The Theory and Practice of Constructing an Optimal Polyphase Sort

M. C. Er
Department of Computing Science, University of Wollongong, Wollongong, NSW 2500, Australia
B. G. T. Lowden
Department of Computer Science, University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK

The construction of both an optimal read-forward polyphase sort and a near optimal read-backward version is described. The former minimizes the merge volume, for a given distribution of strings, without the use of auxiliary data structures, and incorporates a new technique for computing the positions of dummy strings. In the case of the read-backward version, an approach is developed which achieves a merge volume nearest to the minimum, inasmuch as the theoretical optimal read-backward polyphase sort cannot be realized. A dispersion algorithm is also described which optimizes the distribution of strings so that minimum merge volume is assured.

INTRODUCTION

The concept of the polyphase sort has been associated with computing for about 20 years, and has been well described in the literature.[1] Techniques, however, for the construction of an optimal polyphase sort algorithm are still far from satisfactory. One of the basic problems is that if replacement selection is employed to generate the initial strings, then the exact number of strings formed cannot be determined, in general, until completion of the presort phase. This is because the entropy of the file to be sorted is usually uncertain, even though the total number of records may be known in advance. Most of the algorithms described in the literature require that this piece of information be known in order that dummy strings may be allocated to those positions which are involved most frequently in merging, so that an optimal polyphase sort can be achieved without the need for a redistribution pass.

Shell[2] proposed an optimal read-forward polyphase sort algorithm, which not only demands a knowledge of the total number of initial strings, but also uses large P and Q vectors to guide both the dispersion and merging processes. Clearly, the space occupied by these vectors could otherwise be used as part of the heap space in replacement selection and I/O buffer space in the merge phase. Improvements were suggested by Zave[3] which eliminated the huge vectors, but instead introduced a sizeable V-matrix to guide the merge process. The simplest approach, suggested by Malcolm[4] avoids large data structures, but does not achieve an optimal polyphase sort since positions most frequently involved in merging are not taken into account.

Further, the literature is relatively silent regarding the construction of an optimal read-backward polyphase sort as compared with proposals for an optimal read-forward version.

This paper illustrates a way to construct an optimal read-forward polyphase sort using no auxiliary data structures, and also a near optimal read-backward version using a compact vector. The problems associated with the construction of an optimal dispersion algorithm are also discussed.

For a more detailed and rigorous treatment of the material presented in this paper, including programming code for all the algorithms discussed, the reader is referred to Ref. 5.

READ-FORWARD POLYPHASE SORT

The polyphase sort is organized in two distinct phases. In the first phase, which is called the *presort phase*, input data are converted into a number of relatively short sequenced *strings* called unit strings (US). Most commonly, these are generated using replacement selection. In the second or *merge phase*, strings are successively merged into longer strings until one string is left, which is the complete sorted file.

The *merge volume* is the total amount of data passed during the merge phase, expressed in terms of unit strings and can therefore be regarded as a measure of total work done during the sorting process.

A simple example should suffice to remind the reader of the basic operations involved in the polyphase sort, and also define some terminologies and symbols to be used subsequently in this paper.

Assuming there are T tape-drives available, the presort phase would distribute sorted strings onto $(T-1)$ working tapes as they were formed. A W-way merge, $(W = T-1)$, may now commence, after rewinding all the tapes and optionally replacing the input tape by a working tape. W strings, one string from each input tape, are next merged into one longer string and written to the initially vacant tape; this process is repeated until one of the input tapes is depleted. The depleted input tape and the output tape are simultaneously rewound, leaving other tapes as they are, to prepare for the next phase of merging. In subsequent merge passes, the previous depleted input tape becomes the receiving tape, and the rest serve as input tapes. Merging continues in this manner until all strings are merged into one.

The read-forward polyphase sort treats each tape as a FIFO queue; strings written first onto the tape will be read back first. The rewind time, which is that needed to rewind the longest of either the depleted or receiving

CCC-0010-4620/82/0025-0093 $04.50

tape, cannot therefore be eliminated. Figure 1 shows the status of each tape at the end of every pass for $T = 4$, assuming 13, 11 and 7 strings are distributed on tapes 1, 2 and 3 respectively during the presort phase.

Pass	Tape 1	Tape 2	Tape 3	Tape 4	Merge volume
Presort	13_1 rewind	11_1 rewind	7_1 rewind	— rewind & replace	
1	6_1	4_1	— rewind	7_3 rewind	21
2	2_1	— rewind	4_5 rewind	3_3	20
3	— rewind	2_9 rewind	2_5	1_3	18
4	1_{17} rewind	1_9	1_5	— rewind	17
5	— rewind	— rewind	— rewind	1_{31} rewind	31
					107 US

Note: Y_z means y strings of length z US.

Figure 1.

The simplest way to determine those configurations, which guarantee W-way merging can be maintained throughout the merge phase, is to work backwards from the last merge pass. Figure 2 illustrates the configurations for a generalized order of merge, such that the number of strings on $t_1, t_2, t_3, \ldots$ are presented in descending order, and the receiving tape discarded.

L	t_1	t_2	t_3	...	t_{w-2}	t_{w-1}	t_w
⋮	⋮	⋮	⋮		⋮	⋮	⋮
5	16	16	16	...	14	12	8
4	8	8	8	...	7	6	4
3	4	4	4	...	4	3	2
2	2	2	2	...	2	2	1
1	1	1	1	...	1	1	1
0	1	0	0	...	0	0	0
−1	0	1	0	...	0	0	0
−2	0	0	1	...	0	0	0
⋮	⋮	⋮	⋮		⋮	⋮	⋮
$3-w$	0	0	0	...	1	0	0
$2-w$	0	0	0	...	0	1	0
$1-w$	0	0	0	...	0	0	1

Note: For $L < 0$, the numbers for tapes are artificial values

Figure 2.

The above configurations are called *ideal distributions*, each level is called a *perfect level* and the number of strings on each tape, at each perfect level, is called an *ideal number*.

Let F_t^L denote the ideal number on tape t at perfect level L. The properties and relationships between ideal numbers are listed in the Appendix, (A.1) to (A.9), and analysis reveals they are based on a W-generalized Fibonacci series.

DUMMY STRINGS AND MERGE NUMBERS

In practice, the number of strings generated during the presort phase rarely turns out to be ideal. The standard approach, therefore, is to bring the total number of strings, on each tape, up to the next perfect level using dummy strings. Since the latter can be manipulated by an internal book-keeping operation, the cost of processing these dummy strings is negligible compared with real string processing. *Minimum* merge volume would therefore be achieved if dummy strings were allocated to those positions involved most frequently in the merge process.

Define *merge number* to be the number of times a string initially distributed onto a tape, is involved subsequently in merging during the merge phase.

Figure 3 shows *merge number* for $W = 3$ and perfect level L in the range 0 to 5.

	$L=0$	1	2	3	4	5
Position	$t=1\ 2\ 3$	1 2 3	1 2 3	1 2 3	1 2 3	1 2 3
0	1 0 0	1 1 1	2 2 2	3 3 3	4 4 4	5 5 5
1			1 1	2 2 2	3 3 3	4 4 4
2				2 2	3 3 3	4 4 4
3				1	2 2 2	3 3 3
4					3 3	4 4 4
5					2 2	3 3 3
6					2	3 3 3
7						4 4
8						3 3
9						3 3
10						2 2
11						3
12						2

Figure 3.

From Fig. 3, it may be seen that positions with largest merge numbers are scattered along the tapes and so the simple idea of placing dummy strings at these positions turns out, in practice, to be a non-trivial task.

In order to construct an optimal polyphase sort it is necessary to resolve the following problems.

(1) How to compute merge numbers at a given perfect level as shown in Fig. 3.
(2) How to allocate real strings at those positions with smallest merge numbers on a tape.
(3) How to initially distribute real strings onto W tapes taking all the positions with smallest merge numbers into account.

Problems (2) and (3) will be tackled in the next two sections, and we now turn our attention to a discussion of the first.

A close study of Fig. 3 indicates that the patterns of merge numbers for tapes 2 to W are precisely a duplication of tape 1 with fewer strings, and so the problem of generating merge numbers for all W tapes reduces to the task of generating merge numbers for tape 1. As has been pointed out, it is possible to derive merge numbers for a given perfect level by working backwards from perfect level 1, however, this can be a lengthy process and we now describe a method of generating merge numbers for any perfect level without the need to involve merge numbers at lower perfect levels.

Figure 4 illustrates the merging pattern for tape 1, given an initial distribution of 13, 11, 7, a binary '1' indicating one involvement, in a merge pass, of the string occupying the position in question.

Thus in the first pass strings 0–6 are merged to form

	Pass					
Position (p)	1	2	3	4	5	Merge number
0	1	1	1	1	1	5
1	1	1	1	0	1	4
2	1	1	0	1	1	4
3	1	1	0	0	1	3
4	1	0	1	1	1	4
5	1	0	1	0	1	3
6	1	0	0	1	1	3
7	0	1	1	1	1	4
8	0	1	1	0	1	3
9	0	1	0	1	1	3
10	0	1	0	0	1	2
11	0	0	1	1	1	3
12	0	0	1	0	1	2

Figure 4.

part of the first seven strings on the receiving tape. On the second pass strings originally occupying position 0–3 together with strings 7–10 are merged together and so on. In the final pass all strings are involved in the merge process to form a single string—the sorted file. The merge number for each string position is, therefore, simply the sum of '1' bits over all merge passes.

Consider now Fig. 5 which shows the result of taking the ones complement of the bit pattern of Fig. 4 and assigning a 'weight' to each bit.

	Weight					
Position (p)	7	4	2	1	1	Merge number
0	0	0	0	0	0	5
1	0	0	0	1	0	4
2	0	0	1	0	0	4
3	0	0	1	1	0	3
4	0	1	0	0	0	4
5	0	1	0	1	0	3
6	0	1	1	0	0	3
7	1	0	0	0	0	4
8	1	0	0	1	0	3
9	1	0	1	0	0	3
10	1	0	1	1	0	2
11	1	1	0	0	0	3
12	1	1	0	1	0	2

Figure 5.

The weighting function may be seen to be precisely F_1^L. Let BP_i be the ith bit from the right most digit of the bit-pattern (BP). Further, assume that $Q(BP)$ be the number of 1's in the BP. Then the bit-patterns shown in Fig. 5 have the following duality property. They can be interpreted both as positions (P) and also as merge numbers (m) at the positions concerned. This duality can be expressed formally as Eqns (1) and (2).

$$p = \sum_{i=0}^{L-1} BP_i * F_1^i \tag{1}$$

$$m = L - Q(BP) \tag{2}$$

where

$$Q(BP) = \sum_{i=1}^{L-1} BP_i \tag{3}$$

Further, the bit-patterns in Fig. 5 turn out to satisfy the Generalized Zeckendorf Theorem which is described as follows.

Generalized Zeckendorf theorem

Every positive integer I has one and only one unique representation in a W-generalized Fibonacci number system, such that

(i) $I = \sum_{i=0}^{\infty} BP_i * Fib_w^i$

and

(ii) The unique representation using minimum numbers of 1's. Where Fib_w^i is the ith number of w—generalized Fibonacci series.

The construction of a successor function based on the Generalized Fibonacci number system subject to the constraints of the Generalized Zeckendorf Theorem is described in Ref. 5.

AN OPTIMAL READ FORWARD POLYPHASE SORT

After the presort phase is complete, the number of real strings S_t distributed on tape t is known before the merge phase starts. The number of dummy strings D_t needed for tape t is, therefore, the difference between the ideal number F_t^L and S_t (see next section for a discussion of determining the optimal perfect level).

The position of dummy strings should be computed in such a way that the positions with largest merge number are allocated to first, then the positions with second largest merge number and so on until dummy strings are depleted. Note that it is unnecessary to compute the exact position of every dummy string with merge number greater than m, where m is the smallest merge number occupied by dummy string(s). We merely subtract the number of dummy strings for each merge number greater than m from D_t. Only when allocating dummy strings for each merge number m do we need to keep track of the exact positions in order to determine the *boundary* between dummy strings and real strings with merge number m. Once the boundary is determined, the merge phase can commence immediately. Also, during the merge pass, it is unnecessary to remember all the dummy strings' positions since they can easily be computed once the boundary is known.

Two questions arise from the foregoing discussions:

(1) How to successively enumerate the positions of merge number m so that the boundary between real and dummy strings may be determined.
(2) How to distinguish the positions of dummy strings from those of real strings using the boundary value.

From Fig. 6, which is a reconstruction of Fig. 5, we see that dummy strings should cover those positions with the smallest value of $Q(BP)$ (i.e. position 0) followed by positions with the next smallest value of $Q(BP)$ (i.e. positions 1, 2, 4, 7) and so on until all dummy strings are depleted. We therefore consider how to compute the number of each type of $Q(BP)$ and how to successively enumerate the position occupied by the members in each column.

Define each level uniquely defined by a perfect level and a merge number as a *sub-level* (i.e. by tabulating the merge numbers at various perfect levels). Let ${}^mN_t^L$ denote

	Q(BP)			
Position	0	1	2	3
0	00000			
1		00010		
2		00100		
3			00110	
4		01000		
5			01010	
6			01100	
7		10000		
8			10010	
9			10100	
10				10110
11			11000	
12				11010

Figure 6.

the number of strings with the type of merge number m on tape t at perfect level L.

Further, let ${}^{q}A_t^L$ denote the number of bit-patterns of q 1's for tape t at perfect level L. Then ${}^{q}A_t^L$ is related to ${}^{m}N_t^L$ formally as shown in Eqn (4).

$$ {}^{L-m}A_t^L = {}^{m}N_t^L \quad \text{for } L \geq 1 \tag{4}$$

Therefore, to calculate ${}^{q}A_t^L$ is equivalent to evaluate ${}^{L-q}N_t^L$. Hence, the relationships of ${}^{m}N_t^L$ can be extended to ${}^{q}A_t^L$.

From Fig. 7, it should be clear that the bit-pattern of minimum values for each Q(BP) is the right-justification of all 1's in the pattern subject to the constraint of the Generalized Zeckendorf Theorem such that the complete groups of two 1's appear right most as shown in Fig. 7. With this approach, the bit-patterns of minimum values may be generalized to any order of merge.

In order to compute the boundary between the dummy strings and real strings with Q 1's in the bit-patterns it is necessary, after subtracting ${}^{q}A_t^L$ for $q = 0, 1, 2, \ldots (Q-1)$ from D_t, to enumerate the bit-patterns with Q 1's, starting from the minimum value, until the dummy strings are depleted.

Q(BP)	Bit-patterns for minimum values
0	00
1	010
2	0110
3	010110
4	0110110
5	010110110
6	0110110110
7	010110110110
⋮	⋮

Figure 7.

Once the bit-pattern of the boundary is established, whether a position should be occupied by a real string or a dummy string can be decided fairly easily. A position counter is set aside to record the *relative* logical position of the strings (real or dummy) being merged with respect to the first logical strings at the start of each merge pass on all W tapes. Since the position counter has the duality property at any instant, it may be used together with the boundary of a tape, to determine whether the position in question should be occupied by a real string or a dummy string as shown in Fig. 8.

test Q(position counter's BP) < Q(boundary BP)
  then dummy string
or test Q(position counter's BP) > Q(boundary BP)
  then real string
  or test (position counter's BP) ≤ (boundary BP)
    then dummy string
    or real string

Figure 8.

Note that the last test in Fig. 8 evaluates both bit-patterns in the binary number system. It is valid because both Fibonacci and Binary number systems preserve the same monotonicity.

Apart from the position counter, a dummy string counter is set aside for each tape, during the merge phase, to keep track of the number of dummy strings on the tape concerned.

Also at the start of each successive merge pass, revised boundaries may be computed for those tapes incorporating dummy strings.

DISPERSION PROBLEMS

Now, we consider how initially to distribute real strings onto W tapes so that the distributions support the optimal read-forward polyphase sort algorithm discussed in the previous section. In other words, distribution should be done in such a way that it minimizes the merge volume.

Consider the merge volume when some of the positions are occupied by dummy strings. Let S be the total number of real strings formed in the presort phase, such that $F^{i-1} < S < F^i$. If dummy strings are used to bring the total numbers of strings up to a perfect level L, such that $L \geq i$ and dummy strings occupy only those positions with merge numbers $\geq m$ then the merge volume of S real strings starting from perfect level L, V_S^L, can be expressed as Eqn (5).

$$V_S^L = \sum_{j=\min M_1^L}^{m-1} j * {}^{j}N^L + m * \left(S - \sum_{j=\min M_1^L}^{m-1} {}^{j}N^L \right) \tag{5}$$

where ${}_{\min}M_1^L$ is the minimum merge number of tape 1 at perfect level L.

The diagram of $\ln(V_S^L)$ *vs* $\ln(S)$ for $W = 3$ is shown in Fig. 9, where V_S^L denotes the merge volume of S real strings at perfect level L.

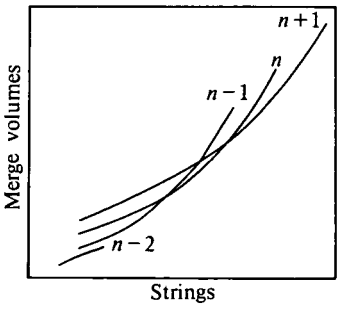

Figure 9.

Define the *optimal perfect level* to be the perfect level which minimizes the merge volume for a given S.

Further, define the *critical point* to be the point where adding a real string would result in a change of perfect level from L to $L + 1$.

From Fig. 9, it is clear that the critical points are precisely the intersection points, of two curves, for perfect levels L and $L + 1$, for $L > 0$, or the upper end point of the curve for perfect level L.

In order to achieve minimum merge volume, it is necessary to distribute strings such that the distribution follows the curve for perfect level L up to the critical point when the direction should be changed to follow the curve for perfect level $L + 1$.

It should be observed that each portion of the curves which minimizes the merge volume are made up of sub-levels (as defined in the previous section) and critical points. Tabulation of the sub-levels and critical points for $W = 3$, is shown in Fig. 10.

Dispersion level	L	t_1	t_2	t_3	Total
1	1	1	1	1	3
2	2	2	2	1	5
3	3	3	2	1	6
4	3	4	3	2	9
5	4	6	5	3	14
6	4	7	6	4	17
7	5	12	10	6	28
8	5	13	11	7	31
9	6	18	14	8	40
10	6	23	19	12	54
11	7	37	30	18	85
12	7	41	34	20	95
13	8	52	40	22	114
14	8	73	60	37	170
15	8	74	61	37	172
16	9	112	89	52	253
17	9	122	98	59	279
18	10	151	114	62	327

Figure 10.

Define each row shown in Fig. 10 as a *dispersion level* (DL), and the numbers for each tape at a dispersion level as *dispersion numbers*. Further, define *optimal dispersion* to be the dispersion of real strings such that the minimum merge volume can be achieved.

It may now be seen that, in order to achieve optimal dispersion, the dispersion algorithm should be guided by *dispersion* levels rather than *perfect* level. Each row of dispersion numbers can be read in when they are needed. The algorithm for computing the dispersion numbers is described in Ref. 5. The question still unanswered is whether or not such an optimal dispersion exists for any order of merge.

Define *anomalous dispersion* to be the situation of at least one of the tapes at perfect level $(L + 1)$ having less strings than the corresponding tape at perfect level L at the critical point concerned. Empirical results[5] suggest that anomalous dispersion rarely happens; and when it does, it incurs not more than 0.5% increase over the theoretical minimum merge volume.

READ-BACKWARD POLYPHASE SORT

The objective of developing the read-backward polyphase sort was to eliminate tape rewind time. This is achieved by basing its operation on the principle of a LIFO stack rather than the FIFO queue of the read-forward version.

The LIFO stack operates as follows: Strings are always written forwards and read backwards. This requires extra hardware to provide a read-backwards facility, and also fast switching between the two modes. Strings written forwards in ascending order would therefore appear in descending order when read backwards and *vice versa*. In order to maintain a $T = (W - 1)$-way merge throughout, the strings being merged, at any given moment, must be ordered in the same direction. This may be achieved by arranging that the strings are initially distributed in such a way that adjacent strings are ordered in opposite directions. Figure 11 illustrates the process for $W = 3$ and $L = 5$.

Pass	t_1	t_2	t_3	t_4	Merge column
Presort	D1 ←				
	A1				
	D1	D1 ←			
	A1	A1			Front of tapes
	D1	D1			
	A1	A1			
	D1	D1	D1 ←		
	A1	A1	A1		
	D1	D1	D1		
	A1	A1	A1		
	D1	D1	D1		
	A1	A1	A1		
	D1	D1	D1	—	
1.				A3	
	D1			D3	
	A1			A3	
	D1	D1		D3	
	A1	A1		A3	
	D1	D1		D3	
	A1	A1	—	A3	21
2.			D5		
			A5	A3	
	D1		D5	D3	
	A1	—	A5	A3	20
3.		D9	D5		
	—	A9	A5	A3	18
4.	D17	D9	D5	—	17
5.	—	—	—	A31	31
					107 US

Figure 11.

The merge volume shown in Fig. 11 equals that shown in Fig. 1 and, since both relate to the same perfect level and both have the same ideal distribution, much of the analysis carried out above, for the read-forward polyphase sort, is still applicable to the read-backward version. Nevertheless, stack operations and alternating string direction add an extra degree of complexity.

Figure 12 illustrates the distribution of merge numbers within the read-backwards polyphase sort for $W = 3$.

From this we observe that the merge numbers on tapes $t_2 - t_w$ are exactly a duplication of that on t_1. Also the duplication starts from the rear and is truncated at the front to make up a perfect level. Therefore, once the

Position	$L=5$			$L=4$			$L=3$			$L=2$			$L=1$			$L=0$		
	t_1	t_2	t_3	t_1	t_2	t_3	t_1	t_2	t_3	t_1	t_2	t_3	t_1	t_2	t_3	t_1	t_2	t_3
0	3D	3D	3D	2A	3D	3D	1D	2A	3D	1D	1D	2A	1D	1D	1D	1A		
1	2A	4A	4A	3D	2A	4A	2A	3D	2A	2A	2A							
2	3D	3D	5D	2A	3D	3D	3D	2A										
3	4A	2A	4A	3D	4A	2A	2A											
4	3D	3D	3D	4A	3D													
5	2A	4A	4A	3D	2A													
6	3D	5D	3D	2A														
7	4A	4A																
8	5D	3D																
9	4A	4A																
10	3D	3D																
11	4A																	
12	3D																	

Figure 12.

merge numbers on t_1 are determined, those for the other $(T-1)$ tapes may easily be derived.

Consider the example as shown in Fig. 13 for $W=3$ and $L=5$.

p	Step 1	2	3	4	5	Merge number
0	0	0	1↑	1↓	1↑	3
1	0	0	1↑	0	1↓	2
2	0	1↑	1↓	0	1↑	3
3	0	1↑	1↓	1↑	1↓	4
4	0	1↑	0	1↓	1↓	3
5	0	1↑	0	0	1↓	2
6	1↑	1↓	0	0	1↑	3
7	1↑	1↓	0	1↑	1↓	4
8	1↑	1↓	1↑	1↓	1↑	5
9	1↑	1↓	1↑	0	1↓	4
10	1↑	0	1↓	0	1↑	3
11	1↑	0	1↓	1↑	1↓	4
12	1↑	0	0	1↓	1↑	3

Note: ↑ upwards direction
↓ downwards direction

Figure 13.

In step 1, starting from the last position and moving to the front, seven (F_1^{L-1}) 1's fill positions $p=12$ to $p=6$, and 0's fill the rest.

In step 2, we split all positions determined in step 1 into two homogeneous groups. Starting from the last position in step 1, we fill four (F_1^{L-2}) 1's from $p=6$ to $p=9$ in a forwards direction; similarly, we fill four 1's from $p=2$ in backwards direction. Zeros fill the rest.

In step 3, the same principle is applied as in step 2 by splitting all positions into four homogeneous groups as determined in step 2. Starting from where we left off in step 2 (i.e. $p=2$ and $p=9$), we fill two (F_1^{L-3}) 1's for each group in opposite direction, zeros fill the rest.

In step 4, the same principle is applied again and again.

In the last step, all positions are trivially filled with 1's because all strings are involved in the last merge pass.

Finally, the sum of binary numbers as determined from step 1 to step 5 is seen to be the merge number for the corresponding position on tape 1.

The relationship between merge numbers for successive perfect levels may be formally expressed as (6)–(8), where Y_p^L denotes the merge number at perfect level L occupying position p.

$$Y_0^L = 1 \tag{6}$$

$$Y_p^L = Y_{F_1^{L-1}-p-1}^{L-1} + 1 \quad \text{for } F_1^L - F_1^{L-1} \leq p \leq F_1^L \tag{7}$$

$$Y_p^L = Y_{p+F_1^{L-1}-T}^{L-1} \quad \text{for } 0 \leqslant p < F_2^{L-1} \tag{8}$$

Let $\max p^L$ denote the position of maximum merge number on tape 1 at perfect level L. Then (9) holds.

$$\max p^L = F_1^L - \max p^{L-1} - 1 \tag{9}$$

The relationship between merge numbers at the same perfect level is not immediately apparent. However, by introducing the notion of a 'mirror', part of the bit-pattern at some positions may be viewed as the mirror images of others, thus providing some clue for analysis. The idea is illustrated in Fig. 14, which is a modification of Fig. 13 but includes the 'mirrors'.

p	Step 1	2	3	4	5	Merge number
0	0	0	1↑	1↓	1↑	3
1	0	0	1↑	0	1↓	2
2	0	1↑	1↓	0	1↑	3
3	0	1↑	1↓	1↑	1↓	4
4	0	1↑	0	1↓	1↑	3
5	0	1↑	0	0	1↓	2
6	1↑	1↓	0	0	1↑	3
7	1↑	1↓	0	1↑	1↓	4
8	1↑	1↓	1↑	1↓	1↑	5
9	1↑	1↓	1↑	0	1↓	4
10	1↑	0	1↓	0	1↑	3
11	1↑	0	1↓	1↑	1↓	4
12	1↑	0	0	1↓	1↑	3

Note: —— Mirror

Figure 14.

From Fig. 14 it may be seen that bit-patterns at $p=5$ to $p=0$ are mirror images of bit-patterns at $p=6$ to $p=11$ respectively. Similarly, bit-patterns at $p=10$ to $p=12$ are the mirror images of bit-patterns at $p=9$ to $p=7$ respectively and so on. We may therefore draw the following conclusion: the merge number of a mirror image is exactly one less than the merge number of the corresponding object. Based on this simple concept a generalized algorithm[5] for enumerating the merge numbers of the read-backward polyphase sort for any given

perfect level, may be constructed which does not depend on evaluating merge numbers at lower perfect levels.

By comparing Fig. 13 with Fig. 4, it may be seen that the read-backward polyphase sort merely rearranges the position of merge numbers without increasing or decreasing the total of each type of merge number. Hence the merge volume formulae, and certain others, developed for the read-forward case, are still appropriate for the read-backward version.

DISPERSION DIFFICULTIES

Apart from the general dispersion problems, already discussed, the read-backward polyphase sort exhibits an extra degree of difficulty resulting from the alternating nature of adjacent strings. In general, if $((L-1) \text{ rem } T) = 0$, for any $W > 1$, then the direction of all first strings at perfect level L is the same, otherwise the direction of one of the first strings (i.e. on the object tape) has odd parity at the perfect level concerned.

In reality, the directions of the first strings have to be decided as soon as the distribution of real strings commences, without prior knowledge of the total number of real strings and the eventual perfect level. Under such constraints, it is not always possible to achieve optimal dispersion. One approach is to use extra dummy strings to proceed to the next higher perfect level which coincides with the initial guess but may involve a non-optimal perfect level. A better approach is to always start the direction of the first strings with one ascending string and $(W-1)$ descending strings (assume the final sorted sequence is ascending). If the perfect level, however, demands that the direction of all first strings be the same, a dummy string can be assumed to exist in front of the relevant tape. To cater for this, whenever $((L-1) \text{ rem } T) = 0$, the total number of real strings distributed on the tape, whose first string is ascending, must be at least one less than the corresponding ideal number. Clearly, the advantage of this approach is that the merge volume is closer to the minimum merge volume than is the case of going up one level higher than the optimal.

NEAR OPTIMAL READ-BACKWARD POLYPHASE SORT

The construction of an optimal read-backward polyphase sort demands the allocation of dummy strings at those positions with the largest merge numbers. However, due to the inherent alternating direction of adjacent strings, a single dummy string (or indeed any odd number of strings) may not be inserted without causing interference to the subsequent merge pattern. An alternative approach is to insert dummy strings in pairs (or any even number). Since no adjacent merge numbers have equal value, the insertion of 'pairwise' dummy strings no longer guarantees that all positions with the largest merge numbers are filled first. Consequently, the theoretical optimal read-backward polyphase sort cannot always be achieved with the pairwise insertion approach.

Nevertheless, due to the fact that any pair of adjacent strings can be considered to comprise an object and its mirror image, the merge number of the mirror image is only one less than the merge number of the corresponding object. The pairwise approach always guarantees, therefore, that a *near* optimal read-backwards polyphase sort can be constructed and the resulting merge volume is in excess of the theoretical minimum merge volume by a value equal to the number of 'wrongly' allocated dummy strings. The additional volume passed through, is in general, far less than would result from a redistribution pass.

With the added complication of pairwise insertion of dummy strings, there would appear to be no simple principle to guide the merge phase of the read-backward polyphase sort and a compromise solution is to record those positions occupied by dummy strings using vectors.

During the presort phase, strings are distributed to W tapes in line with the principle of achieving optimal dispersion discussed earlier in this paper.

Following the presort phase, a vector of merge numbers (VMN) may be generated using the algorithm for enumerating merge numbers, since the optimal perfect level and the ideal numbers of all W tapes are known at this stage.

Further, in order to speed up the scanning process of finding the next dummy string position, it is worth building a small vector of numbers of each merge number type (VNMN).

A vector of positions of dummy strings (VPIDS), for each of W tapes which have dummy strings, can then be constructed with the aid of VMN.

If a dummy string is required to be inserted in front of the first real string then the first string position should be selected and the total number of dummy strings remaining, on that tape, reduced by one. If, however, the total number of dummy strings left on the tape is an odd number, thus reducing the value to an even number. Subsequently, pairwise selection can commence.

Once a string's position p is selected, its merge number m in VMN is changed to zero and the position p is recorded in VNMN, causing both the total number of dummy strings left on that tape, and the amount of the type of merge number m in VNMN be decremented by one. Therefore, the VNM is scanned from the last string's position to the front until the largest merge number m (at position p, say) is found as indicated by VNMN. If the adjacent position $(p-1)$ or $(p+1)$ contains a non-zero merge number, both p and $(p-1)$ or $(p+1)$ are selected as a pair. However, if position $(p-1)$ and $(p+1)$ contain zeros it is not possible to select p and $(p-1)$, or p and $(p+1)$, because $(p-1)$ and $(p+1)$ had been selected previously; however, the merge number at p is still changed to zero, and the amount of the type of merge number m in VNMN is decremented by one. Hence the search for largest merge number can always be based on the information provided by VNMN. The searching process is repeated until the number of dummy strings for the tape concerned is reduced to zero and finally, VPIDS is sorted into sequence.

The same procedure is repeated to create VPIDS for other tapes, and although VMN needs to be re-established every time, its length may be shortened by zeroing the first $(F_1^L - F_t^L)$ string positions.

Figure 15 illustrates the selection of 8 dummy strings, including the first string position, for ideal number 13 on tape 1 at perfect level 5 for $W = 3$.

Note that none of the vectors described above, exist

VMN	VNMN	VPIDS	@
3 2 3 4 3 2 3 4 5 4 3 4 3	0 2 6 4 1	-	8
0 2 3 4 3 2 3 4 5 4 3 4 3	0 2 5 4 1	0	7
0 2 3 4 3 2 3 4 5 4 3 4 0	0 2 4 4 1	0 12	6
0 2 3 4 3 2 3 0 0 4 3 4 0	0 2 4 3 0	0 12 8 7	4
0 2 3 4 3 2 3 0 0 4 0 0 0	0 2 3 2 0	0 12 8 7 11 10	2
0 2 3 4 3 2 3 0 0 0 0 0 0	0 2 3 2 0	0 12 8 7 11 10	2
0 2 0 0 3 2 3 0 0 0 0 0 0	0 2 2 1 0	0 12 8 7 11 10 3 2	0
0 2 0 0 3 2 3 0 0 0 0 0 0	0 2 2 1 0	0 2 3 7 8 10 11 12	0

Note: @ is total dummy strings left

Figure 15.

until the presort phase has been completed and so they in no way affect initial string lengths. Also the space occupied by VMN and VNMN can be released prior to the start of the merge phase. A VPIDS for the receiving tape during the merge phase is needed as well, although space relating to these vectors, may be reduced by packing several values to a computer word.

During the merge phase a dummy string is assumed, if the string position under consideration appears in VPIDS of the tape concerned, otherwise a real string is read from the tape. If all the input tapes present dummy strings at a string position, the corresponding dummy string position on the output tape is recorded on its associated VPIDS. To avoid searching VPIDS, it is possible to hold a pointer to the next dummy string position. The merge phase proceeds in read-backwards and write-forwards manner until one string is left.

ELIMINATION OF REWIND TIME

The development of the read-backward polyphase sort was aimed at eliminating tape rewind and hence saving rewind time. However, it is possible, with a read-forwards facility only, to reduce tape rewind time by overlapping tape rewinding with merging.[6] That is, when one tape is being rewound, merging continue on the remaining tapes. This is known as the tape-splitting polyphase sort. Such an approach achieves $W = T - 2$ throughout the merge phase, although minimum of four tape drives is required.

As discussed earlier, the only rewind time incurred in the read-backward polyphase sort is the rewinding of the initial input tape and final output tape. Further, there is no reason why a $(W - 1)$ way merge, such that $W = T - 1$, could not be performed whilst rewinding and replacing the initial input tape. That is, strings are distributed onto W tapes, during the presort phase. While the input tape is being rewound and replaced, a $(W - 1)$-way merge can be carried out, and the resulting strings written onto that empty tape. The approach is illustrated in Fig. 16 for 24 real strings with $T = 4$, assuming tape 4 is the initial input tape.

During the presort phase, the dispersion algorithm distributes ideal numbers of strings on tape 1 to tape $(W - 1)$ for a perfect level L of W-way merge leaving tape W empty. More precisely, dispersion is aimed at distributing F_1^L strings on tape t for $1 \leq t \leq W - 1$. (A.4) guarantees that, in pass 1 of a $(W - 1)$-way merge, F_1^{L-1} strings can be merged from $(W - 1)$ tapes and written onto tape W leaving F_{t+1}^{L-1} on tape t for $1 \leq t < W$. Close study of Fig. 12 confirms that after pass 1 of the

Pass	t_1	t_2	t_3	t_4
Presort	D1	D1	—	—
	A1	A1		
	D1	D1		
	A1	A1		
	D1	D1		
	A1	A1		
	D1	D1		
	A1	A1		
	D1	D1		
	A1	A1		
	D1	D1		
	A1			
	D1			
1	D1	D1	A2	rewind
	A1	A1	D2	and
	D1	D1	A2	replace
	A1	A1	D2	
	D1		A2	
	A1		D2	
			A2	
2	D1	—	A2	D4
	A1		D2	A4
			A2	D4
				A4
3	—	D7	A2	D4
		A7		A4
4	D13	D7	—	D4
5	—	—	A24	—

Figure 16.

$(W - 1)$-way merge, the directions of strings on all W tapes exactly coincide with the string direction pattern at perfect level $(L - 1)$. A W-way read-backward polyphase sort, therefore, can proceed from there onwards (i.e. applying the near optimal read-backward algorithm).

If, however, a full complement of strings on tape 1 to tape $(W - 1)$ are not fully attained in the presort phase, strings are merged such as to leave F_{t+1}^{L-1} strings on tape t for $1 \leq t < W$ after the first merge pass, and then the near optimal read-backward polyphase sort algorithm applied described earlier. It is still possible to merge F_1^{L-1} strings from each of $(W - 1)$ tapes; however, in view of the fact that the merge strings produced in pass 1 are $(W - 1)$ times as long as the initial strings, on average, and the corresponding merge numbers are consequently proportionally increased, then heuristically it is better to reduce the number of such strings.

Notice that, quantitatively, it might appear possible to distribute $F_t^{L-1} + F_W^{L-1}$ on tape t for $1 \leqslant t < W$; however, after merging F_W^{L-1} strings from each of $(W-1)$ tapes and writing onto tape W in pass 1, the directions of the strings will not exactly match the required direction pattern and hence in practice this would not be a viable approach.

REMARKS

In this paper, we have shown how to construct an optimal read-forward polyphase sort and a near optimal read-backward version. Moreover, the underlying principles and the dispersion problems have also been discussed. It is worth pointing out that the duality of bit-patterns, developed for the read-forward polyphase sort, is also applicable to the tape-splitting polyphase sort.

Acknowledgments

The authors are indebted to J. Washbrook of University College, London, for his constructive comments on the first draft of this paper. The referee's comments greatly improve the presentation of this paper.

REFERENCES

1. D. E. Knuth, *Sorting and Searching*. Addison-Wesley, Reading, Massachusetts (1975).
2. D. L. Shell, Optimising the polyphase sort. *Communications of the ACM* **14** (No. 15), 713–719 (1971).
3. D. A. Zave, Optimal polyphase sorting. *SIAM Journal on Computing* **6** (1977).
4. W. D. Malcolm, String distribution for the polyphase sort. *Communications of the ACM* **6**, 217–220 (1963).
5. M. C. Er, The theory and practice of constructing an optimal polyphase sort, MSc. Thesis, available from the University of Essex Library (1978).
6. R. L. McAllester, Polyphase sorting with overlapped rewind. *Communications of the ACM* **7**, 158–159 (1964).

Received December 1980

APPENDIX

$$F_1^L = \left\{\begin{matrix} 1, & t = 1 - L \\ 0, & \text{otherwise} \end{matrix}\right\} \text{ for } 1 - W \leq L \leq 0 \qquad \text{(A.1)}$$

$$F_t^1 = 1 \text{ for } 1 \leq t \leq W \qquad \text{(A.2)}$$

$$F_W^L = F_1^{L-1} \text{ for } L \geq 1 \qquad \text{(A.3)}$$

$$F_t^L = F_1^{L-1} + F_{t+1}^{L-1} \text{ for } L \geq 1 \text{ and } 1 \leq t \leq W - 1 \qquad \text{(A.4)}$$

$$F_t^L \geq F_{t+1}^L \text{ for } L \geq 0 \text{ and } 1 \leq t \leq W - 1 \qquad \text{(A.5)}$$

$$F_t^L = \sum_{i=1}^{W} F_t^{L-i} \text{ for } L \geq 1 \text{ and } 1 \leq t \leq W \qquad \text{(A.6)}$$

$$F^L = \sum_{i=1}^{W} F_i^L \text{ for } L \geq 1 - W \qquad \text{(A.7)}$$

$$F^L = \sum_{i=1}^{W} F^{L-i} \text{ for } L \geq 1 \qquad \text{(A.8)}$$

$$F_1^{L-T+t} = F_t^L - F_{t+1}^L \text{ for } L \geq 1 \text{ and } 1 \leq t \leq W - 1 \text{ where } T = 1 + W \qquad \text{(A.9)}$$