	1153	435
3	1211	352	507	1530	513	855	1737	1466	623
4	345	296	309	504	587	532	514	377	315
5	341	579	531	651	511	1034	701	607	273
6	438	1174	1440	1079	756	721	1421	732	375
7	432	562	891	1799	396	624	872	625	400
8	398	488	351	733	379	344	497	595	365

If R is a relator in G, then any cyclic permutation R' of R is obviously a relator in G equivalent to R. If we consider the set of presentations for G obtained by taking different relative cyclic permutations of the letters of the relators of a presentation $\mathcal{P}$ of G, then, sometimes, a large variation in T_H is noted. For example, consider the trivial Macdonald group $G(2, 2)$ written

$$G(2, 2) = \text{gp}\langle a, b \mid b^{-1}a^{-1}bab^{-1}aba^{-2} = a^{-1}b^{-1}aba^{-1}bab^{-2} = 1\rangle.$$

Table 7 contains the values of T_H for all distinct permutations of the two relators of $G(2, 2)$. The ith column contains the effect of rotating relator 1 left i places while the jth row contains the effect of rotating relator 2 left j places.

The best result, $T_H = 273$ (row 5, column 8), is obtained when $G(2, 2)$ is written

$$G(2, 2) = \text{gp}\langle a, b \mid a^{-1}b^{-1}a^{-1}bab^{-1}aba^{-1} = bab^{-2}a^{-1}b^{-1}aba^{-1} = 1\rangle,$$

i.e., when the two relators agree on the last four letters. The worst result, $T_H = 2{,}219$ (row 2, column 6), is obtained for

$$G(2, 2) = \text{gp}\langle a, b \mid ba^{-2}b^{-1}a^{-1}bab^{-1}a = aba^{-1}bab^{-2}a^{-1}b^{-1} = 1\rangle.$$

Note that there is a factor of 8 between the best and worst results! The minimum values of T_H are obtained when the relators are permuted so that one of the following arrangements results:

(a) The first four letters of each relator are identical.

(b) The last four letters of each relator are identical.

(c) The word formed by the first four letters of one relator is the inverse of the word formed by the last four letters of the other relator.

These arrangements imply that, usually, fewer cosets have to be defined in order to complete a relator cycle at coset i. That this behaviour is not peculiar to groups having a very small number of relators can be seen by considering the following presentation for the Hall-Janko group J_2 (McKay and Wales [16]):

$$J_2 = \mathrm{gp}\langle a, b, c \mid a^3 = b^3 = c^3 = abab^{-1}a^{-1}b^{-1} = (ca)^5 = (cb)^5 = (cb^{-1}cb)^2 = a^{-1}baca^{-1}bac^{-1}a^{-1}b^{-1}ac^{-1} = aba^{-1}caba^{-1}c^{-1}ab^{-1}a^{-1}c^{-1} = 1\rangle.$$

Enumerating the cosets of $\langle a, c, b^{-1}cb\rangle$ of index 315 and considering all permutations of the last two relators, we find that T_H varies from 1575 to 4255 cosets. The best result is obtained when the second last relator is rotated left 7 places, the last relator is rotated left 11 places, and the order of the two relators is interchanged. The worst result is obtained when the second last relator is rotated left 1 place and the last relator rotated left 5 places.

It should also be emphasized that the order in which relators of equal length are processed can have a major effect on M_H and T_H. For example, still considering $J_2 \mid \langle a, c, b^{-1}cb\rangle$, if we rotate the second last relator left 7 places and the last relator left 5 places, we get $M_H = 931$, $T_H = 2{,}038$. If now the order of these two rotated relators is interchanged, we get $M_H = 2{,}380$, $T_H = 3{,}357$.

Finally, we note that if the inverses of the two relators of Cam(3) are used rather than the given relators, then M_H increases by 20 per cent while $\hat{M}_L$ increases by about 10 per cent.

The operations of changing the order in which relators are processed and permuting the letters of relators have little or no effect on the Felsch algorithm. However, as the lookahead algorithm defines cosets the same way as the HLT algorithm when in the defining phase, these two operations can have an important effect on the lookahead algorithm, and in particular on the value of $\hat{M}_L$.

4.3. *Effect of Redundant Relators.* Given a set of defining relations $\mathcal{P}$ for a group G, a relation $R \in \mathcal{P}$ is said to be *redundant* with respect to $\mathcal{P}$ if R can be deduced from the set of relations $\mathcal{P}$ with the relation R deleted. The presence of a number of redundant relators in a presentation often facilitates the work of the TC algorithm, particularly, if the redundant relators are not trivial consequences of the other relators. The reflection group Weyl B_6 and the Janko group J_3 in Section 4.1 are illustrations of this. In Section 4.1, we also noted the effect of adding certain critical redundant relations to some pathological enumerations.

On the other hand, the presence of too many long redundant relators can cause the definition of unnecessary cosets apart from the extra processing time involved. This is nicely illustrated by the following presentation for $\mathrm{PSL}_2(25)$ of order 7,800, which is due to Beetham [1].

$$\begin{aligned}\mathrm{PSL}_2(25) = \mathrm{gp}\langle a, b, c, d \mid a^5 = b^5 = c^5 = d^5 = [a, b] = [c, d]\\ = (ac)^2 = (ab^{-1}d^{-2})^2 = (a^2c^{-2})^2\\ = (b^2c^{-1}d)^2 = (bc^{-2}d^2)^2 = (a^{-2}bcd)^2\\ = (a^{-1}bc^2d^{-1})^2 = (a^2b^{-2}d^{-1})^2\\ = (a^{-2}b^{-1}c^{-1}d^2)^2 = (ab^{-2}c^2d)^2\\ = (a^2b^2c^{-1}d^{-2})^2 = (ab^2c^{-2}d^{-2})^2 = 1\rangle.\end{aligned}$$

The last five relators are redundant. Let us denote the presentation of $\mathrm{PSL}_2(25)$ formed by removing the last i relators from the above presentation by $P^{(i)}$, where $i = 0, 1, 2, 3, 4, 5$. We summarize the effect of removing the last five relators one by one in Table 8. In each case the enumerations were performed over the subgroup $H = \langle a, b, d^{-1}b^{-1}c^2\rangle$, which is the normalizer of the Sylow 5-subgroup.

It can be seen that the presence of the fourth last relator makes the enumeration much easier. However, the addition of further redundant relators causes the enumeration to become steadily more difficult. In general, a pathological presentation $\mathcal{P}$ can be greatly improved by the addition of the right redundant relators. (See Section 4.1.) Although one may have no idea as to what the critical missing relators are, it is sometimes worthwhile adding any further known relators which are not immediately derivable from $\mathcal{P}$.

On the other hand, the addition of trivially redundant relators, such as cyclic permutations and inverses of cyclic permutations of relators, to a presentation, seldom helps in the case of a pathological enumeration. As far as the HLT method is concerned, the additional relators will change the values of M_H and T_H, because of the different order of coset definition. However, this change is not necessarily an improvement and the execution time usually increases because of the increased processing per coset. Even if such redundant relators are used only in the lookahead phase, little advantage is gained, for if a cyclic permutation of relator R gives a deduction when applied to coset i, then that same deduction must be made when relator R itself is applied to some other coset j.

Unfortunately, it is usually difficult to derive useful additional relations by hand. However, it is often possible to use the coset enumeration process itself to derive additional relations. A simple minded method of doing this consists in doing a partial

TABLE 8

Effect of removing redundant relators on a presentation for $\mathrm{PSL}_2(25)$

Enumeration	I	M_H	T_H	τ_H	ϵ_H
$P^{(0)} \mid H$	26	986	1416	38	54
$P^{(1)} \mid H$	26	983	1362	38	52
$P^{(2)} \mid H$	26	789	1075	30	41
$P^{(3)} \mid H$	26	617	853	24	33
$P^{(4)} \mid H$	26	487	673	19	26
$P^{(5)} \mid H$	26	929	1177	36	45

enumeration of the cosets of the identity using the Felsch method and examining those cosets which are found to be coincident. If i and j are coincident cosets, it is a simple matter to read from the partial coset table, coset representatives $w(i)$ and $w(j)$ for i and j, to obtain the relation $w(i) = w(j)$. The most promising of such relations are added to $\mathcal{P}$ and either the above process is repeated to deduce further relations or the enumeration of cosets of the desired subgroup is attempted.

As an example of this process, consider the presentation (due to Brunner [2]):

$$C_{29} = \text{gp}\langle a, b \mid bab^2(ab^{-1})^2a^2 = (b^2a)^2baba^{-2} = [a, b]^2 = 1\rangle$$

of the cyclic group of order 29. Using Felsch's method to enumerate the cosets of the identity, 15 redundant cosets were found during the definition of 14,000 cosets. One of the corresponding relators found was

$$a^4ba^{-2}b^3a^{-1}b^2.$$

Enumerating the cosets of $\langle a\rangle$ (index 1 in C_{29}), we find that M_F drops from 1084 to 557 with the addition of the above relator.

The Felsch algorithm is preferred when trying to deduce new relations this way because it minimizes the number of redundant cosets corresponding to unimportant relations in the group.

A more sophisticated method of deriving new relations from coincidences is described in Leech [13]. Unlike the above method, his method can be used to derive relations corresponding to coincidences which occur when enumerating the cosets of nontrivial subgroups.

4.4. *Behaviour of the* TC *Algorithm with Respect to Different Presentations of the Same Group.* It is of considerable interest to take a number of different presentations for some group G and study the behaviour of the TC algorithm over this set of presentations.

We begin by examining the effect of the length of relators. Intuitively, we would expect that the presence of long relators would lead to the introduction of proportionally more redundant cosets and this effect is illustrated by the four presentations of the symmetric group S_5 of order 120, given in Table 9. In each case, the cosets of the identity have been enumerated and we tabulate the number of Felsch redundant cosets R_F, and the number of HLT redundant cosets R_H. In addition, we tabulate the maximum relator length m and the mean relator length $\bar{m}$. Note, in particular, the behaviour of the first three presentations which are virtually identical except for the last relator.

In Table 10, we compare the behaviour of the algorithm with respect to eleven different presentations of the simple group $\text{PSL}_2(13)$ of order 1092. In each case, the cosets of the identity have been enumerated. It can be seen that while there is considerable correlation between the degree of difficulty for the Felsch and HLT algorithms, the correlation is by no means complete. For example, though presentation 1 is by far the worst for Felsch, it is nowhere near the worst for the HLT method. Also, there is a fairly high correlation between m and M_H. (For a discussion of some of these presentations see Leech [12].)

As noted earlier, the HLT algorithm is particularly susceptible to the ordering of the relators and the relative ordering of their letters. Thus, we would expect that, if these effects could be removed, there would be an even higher correlation between

TABLE 9

Behaviour of algorithms with respect to different presentations of S_5

	Presentation	m	$\bar{m}$	R_F	R_H
1	$\langle a^4 = b^6 = (ab)^2 = (a^{-1}b)^3 = 1 \rangle$	6	5	0	3
2	$\langle a^5 = b^6 = (ab)^2 = (a^2b^2)^2 = 1 \rangle$	8	5.75	7	35
3	$\langle a^5 = b^4 = (ab)^2 = (a^2b^2)^3 = 1 \rangle$	12	6.25	9	112
4	$\langle a^2 = b^5 = (ab)^4 = [a, b]^3 = 1 \rangle$	12	6.75	7	26

TABLE 10

Behaviour of algorithms with respect to different presentations of $\mathrm{PSL}_2(13)$. *Each enumeration is over the identity so that the index is* 1092 *in each case*

	Presentation	m	$\bar{m}$	M_F	M_H	T_F	T_H	τ_F	τ_H
1	$G^{3,7,13}$	13	6	2,519	4,654	2,519	7,509	2.30	4.3
2	$\langle a^7 = b^2 = (ba)^3 = [b, a]^7 = 1 \rangle$	28	11	1,732	5,455	1,824	6,784	1.58	5.0
3	$\langle a^2 = b^3 = (ab)^7 = [a, b]^7 = 1 \rangle$	28	12	1,590	6,132	1,648	8,826	1.45	5.6
4	$\langle a^2 = b^3 = c^7 = abc = (cba)^7 = 1 \rangle$	21	7	1,565	10,692	1,886	17,211	1.43	9.8
5	$\langle a^3 = b^7 = (ab)^2 = (a^2b^2)^7 = 1 \rangle$	28	10	1,578	12,782	1,619	23,082	1.44	11.7
6	$\langle a^7 = b^2 = (ab)^3 = (a^3b)^7 = 1 \rangle$	28	11	1,399	3,337	1,534	4,408	1.28	3.1
7	$\langle a^7 = b^7 = (ab^3)^2 = (ab^2)^3 = 1 \rangle$	9	8	1,271	1,869	1,506	2,454	1.16	1.7
8	$\langle a^7 = b^2 = (a^2b)^3 = (ab)^7 = 1 \rangle$	14	8	1,092	1,245	1,180	1,587	1.00	1.1
9	$\langle a^7 = b^7 = (ab)^2 = (a^{-1}b)^3 = 1 \rangle$	7	6	1,092	1,498	1,121	1,856	1.00	1.4
10	$\langle a^7 = b^2 = (ab)^6 = (a^2b)^3 = 1 \rangle$	12	7	1,092	1,092	1,153	1,264	1.00	1.0
11	$\langle a^2 = b^3 = (ab)^7 = [a, b]^6 = 1 \rangle$	24	11	1,092	1,530	1,107	2,848	1.00	1.4

m and M_H and between τ_F and τ_H. Then, we are faced with the central problem: Why are the first seven presentations bad compared to the last four? In particular, consider presentations 3 and 11, where the only difference is that the last relator in 3 is $[a, b]^7$ while in 11 it is $[a, b]^6$.

We conclude this section with two further observations. The effect of adding certain redundant relations to the presentations $G(\alpha, \beta)$ and Cam(3) has been noted in Section 4.1. Secondly, if the relator $(a^{-1}bab)^4$ in the presentation of the Burnside group $B_{2,4}$ is replaced by the relator $[a, b]^4$ (still giving the same group) when enumerating the cosets of the identity, then T_F drops from 5,022 to 4,282 (M_F is 4,096 in both cases), while M_H drops from 12,386 to 7,333.

4.5. *Behaviour of the Algorithm with Respect to Different Subgroups of a Fixed Group.* In this section, we suppose that we are given a fixed presentation of a group G and we shall consider two questions:

(a) How does the difficulty of an enumeration depend upon the subgroup chosen?

(b) How does the difficulty of an enumeration depend upon the subgroup generators used?

The answer to the first question, in general, is that τ_H and τ_F remain fairly constant with varying index. This is illustrated by the groups $G^{3,7,13}$, $M_{11}^{(2)}$ and $(8, 7 \mid 2, 3)$ in Table 11. Note, however, that the enumerations

TABLE 11

Behaviour of the algorithm with respect to different subgroups

Enumeration	I	M_F	M_H	T_F	T_H	τ_F	τ_H
$G^{3,7,13} \mid E$	1,092	2,519	4,654	2,519	7,509	2.30	4.3
$G^{3,7,13} \mid \langle a \rangle$	364	859	1,628	859	2,774	2.36	4.5
$G^{3,7,13} \mid \langle b \rangle$	156	378	739	379	1,182	2.42	4.7
$G^{3,7,13} \mid \langle c \rangle$	84	154	354	154	509	1.83	4.2
$M_{11}^{(2)} \mid \langle ac \rangle$	2,640	3,878	8,952	5,165	16,292	1.47	3.4
$M_{11}^{(2)} \mid \langle c \rangle$	1,980	2,580	6,027	3,843	13,292	1.30	3.0
$M_{11}^{(2)} \mid \langle b \rangle$	1,584	2,851	5,134	3,419	10,386	1.80	3.2
$M_{11}^{(2)} \mid \langle a \rangle$	720	1,562	3,027	1,932	5,130	2.17	4.2
$M_{11}^{(2)} \mid \langle a^4, b, c^2 \rangle$	12	36	69	39	82	3.00	5.7
$(8, 7 \mid 2, 3) \mid E$	10,752	26,270	49,301	26,314	57,468	2.45	4.6
$(8, 7 \mid 2, 3) \mid \langle b \rangle$	1,536	4,583	7,058	4,585	8,252	2.99	4.6
$(8, 7 \mid 2, 3) \mid \langle a \rangle$	1,344	2,667	5,580	2,688	6,452	1.98	4.1
$(8, 7 \mid 2, 3) \mid \langle a^2, a^{-1}b \rangle$	448	1,302	2,253	1,306	2,602	2.90	5.0
$\mathrm{PSL}_2(17) \mid E$	2,448	2,448	2,448	2,677	2,560	1.00	1.0
$\mathrm{PSL}_2(17) \mid \langle ab \rangle$	612	612	621	661	640	1.00	1.0
$\mathrm{PSL}_2(17) \mid \langle [b, a][b, a^2] \rangle$	144	258	343	271	345	1.79	2.4
$J_2 \mid \langle a, c, c^b \rangle$	315	315	587	376	1,853	1.00	1.9
$J_2 \mid \langle a, b, b^{ca^{-1}c} \rangle$	100	1,305	12,907	1,315	14,800	13.05	129

$$M_{11}^{(2)} \mid \langle a^4, b, c^2 \rangle \quad \text{and} \quad (8, 7 \mid 2, 3) \mid \langle a^2, a^{-1}b \rangle$$

are slightly more difficult than the others. The enumeration $\mathrm{PSL}_2(17) \mid \langle [b, a][b, a^2] \rangle$ is significantly more difficult than other enumerations in that group over cyclic subgroups. So, we tentatively conclude that the complexity of a word taken as a subgroup generator has an influence on the degree of difficulty of an enumeration. The two enumerations in J_2 apparently demonstrate that in some situations the choice of subgroup has a great effect on the difficulty of an enumeration. However, further experiments lead us to believe that this effect is caused by the choice of subgroup generators rather than the choice of subgroup. Note that the subgroup $\langle a, b, b^{ca^{-1}c} \rangle$ is isomorphic to the unitary simple group $U_3(3)$ while the subgroup $\langle a, c, c^b \rangle$ is the centralizer of an involution. As yet we have no knowledge of how the embedding of subgroup H in G affects the difficulty of enumerations $G \mid H$.

Regarding question (b), we have found that the choice of the generating set for the subgroup H can have a great effect on the enumeration $G \mid H$. This is illustrated in Table 12 by tabulating the values of M_H for four different sets of generators of the normal closure N of element a in Men(5) (recall that Men(n) was defined in Section 1). Here, we see that while N is generated by a and a^b alone, the addition of redundant generators a^c, a^d, a^e decreases M_H by a factor of 28. A further example is provided by the enumeration $(8, 7 \mid 2, 3) \mid \langle a^2, a^{-1}b \rangle$ where, as noted in Section 4.2, the addition of the redundant subgroup generator $(a^2b^4)^6$ greatly decreases the difficulty of the enumeration. In the case of the Felsch algorithm, V. Felsch has observed

TABLE 12

Effect of redundant subgroup generators on an enumeration in Men(5)

Subgroup generators	I	M_H	T_H	τ_H
$\langle a, a^b \rangle$	4	694	798	173
$\langle a, a^b, a^c \rangle$	4	83	109	21
$\langle a, a^b, a^c, a^d \rangle$	4	83	108	21
$\langle a, a^b, a^c, a^d, a^e \rangle$	4	25	31	6

that the use of the defining relators as redundant subgroup generators seldom hurts and occasionally has the effect of drastically reducing the number of redundant cosets for a difficult enumeration. For example, in the enumeration $B_{2,4} \mid E$, the number of redundant cosets introduced is more than halved.

The enumeration $G \mid H$ involves constructing a set of generators for a subgroup $\bar{H}$ of the free group F of rank r (assuming that G is given as an r-generator group) such that the coset table of $\bar{H}$ in F is isomorphic to the coset table of H in G. As the images in F of the Schreier generators of H in G generate $\bar{H}$, we would expect that, if we used the Schreier generators of H, then the enumeration $G \mid H$ would be particularly easy. That this is so in practice is demonstrated by the enumeration $J_2 \mid \langle a, b, b^{ca^{-1}c} \rangle$. Using a complete set of 201 Schreier generators for the subgroup instead of a, b and $b^{ca^{-1}c}$, M_H dropped from 12,907 to 101!

John McKay reports that if a number of complicated generators are taken for H, then the number of redundant cosets occurring in the enumeration $G \mid H$ can depend significantly on the order in which the generators of H are applied to coset 1. He suggests that, before proceeding with the enumeration proper, the user should apply different permutations of the generators of H to coset 1 and take that permutation for which the number of cosets introduced so far is a minimum.

4.6. *The Use of External Storage.* It is clear that the next major step in the evolution of coset enumeration programs must be the development of forms of the algorithm which operate efficiently when the coset table is kept on some external storage medium such as disk. The present algorithms access the table in a semi-random fashion so that the task of developing a form of the algorithm for use with external storage will be rather difficult. To illustrate some of the problems, we shall describe an experiment in which HLT and Felsch programs running on an IBM 360/50, were modified so that the coset table was kept on an IBM 1316 disk pack.

The coset table is broken up into blocks, a small number of which may be resident in the core memory at any instant. Whenever a reference is made to a row of the coset table not already in core, the block on disk containing this row must be read into core, while some block currently in core must be written onto disk in order to make room for the new block. The first major choice which has to be made is the block size, where a block is to consist of a fixed number of rows of the coset table. Table 13 demonstrates the effect of different block sizes on the HLT and Felsch algorithms. It is seen that while the number of reads from disk is roughly constant with decreasing block size the execution time drops sharply, because of faster transfer times, so that smaller block sizes are to be preferred.

TABLE 13

Behaviour of the disk storage versions of the HLT *and Felsch algorithms with respect to varying block sizes*

Enumeration	*Method*	*Cosets per block*	*Cosets in core*	*Reads from disk*	*Time*
$G^{3,7,12}\|E$	HLT	113	4,184	1,204	23^m
$G^{3,7,12}\|E$	HLT	54	4,222	1,079	9^m
$G^{3,7,12}\|E$	HLT	35	4,305	1,079	8^m
$G^{3,7,12}\|E$	HLT	25	4,300	1,116	7^m
$(5, 2, 6; 6) + (a^2b)^4\|E$	Felsch	90	3,870	28,350	227^m
$(5, 2, 6; 6) + (a^2b)^4\|E$	Felsch	20	3,860	27,148	112^m
$(5, 2, 6; 6) + (a^2b)^4\|E$	Felsch	11	3,861	29,425	109^m

With virtually random access throughout the coset table, the least recently used algorithm was used when replacing a block in core with a block from disk. The number of writes to disk can be reduced by associating a flag with each block resident in core to indicate whether the block has been changed since its arrival in core. If it has not been changed at the time of its replacement, it is, of course, unnecessary to copy it onto the disk. This can reduce the number of disk writes by up to a factor of 2. For further details see Watson [22].

Table 14 contains the statistics from a number of enumerations using disk. It can be seen that the slower processing of the Felsch algorithm is more than made up for by the smaller ratio of total cosets to cosets in core, so that the Felsch execution times are significantly better than the HLT execution times. Even so, the Felsch execution times are still unacceptably large. At the time of writing, no similar experiments have been carried out with a lookahead program but it is clear that new techniques will have to be used if we are to develop economic disk-based coset enumeration programs.

5. Conclusion. We have constructed a simple but powerful coset enumeration algorithm, the lookahead algorithm, by adapting the HLT process so that when storage space is exhausted, the algorithm halts the definition of new cosets but continues to apply the relators to cosets in the hope of discovering coincidences. If coincidences are found, the corresponding space becomes free so that the definition of new cosets can proceed. The performance of the algorithm is further improved by setting up the coset table as a doubly linked list so that free rows corresponding to coincidences can be readily recovered.

In a large number of tests, we have found that the lookahead algorithm is quite competitive with the Felsch algorithm with respect to the minimum amount of store required to complete an enumeration. Further experiments have shown that the HLT algorithm is usually faster than the Felsch algorithm and that the lookahead algorithm is as fast or faster than the HLT algorithm except occasionally when one is near the minimum store necessary for the enumeration.

TABLE 14

Behaviour of the disk storage versions of the HLT *and Felsch algorithms*

Enumeration	$G^{3,7,16} \mid E$	$G^{3,7,16} \mid E$	$G^{3,7,17} \mid E$	$G^{3,7,17} \mid E$	$((5, 5, 6; 3)) + (abcb)^6 \mid A_5$	$((5, 5, 6; 3)) + (abcb)^6 \mid A_5$	Men (5) $\mid \langle b, c \rangle$
Index	21,504	21,504	1	1	30,720	30,720	55,104
Method	Felsch	HLT	Felsch	HLT	Felsch	HLT	Felsch
M	75,547	~116,550	75,454	~135,600	30,720	~51,400	62,212
T	75,548	256,946	75,469	224,412	30,878	208,175	96,451
Reads from disk	167,456	~312,100	105,274	172,817	109,932	~405,000	1,047,287
Writes to disk	137,683	164,274	89,739	84,444	103,624	~121,000	668,491
Cosets per block	25	35	25	35	25	35	17
Total blocks: blocks in core	17:1	60:1	18:1	52:1	8:1	48:1	31:1
Average usage of a block while in core	42	37	70	60	58	22	20
CPU *time*	8^h22^m	15^h30^m	6^h10^m	9^h15^m	6^h14^m	16^h30^m	42^h13^m
Wait time mechanical	5^h48^m	9^h20^m	3^h45^m	5^h0^m	2^h13^m	10^h6^m	37^h18^m

Further experiments have led to the discovery of a number of presentations and classes of presentations which cause the Todd-Coxeter algorithm to perform very badly. The existence of such presentations is, of course, related to the unsolvability of the word problem for groups. In addition, we have discovered that

(a) structurally similar presentations can lead to large variations in enumeration difficulty;

(b) the order in which relators are processed and the relative ordering of the letters of relators may have a significant effect on the performance of the HLT (and hence lookahead) algorithms;

(c) the presence of certain redundant relators can have a major effect on the difficulty of an enumeration;

(d) the total number of cosets defined in an enumeration is usually proportional to the index of a subgroup when one is considering enumerations of different subgroups with respect to a fixed presentation. The choice of generators for a particular subgroup and the order in which they are processed can have an important effect on the enumeration.

The two major problems in the area of coset enumeration at present are the need for an adequate theory which would enable us to explain in group theoretic terms why an arbitrary enumeration is hard or easy, and the lack of a form of the algorithm which makes effective use of secondary storage. It is our hope that the work described in this paper will form the starting point for attempts at the solution of these two problems.

Acknowledgements. We would like to thank the numerous people who contributed ideas and information during this study. In particular, we would like to thank John Leech for critically reading a preliminary version of this manuscript and Bill Alford for recomputing many of the statistics presented here.

This work was begun by Cannon and Dimino at Bell Telephone Laboratories, Murray Hill, during 1969/1970. During 1970, Watson began a similar project at the Australian National University and this continued through 1971. Cannon and Havas continued the work at the University of Sydney during 1971, supported by the Australian Research Grants Committee, the Commonwealth Scientific and Industrial Research Organization, and Control Data of Australia. We would like to express our gratitude to each of these bodies for their support.

Department of Pure Mathematics
University of Sydney
Sydney, Australia

Bell Laboratories
Murray Hill, New Jersey

Department of Pure Mathematics
University of Sydney
Sydney, Australia

Department of Mathematics
Institute of Advanced Studies
Australian National University
Canberra, Australia

1. M. J. BEETHAM, "A set of generators and relations for the groups PSL(2,q), q odd," *J. London Math. Soc.*, v. 3, 1971, pp. 544–557. MR **44** #2806.

2. A. BRUNNER, Personal communication.

3. C. M. CAMPBELL, "Some examples using coset enumeration," in *Computational Problems in Abstract Algebra* (Edited by John Leech), Pergamon Press, New York, 1970, pp. 37–41. MR **40** #7341.

4. H. S. M. COXETER, "The abstract groups $R^m = S^m = (R^jS^j)^{p_j} = 1$, $S^m = T^2 = (S^jT)^{2p_j} = 1$ and $S^m = T^2 = (S^{-j}TS^jT)^{p_j} = 1$," *Proc. London Math. Soc.* (2), v. 41, 1936, pp. 278–301.

5. H. S. M. COXETER, "The abstract groups $G^{m,n,p}$," *Trans. Amer. Math. Soc.*, v. 45, 1939, pp. 73–150.

6a,b. H. S. M. COXETER & W. O. J. MOSER, *Generators and Relations for Discrete Groups*, Springer-Verlag, Berlin, 1957; 2nd ed., Ergebnisse der Mathematik und ihrer Grenzgebiete, Neue Folge, Band 14, Springer-Verlag, Berlin and New York, 1965. MR **19,** 527; MR **30** #4818.

7. L. A. DIMINO, "A graphical approach to coset enumeration," *SIGSAM Bulletin,* v. 19, 1971, pp. 8–43.

8. H. FELSCH, "Programmierung der Restklassenabzählung einer Gruppe nach Untergruppen," *Numer. Math.*, v. 3, 1961, pp. 250–256. MR **24** #B488.

9. M. J. T. GUY, *Coset Enumeration*, Lecture delivered at the Conference on Computational Problems in Abstract Algebra, Oxford, 29 August–2 September, 1967.

10. G. HIGMAN & J. MCKAY, "On Janko's simple group of order 50,232,960," *Bull. London Math. Soc.*, v. 1, 1969, pp. 89–94. MR **40** #224.

11. J. LEECH, "Coset enumeration on digital computers," *Proc. Cambridge Philos. Soc.*, v. 59, 1963, pp. 257–267. MR **26** #4513.

12. J. LEECH, "Generators for certain normal subgroups of (2,3,7)," *Proc. Cambridge Philos. Soc.*, v. 61, 1965, pp. 321–332. MR **30** #4821.

13. J. LEECH, "Coset enumeration," in *Computational Problems in Abstract Algebra* (Edited by John Leech), Pergamon Press, New York and Oxford, 1970, pp. 21–35. MR **40** #7343.

14. J. LEECH & J. MENNICKE, "Note on a conjecture of Coxeter," *Proc. Glasgow Math. Assoc.*, v. 5, 1961, pp. 25–29. MR **27** #196.

15. I. D. MACDONALD, "On a class of finitely presented groups," *Canad. J. Math.*, v. 14, 1962, pp. 602–613. MR **25** #3992.

16. J. MCKAY & D. WALES, "The multipliers of the simple groups of order 604,800 and 50,232,960," *J. Algebra*, v. 17, 1971, pp. 262–272. MR **43** #340.

17. N. S. MENDELSOHN, "An algorithmic solution for a word problem in group theory," *Canad. J. Math.*, v. 16, 1964, pp. 509–516; Correction, *Canad. J. Math.*, v. 17, 1965, p. 505. MR **29** #1248; **31** #237.

18. J. L. MENNICKE & D. GARBE, "Some remarks on the Mathieu groups," *Canad. Math. Bull.*, v. 7, 1964, pp. 201–212. MR **29** #150.

19. B. H. NEUMANN, Personal communication.

20. J. A. TODD & H. S. M. COXETER, "A practical method for enumerating cosets of a finite abstract group," *Proc. Edinburgh Math. Soc.*, v. 5, 1936, pp. 26–34.

21. H. F. TROTTER, "A machine program for coset enumeration," *Canad. Math. Bull.*, v. 7, 1964, pp. 357–368; Program listing in *Mathematical Algorithms*, v. 1, 1966, pp. 12–18. MR **29** #5415.

22. J. M. WATSON, *An Extended Todd-Coxeter Algorithm*, Department of Mathematics, Institute of Advanced Studies, Australian National University, Canberra, 1971, 15 pages.

[Bibliography]

[5] **The Abstract Groups G m,n,p**
H. S. M. Coxeter
Transactions of the American Mathematical Society, Vol. 45, No. 1. (Jan., 1939), pp. 73-150.
Stable URL:
http://links.jstor.org/sici?sici=0002-9947%28193901%2945%3A1%3C73%3ATAG%3E2.0.CO%3B2-M