AN ECONOMICAL PROGRAM FOR LIMITED PARSING OF ENGLISH

D. C. Clarke and R. E. Wall*
IBM San Jose Research Laboratory
San Jose, California

Automatic syntactic analysis has often been proposed as a component of mechanized indexing systems.1,2 However, up to this time, frequency counting and statistical association techniques have been more favored, since these involve operations which can be performed with great speed on present day computers. Syntactic analysis programs,3 especially the few which have relatively complete grammars, have suffered from the disadvantage of slow and expensive operation and consequently have seldom been applied beyond the field of mechanical translation. In this paper, we report the design and testing of a limited syntactic recognition program for English which shows promise of becoming accurate enough to aid in mechanized indexing, yet sufficiently inexpensive to make large-scale use practicable.

We originally developed this system as an extension of Baxendale's Title Analyzer Program,4 which used a small number of syntactic clues and a discard list to select "significant" words and phrases from titles of technical articles for use as index terms. However, the shallowness of an index produced only from titles seriously limited the applicability of the Baxendale program, so it seemed natural to apply similar techniques to abstracts of technical articles as well. Abstracts, like titles, are intended to be concise statements of the information in a document, but by virtue of their greater length should provide more potential index terms.

The syntactic recognition problem with abstracts, however, is much more difficult than with titles. The latter often consist only of noun and prepositional phrases and rarely post-modifying participles or relative clauses. Abstracts, on the other hand, potentially exhibit the full range of syntactic constructions of formal written English, except for interrogative and exclamatory sentences, which are excluded by precepts of style. For this reason, the fairly simple procedures of the Title Analyzer Program were inadequate for dealing with abstracts with any reasonable degree of accuracy. While our initial efforts were directed (as in the Title Analyzer) toward the identification of nouns and their modifiers, the result has been a program written in COMIT5 yielding a nearly complete parsing of the "surface" syntactic structure of each sentence. The overall sequence of operations in the program is shown in Fig. 1.

In the description of the program which follows we will emphasize those features of the design imposed by the necessity for economical performance in a projected mechanized indexing system.

The program accepts cards in a format which is

---

*Present address (R. E. Wall): Department of Linguistics, Indiana University, Bloomington, Ind.
**N**-type GaAs doped with Te to $3 \times 10^{17}$ cm$^{-3}$ and $9.6 \times 10^{17}$ cm$^{-3}$ have absorption coefficients of 20 cm$^{-1}$ and 10 cm$^{-1}$, respectively, at 1.475 eV and 77$^\circ$ K.

**Printed input text**

*N-TYPE* *GA*AS DOPED WITH *TE TO *( **F **) AND *( **F **) HAVE ABSORPTION COEFFICIENTS OF *( **F **) AND *( **F **) , RESPECTIVELY , AT 1.475 E*V AND 77 DEGREES *K .

**Keypunched input text**

Figure 2. Sample input sentence.

**DICTIONARY**

The function of a dictionary in a syntactic recognition program is to assign each word* of the input text to one or more word classes (traditionally such

---

*In this discussion of the dictionary we are referring to words as tokens or separate occurrences, not as types or the total of different forms.
AN ECONOMICAL PROGRAM FOR LIMITED PARSING OF ENGLISH

categories as noun, verb, adjective, etc.). The usual approach is to use a “complete” dictionary, which contains (ideally) every word form in the language. Our program uses a “computational” dictionary of the type described by Klein and Simmons which makes word-class assignments on the basis of orthographic features. The current dictionary consists of three lists which contain about 1,000 entries in all:

1. Common function words—prepositions, articles, conjunctions, etc.
2. Word endings—-ing, -tion, -ed, -ous, etc.
3. Exceptions to the word ending rules—thing, feed, mention, etc.

One requirement of the program was that it should be suitable for use in a mechanized indexing system. A complete dictionary operating on technical text would require an addition each time a new word was encountered. Such additions are not compatible with economic automatic processing. We thus chose to use a computational dictionary designed to encode correctly the relatively few types which account for the large proportion of tokens in running text.

Words which are not classified by the computational dictionary are arbitrarily assigned to the noun/verb category. This choice is again influenced by potential use of the program in a mechanized indexing system. The hypothesis is that the importance of nominal constructions in selection of index unit candidates places emphasis on the bracketing of all noun phrases. The grammatical algorithm is designed to deal with the noun/verb ambiguity.

One advantage of our computational dictionary is that it is small enough to be contained in the core storage of an IBM 7094 along with the grammar rules and program instructions. This allows for a binary search through the dictionary lists for every word as it occurs in text order. A full dictionary, on the other hand, would have to be stored on magnetic tape (in which case the input text would have to be run in batches, alphabetized, looked up, and re-sorted into text order) or else stored on drums or disks, thereby sacrificing the advantage of a binary search. A further consequence of the small computational dictionary contained in core storage is that the processing of sentences can be opened. Since there is no need to handle the text in batches, any number of sentences can be run without interruption once the program has been loaded.

These conveniences, however, are paid for in two ways. One is the obvious limitation that many words will receive the arbitrary noun/verb classification because they were not found in the dictionary, and any misclassification may lead to erroneous phrase bracketings. The other disadvantage is in the lack of refinement possible in certain word classes. For example, although the suffix -tion nearly always serves to identify words as noun forms, they cannot be further subclassified by this clue as animate-inanimate, abstract-concrete, or countable-uncountable. Thus, a computational dictionary introduces error in syntactic recognition not only by incorrect word-class assignments, but also by limiting the discrimination which can be made in the grammar.

GRAMMAR

The grammar gives rules for the allowed combination of word classes into phrases and clauses. Nine types of phrases—nominal, pronominal, adjectival, past participle, present participle, prepositional, verbal, infinitive, and adverbial—and eight kinds of subordinate and relative clauses are recognized. The kind of clause is dependent on the clause introducer and on the alternative structural patterns predicted by that introducer. For example, if a clause is introduced by which, the grammar expects that a verb will be found but that a subject is optional. If a verb is not found, the algorithm will search for a re-bracketing to fulfill the requirement for a verb. The output is a labeled bracketing of these phrases and clauses together with the syntactic word class for each word. An example of the output is shown in Fig. 3.

Each phrase is enclosed in parentheses and is followed immediately by an identifying label (NOUP = noun phrase, PREP = prepositional phrase, etc.). A hyphen separates the phrase label from a list of the word classes assigned to each word in the phrase. These classes are based on those of Kuno and Oettinger with many modifications. Although a complete list of the word classes and their defining characteristics would be too long to include here, it would perhaps be helpful to give a few of those appearing in Fig. 3.

†A reverse-alphabetized word list was most helpful in discovering word endings and exceptions.
Figure 3. Sample of output from syntactic analysis program.

The beginning and end of the relative clause are marked by the symbols ***BC 1 and ***EC 1.

Such an analysis falls short of a "complete" phrase-structure parsing in two ways. First, the labelings of fine structures within phrases are suppressed. For example, in the sentence shown in Fig. 3, the noun phrase an arithmetic unit is not overtly labeled as such but is included in the prepositional phrase. Likewise, the finer structure of this noun phrase itself (arithmetic unit = NP, etc.) is not given explicitly.

Using phrase-structure tree diagrams we might illustrate the difference as in Fig. 4. The tree diagram at the left represents the phrase marker as it might appear for this 4-word prepositional phrase in a phrase-structure parsing; the tree at the right shows the same phrase as it would be analyzed by our grammar.

The second difference is the failure to mark dependencies between phrases and clauses. For example, in the output in Fig. 3, the point of attachment of the prepositional phrase for an arithmetic unit is not specified. The clues for joining such modifiers to the proper head seem in many cases to be purely semantic ones, and the problem is always troublesome in any parsing scheme. Jane Robinson cites the example9 I saw the man with the telescope in the park, which can have several different readings, depending on the words which the prepositional phrases are understood to modify. We do not yet know whether the simple expedient of joining post-modifiers to the nearest allowable preceding structure can be improved upon with the aid of syntactic information alone, nor do we know how much of this interphrase structure will be necessary in order to do the job of indexing. In any case, delimiting the phrase boundaries as we have done is a prerequisite to any attempt to specify these dependencies algorithmically.

The current implementation of our program does not incorporate an explicit, separable grammar. However, a formal description of the grammar in a context sensitive phrase-structure notation has been written to provide documentation.
ALGORITHM

The algorithm is a set of procedures for assigning an allowable syntactic structure to an input sentence according to the rules set forth in the grammar. A serious problem in implementing parsing algorithms has always been that the processing time tends to increase exponentially with the number of words in the sentence being analyzed (because as sentence length increases, so in general does the number of combinatorial alternatives which must be considered). Consequently, the practical upper limit on sentence length for the best existing programs has been about 40 words, and for most programs it has been much below that. Longer sentences are not at all rare, however, particularly in technical writing, and any parsing system which is intended to be a practical component of an indexing system should be able to handle them. We therefore attempted to design this parsing algorithm so that the total processing time would be directly proportional to sentence length.

The general sequence of operations is as follows. After dictionary lookup is complete, all phrase boundaries are tentatively identified in one left-to-right pass through the sentence. On a second left-to-right pass, clause boundaries are established, and tests for well-formedness in each clause are performed. Nested clauses are handled by a pushdown storage mechanism. Whenever an ill-formed condition is recognized, the algorithm initiates an ordered search for alternatives pertinent to that condition (different word classes or different phrase boundaries) and will choose the first alternative which resolves that condition. This strategy, together with the restriction that no set of alternatives can be tried more than once during the analysis of a sentence, avoids the repetitive tracing of substructures which have already been recognized as well-formed. Thus, even worst-case analysis times will vary linearly (or nearly so) with sentence length. A final series of passes through the sentence serves to link phrases joined by coordinating conjunctions and performs a few minor revisions before output.

We can illustrate one of the parts of the algorithm, the clause well-formedness testing, by using the example of the sentence in Fig. 3. During dictionary lookup, both paper and presents have received the arbitrary noun/verb coding, but the latter word, because of its -s suffix, has been coded as plural noun and third-person singular verb.

After the first pass, the phrases have been bracketed...
The noun homograph of presents has been tried first and has led to the incorrect bracketing shown. On the second pass the lack of a verb for the independent clause is noted (the relative clause is well-formed), and the algorithm then examines the first noun phrase, beginning at the right-most word, for a verb homograph. Presents is found to have such a homograph, so the subject is redefined as the remaining noun phrase, This paper, and the test for number agreement between subject and verb is made. Since this alternative produces a well-formed clause, the final bracketing is:

\[
\begin{align*}
\text{This paper} & \quad \text{presents} \\
\text{ATES NOUS NOUP} & \quad \text{VZZS NOUP} \\
\text{VZZP VZZS} & \quad \text{NOUP ATBS}
\end{align*}
\]

Had this alternative not been allowable, the other words in the phrase would have been examined for verb homographs. If none was found, the phrase would have been restored to its original bracketing and the next noun or prepositional phrase to the right in the same clause would have been broken apart and examined similarly. This procedure would have continued until either a verb and subject had been established or else all the relevant possibilities had been exhausted. However, when all the clauses of a sentence have been made well-formed, no more alternatives are tried. Thus, the algorithm arrives at only one syntactic structure for each sentence, in contrast with the multiple parsings generated by a program such as the Harvard Syntactic Analyzer.

While multiple analyses are clearly useful in linguistic research for exposing syntactic ambiguities, in a practical application such as mechanized indexing they are an embarrassment of riches. Having all structural descriptions for each sentence would be of little use since at the present time there is no method for deciding which analyses are also semantically acceptable and, further, which is the one “correct” reading intended by the author. Perhaps one could hope to select instead the “syntactically most probably” parsing if adequate statistics of English grammatical structures were available. Since they are not, we have ordered the search for alternatives according to what seem to us, intuitively, to be the most frequently occurring structures.

A useful by-product of the algorithm arises from the fact that all the phrases of a sentence are tentatively recognized in the first pass. Should the analysis of any sentence be terminated because of excessive time, or if no grammatically acceptable parsing for some clause can be found, many substrings of the sentence will still be correctly identified. Thus, in cases of noncatastrophic failure, we are able to get partial results and go on to the next sentence.

 Provision is also made for handling parenthetic expressions (the clause well-formedness tests are omitted) and clauses separated by semicolons (treated as separate sentences). Sentences up to 100 words in length can be analyzed. However, some very long sentences, depending on the particular structures they contain, will occasionally require more COMIT “workspace” than is currently available. In such cases, the program writes out the results of the dictionary lookup routine on a tape which can later be used as input to the syntactic analysis portion.

**PROGRAM TESTING**

The program was coded in COMIT because of the ease it provides in the design and updating of experimental models. Initial debugging and testing were carried out on a sample of 70 consecutive sentences taken from abstracts in the *IBM Journal of Research and Development*. This text, which we will refer to as IBM #1, was chosen because it contained a fairly wide range of technical subject matter. The results were used to make further refinements to the grammar, and then the program was tested on 5 more texts, each containing 70 sentences, from randomly selected abstracts. One text was taken from another issue of the *IBM Journal* (IBM #2) and the others from the fields of chemistry, physics, acoustics, and, for comparison with the technical material, literary criticism. The accuracy with which phrases were identified is indicated

---

*The method of counting phrases is in accord with our earlier remarks about “complete” parsing. Thus, the object in a prepositional phrase does not count as an additional nominal or pronominal phrase.*
AN ECONOMICAL PROGRAM FOR LIMITED PARSING OF ENGLISH

Table 1. Accuracy of Identifying Phrases.

<table>
<thead>
<tr>
<th>Text</th>
<th>No. of Words</th>
<th>NP</th>
<th>PP</th>
<th>Pn P</th>
<th>Inf P</th>
<th>VP</th>
<th>PMA</th>
<th>PMR</th>
<th>PMP</th>
<th>Av P</th>
<th>Totals</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM #1</td>
<td>1593</td>
<td>154/170</td>
<td>209/222</td>
<td>6/6</td>
<td>10/11</td>
<td>89/95</td>
<td>10/15</td>
<td>10/11</td>
<td>17/17</td>
<td>11/15</td>
<td>516/562</td>
<td>92</td>
</tr>
<tr>
<td>IBM #2</td>
<td>1867</td>
<td>170/182</td>
<td>248/267</td>
<td>11/12</td>
<td>12/13</td>
<td>100/110</td>
<td>12/19</td>
<td>6/7</td>
<td>20/20</td>
<td>13/19</td>
<td>597/649</td>
<td>92</td>
</tr>
<tr>
<td>Chemistry</td>
<td>1716</td>
<td>160/174</td>
<td>214/241</td>
<td>12/12</td>
<td>16/16</td>
<td>106/114</td>
<td>16/20</td>
<td>14/15</td>
<td>14/15</td>
<td>40/40</td>
<td>570/614</td>
<td>91</td>
</tr>
<tr>
<td>Acoustics</td>
<td>2192</td>
<td>193/216</td>
<td>249/283</td>
<td>5/5</td>
<td>18/21</td>
<td>117/123</td>
<td>10/11</td>
<td>8/11</td>
<td>22/23</td>
<td>14/15</td>
<td>401/465</td>
<td>93</td>
</tr>
<tr>
<td>Literary criticism</td>
<td>2192</td>
<td>193/216</td>
<td>249/283</td>
<td>5/5</td>
<td>18/21</td>
<td>117/123</td>
<td>10/11</td>
<td>8/11</td>
<td>22/23</td>
<td>14/15</td>
<td>401/465</td>
<td>93</td>
</tr>
<tr>
<td>Total</td>
<td>10878</td>
<td>1004/1092</td>
<td>1393/1526</td>
<td>99/100</td>
<td>97/102</td>
<td>657/721</td>
<td>95/127</td>
<td>53/63</td>
<td>96/102</td>
<td>113/128</td>
<td>3607/3961</td>
<td>92%</td>
</tr>
<tr>
<td>Technical material only</td>
<td>8686</td>
<td>811/876</td>
<td>1144/1243</td>
<td>45/46</td>
<td>70/75</td>
<td>524/560</td>
<td>71/92</td>
<td>49/57</td>
<td>86/88</td>
<td>76/94</td>
<td>2876/3121</td>
<td>93%</td>
</tr>
</tbody>
</table>

NP—noun phrase
PP—prepositional phrase
Pn P—pronoun phrase
Inf P—infinitive phrase
VP—verb phrase
PMA—post-modifying adjective
PMR—post-modifying present participle
PMP—post-modifying past participle
Av P—adverbial phrase

*Total number of words includes those contained in parenthetic expressions. This accounts for the discrepancies between the totals and those given for the acoustic text in Table 1.

A further test using 1,000 sentences taken from Nuclear Science Abstracts gave similar results.

It was of interest to compare the performance of our program with that of one of the most complete parsing programs available, the Harvard Syntactic Analyzer (HSA). We obtained this program from SHARE, modified it to produce only one analysis (i.e., the first found) for each sentence, and tested it on our first text, IBM #1. After homograph codes were supplied for words not found in the HSA dictionary, the program produced an analysis for 59 of the 70 sentences. Seven were rejected as ungrammatical, and four had not been analyzed after at least 5 minutes of running time on each by the SYNTAX subroutine. One sentence was allowed to run for 17 minutes without success. Table 2 shows the comparison of our program with the HSA in identifying phrases in the 59 sentences which were analyzed by both.

Table 2. Comparison with Modified Harvard Syntactic Analyzer (59 Sentences from IBM #1; 1244 Total Words).

<table>
<thead>
<tr>
<th></th>
<th>NP</th>
<th>PP</th>
<th>Pn P</th>
<th>Inf P</th>
<th>VP</th>
<th>PMA</th>
<th>PMR</th>
<th>PMP</th>
<th>Av P</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>91%</td>
<td>93%</td>
<td>100%</td>
<td>90%</td>
<td>92%</td>
<td>78%</td>
<td>88%</td>
<td>100%</td>
<td>78%</td>
<td>92%</td>
</tr>
<tr>
<td></td>
<td>83%</td>
<td>87%</td>
<td>100%</td>
<td>90%</td>
<td>92%</td>
<td>100%</td>
<td>63%</td>
<td>42%</td>
<td>67%</td>
<td>85%</td>
</tr>
</tbody>
</table>

PROCESSING TIME

The core clock was used to measure the time of each phase of our program for every sentence analyzed. A phase kindly supplied by Mr. K. L. Deckert of the IBM Systems Development Laboratory, San Jose, plotted the times against sentence length on a CALCOMP plotter and calculated the slopes of the resulting lines by the method of least squares. The summarized results appear in Table 3.

All the data appeared to be well represented by straight lines except for the second pass (testing for clause well-formedness and trying alternatives), which as expected displayed considerable scattering. The times for sentences containing parenthetic

Table 3. Average Processing Times for Each Phase of Program.

<table>
<thead>
<tr>
<th>Phase</th>
<th>sec/Time/word</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Input and dictionary lookup</td>
<td>0.072</td>
</tr>
<tr>
<td>2. Bracketing of phrases</td>
<td>0.047</td>
</tr>
<tr>
<td>3. Testing clause well-formedness and trying alternatives</td>
<td>0.019</td>
</tr>
<tr>
<td>4. Rebracketing coordinated structures and other minor corrections</td>
<td>0.024</td>
</tr>
<tr>
<td>5. Output</td>
<td>0.017</td>
</tr>
<tr>
<td>Total</td>
<td>0.179</td>
</tr>
</tbody>
</table>
expressions, because they are treated in a special way, were also found to deviate markedly from the line. Nonetheless, we found no clear indication that the total time might be increasing exponentially with length for sentences of the order of 50 to 75 words.

The total time for our program to analyze the 59 sentences of IBM #1 (Table 2) was about 4.5 minutes plus 4.2 minutes for compilation (run on the IBM 7094 but adjusted to the IBM 7090 time). The time for the modified Harvard Syntactic Analyzer was about 30 minutes (not measured precisely), which does not include the time for dictionary lookup and updating. We have recently been informed by Dr. Kuno that the running time for the Harvard program has now been reduced substantially. However, this version is not yet available to us for testing.

**DICTIONARY CODING**

Since the computational dictionary is a fundamental part of our program, we were concerned with its ability to assign words to word classes compared to the assignments made by a complete dictionary. Klein and Simmons reported that their computational dictionary could correctly assign unique word-class codes to about 90 percent of the words in their sample texts. However, this figure measures the results of two operations: first, assigning each word all its possible word-class codes and, second, eliminating ambiguous codes by means of the context. Our dictionary performs only the first of these functions, while word-class ambiguities are resolved in the syntactic recognition routine. Therefore, we counted as “correct” those words which were identified by the computational dictionary or which were not found and were indeed noun/verb ambiguities. Table 4 gives the results for the acoustics text. Words coded arbitrarily received only the noun/verb classification; the fraction of these marked “incorrect” should have been assigned to either noun only or verb only, or else should have had additional codes attached as well. Similar data were collected for all six experimental texts with an earlier version of the dictionary. The percentage of correct coding was somewhat lower (86 percent), but it was nearly the same for each of the texts.

In order to determine the extent to which parsing errors arose from inadequacies in the computational dictionary, we reran two of the texts with corrections to dictionary coding supplied by hand. The overall accuracy in identifying phrases increased from 91 to 93 percent for the chemistry text and from 87 to 90 percent for literary criticism. Thus, using a perfect dictionary with the present grammar and algorithm seems to improve the accuracy by about 2 to 3 percent.

**DISCUSSION**

The principal result of our work thus far has been to show that the approach to parsing, which we adopted for purely practical reasons, nonetheless succeeds as well in identifying phrases as at least one other more sophisticated routine. We were frankly surprised at this result. Because our program was to operate with many handicaps—a minimal dictionary, simple grammar, and severe time and space constraints on the whole program—we did not suppose that it would be able to perform so well.

We must emphasize that the comparison with the HSA (1963 version) should be accepted with reservations. The HSA, as previously noted, provides a more complete syntactic description of each sentence than does our program, and therefore the running times are not directly comparable. Also, the sample for comparison, only 59 sentences, is rather small. One might also argue that selecting only the HSA’s first analysis from each sentence may have produced a bias, but there seems to be no reasona-
ble alternative to this choice, and in fact there is some reason to believe that the first analysis (rather than the second, the last, etc.) has a greater probability of being the "correct" syntactic analysis than does any other.\(^11\)

We believe that two factors are chiefly responsible for the degree of success which our program has so far achieved. First, we have made some fortunate guesses about the probability of occurrence of syntactic constructions, at least for the kind of technical writing we have investigated. (Note that the accuracy for the literary criticism text was somewhat lower than for the others despite the fact that the dictionary coded about the same percentage of words correctly in this text.) The second factor is the strategy of searching out alternatives only to remedy a particular syntactic ill-formation. This technique allows most of the previously made "probable" choices to be left intact whenever an error is corrected.

Three kinds of errors were frequently made by our program: (1) incorrect bracketing of coordinated structures around and or or; (2) unresolved noun/verb ambiguity; and (3) incorrect assignment of words with suffix -ing, which may be adjectival (pre- and post-modifying), verbal, or gerundive. It is interesting to note that the pattern of errors made by the HSA differs considerably. A frequent mistake was also the noun/verb confusion, but it usually arose from erroneously finding relative clauses beginning with an elliptical which or that. For example, a sentence beginning [The maximum signal] [has] ... was analyzed as if it has been [The maximum] (which) [signal] [has] ... , with a plural noun occurring later in the sentence being called the predicate verb for the subject maximum.

The current version of our program occupies about 29,000 registers of an IBM 7094, thus allowing about 4,000 registers for COMIT "workspace" during analysis. Therefore, no substantial additions to the dictionary, grammar, or algorithm are possible while maintaining the program's present design on the IBM 7094. Some slight improvements in performance can undoubtedly be made at the expense of much more investigation into grammatical refinements. This would undoubtedly lead to an increase in running time and storage space required. We do not know what the limits of accuracy are for our approach, but we estimate that less than half the errors are in theory correctable (by an expanded grammar); the remaining are genuine syntactic ambiguities which presumably can only be resolved by extrasyntactic information. If this estimate holds, it means that an accuracy of about 94 percent in identifying phrases is theoretically attainable.

The point of balance between cost and accuracy will depend on the particular application envisioned for the program. Despite the encouraging results thus far, we cannot claim that our program will guarantee the feasibility of a mechanized indexing system. It is clear that more will be required for automatic indexing than an identification of phrases and clauses. For example, it may be necessary to specify some interphrase dependencies, and a means for the deletion of items deemed nonsignificant will almost certainly be necessary. Also, some syntactic transformations to convert the material into a format suitable for searching (whether by machine or human) will probably be essential. Nonetheless, the prospects for using at least a limited syntactic analysis program in automatic indexing on a large scale now seem much more hopeful.

ACKNOWLEDGMENT

The authors wish to express their appreciation to Mr. J. Bennett and Miss P. Baxendale for substantial contributions to this work throughout its development. Mr. Bennett assisted materially in the programming and made several modifications to the COMIT system which greatly expedited the debugging.

REFERENCES


