High Storage Utilisation for Single-Probe Retrieval Linear Hashing

S. F. OU AND A. L. THARP[1]

Computer Science Department, North Carolina State University, Raleigh, North Carolina 27695-8206 USA

Many information systems require expandable files. Linear hashing is a well-known technique for growing a file without requiring a complete reinsertion of the data. With the advantage of expandability comes greater retrieval costs as compared to static file methods. A recent modification of linear hashing, Linear Hashing with Separators (LHS), requires only a single access of auxiliary memory. However, this method too has limitations. It introduces secondary clustering which increases insertion costs and it has lower storage utilisation than might be desirable.

After surveying linear hashing and LHS, this paper introduces an algorithm called High Storage Utilisation for Single-Probe Retrieval (HSUSPR is read SOO-SPRAH). HSUSPR linear hashing maintains the single probe retrieval of LHS but with lower insertion costs and greater auxiliary storage utilisation. To achieve these benefits HSUSPR does use more primary memory for storing index information.

Extensive experimental results are given comparing HSUSPR with LHS. Experimental results also are provided which indicate that the performance of HSUSPR is independent of the distribution of the key data. HSUSPR is particularly useful for information systems with many insertions and/or those systems involving real-time updates.

Received October 1990, revised May 1991

1. INTRODUCTION

Over the years, several extendible hashing algorithms have been developed. Compared with traditional hashing algorithms, these algorithms have the advantage of allowing the file size to change without completely reorganising the file. The three major extendible hashing algorithms are extendible hashing,[1] dynamic hashing,[2] and linear hashing.[3] This paper focuses on linear hashing, which as can be noted from Table 1, has the advantages of controllable storage utilisation and no indexes. Linear hashing hashes a key to a page in the file. The overflow pages are handled by chaining. That is, overflow records are placed in overflow pages which are linked to the home page. This whole succession of linked pages is called a chain. For the file to expand gracefully, each chain in the file splits, appending a chain to the file. The records in the split chain (that is, the primary page and its overflow pages) are divided between the two chains. The order of splitting a chain is sequential starting from the first chain of the file. A pointer, NEXT, points to the chain to be split next. When all the chains are split, the file size has doubled.

Table 1. Characteristics of Extendible Hashing, Dynamic Hashing, and Linear Hashing

Characteristics	Extendible hashing	Dynamic hashing	Linear hashing
Index–Y/N	Yes	Yes	No
Index depth-	1	> 1	0
Expansion cost/expansion* no. of probes	2	2	6.24–9.47
Control storage utilisation–Y/N	No	No	Yes
Maximum storage utilisation	69%	69%	>= 90%
Successful retrieval no. of probes	1	1	1.05–1.57
Unsuccessful retrieval no. of probes	1	1	1.27–2.48

* Doubling of the file size.

[1] Please address all correspondence to this author.

The graceful expansion of linear hashing is due to using *splitting functions*.[3] The file starts with N chains; therefore, the hashing function produces a value between 0 and N − 1. If c is the key, $h_0(c) = c \bmod N$. The splitting functions for h_0, are $h_1, h_2, h_3, \ldots$. However, these functions must have the following property: $h_i(c) = \{0, 1, 2, \ldots, 2^i N - 1\}$.

The file begins with $h_0(c) = c \bmod 2^0 * N$. When a chain is split, the records are rehashed with $h_1(c) = c \bmod 2^1 * N$. $h_1(c)$ divides the records into the two chains. Once all the original chains in the file are split, the pointer NEXT is reset to 0 and $h_1(c)$ becomes the primary hashing function. Now when chains are split, the records are rehashed with $h_2(c)$. After each doubling of the file, the level i is increased by one. For splitting, the function for rehashing the records is always $h_{i+1}(c) = c \bmod 2^{i+1} * N$. Splitting a chain occurs when the **storage utilisation**, α, goes beyond a given limit. To retrieve a record, we try the hash function $h_i(c) = c \bmod 2^i * N$; if the address is less than the value of NEXT, then we know that the chain has been split. We then hash the key, c, with $h_{i+1}(c) = c \bmod 2^{i+1} * N$ to locate the actual address.

Each overflow page access requires one probe; hence, the longer the chain, the more probes that are needed, on the average, to retrieve a record for both successful and unsuccessful retrieval. Litwin indicated in Ref. 3 that for αs of 75% to 90%, the average successful retrieval requires between 1.05 to 1.57 probes and an unsuccessful retrieval requires from 1.27 to 2.48 probes.

Because retrievals require more than one access, variations of linear hashing to improve retrieval performance have appeared. A recent variation is *Linear Hashing with Separators* (LHS).[4] It handles overflow

records by linear probing, that is, overflow records are stored in the first vacant page closest to the home address. This process causes secondary clustering which is the occurrence of records belonging to different home addresses following the same sequence of probe addresses. Secondary clustering causes poorer insertion costs, expansion costs, total insertion costs, and an unnecessarily larger record pool size.

LHS uses separator signatures to indicate the smallest signature which is rejected from a page. A signature is an encoding of the contents of a record.[5] The separator uniquely separates the records in a page from those in the immediate overflow page. On retrieval, the separators, which are stored in primary memory, are searched so that only one probe of auxiliary memory is needed. LHS also distributes the records uniformly over all the pages by partial expansions. With partial expansions, the doubling of the file occurs after a series of partial expansions. Instead of one chain expanding into two, several chains may expand into a group of chains one greater. If a doubling occurs with two partial expansions, the first expansion increases the file size to 1.5 times the original while the second completes the doubling.

Larson himself notes two deficiencies of LHS: (1) the existence of secondary clustering, and (2) its poor performance with αs above 85%. This paper presents an algorithm which eliminates these deficiencies in LHS. Our goals were to develop an algorithm with the following properties: (1) one probe per successful retrieval; (2) at most one probe per unsuccessful retrieval; (3) computation time for the addresses to be totally independent of the file size; (4) main memory storage limited to about two bytes per page; (5) controllable and high αs – typically greater than 85%, and (6) no secondary clustering so that the algorithm will perform efficiently in insertions and expansions with a smaller record pool size.

2. HIGH STORAGE UTILISATION FOR SINGLE-PROBE RETRIEVAL LINEAR HASHING

High Storage Utilisation For Single-Probe Retrieval linear hashing (HSUSPR is *read* SOO-SPRAH)[6] combines the use of separators from LHS with the organisational scheme for overflow pages from the original linear hashing. Normally, the storage required to link the separators associated with the data pages would be excessive for storing in primary memory because of using explicit links. HSUSPR reduces this effect considerably by employing a memory management technique which uses a displacement value to *compute* the actual location of an overflow page rather than having an explicit pointer. Because the displacement value requires much less storage than the pointer, it may be possible to keep it and the separators in primary memory. Hence, only one access of auxiliary memory is needed per retrieval.

The overflow pages are organised into blocks so a small displacement field can locate the overflow pages within that block. A group of primary pages share an overflow block. The separator signature requires 8 bits and, for convenience, the displacement field is another 8 bits. Each auxiliary memory page can then be represented by a 16 bit node. With an 8 bit displacement, the overflow block can only be 256 pages. A group of primary pages must not require more than 256 overflow pages. In our simulation, limiting a group of primary pages to 128 worked fine.

Logically the size of the group should be proportional to the overflow block because the more primary pages in a group, the more overflow pages that may be required. In the simulation, the overflow block size is the same as the group size. Algorithm 1 computes the group size given the current level. There are two important issues here; first, the initial **if** statement ensures that the *divisor* is less than 128. The second **if** statement decides when the group size should increase, that is, when there are more than 31 groups. The number of groups is limited to 31, rather than 32, because a dummy group is used to indicate where the free pages begin. 32 groups can be represented by 5 bits. The more groups there are, the more bits that are needed to represent these groups; therefore, it is desirable to keep the number of groups small.

```
divisor := 0;
no := -1;
repeat
  if divisor < 128 then
    no := no+1;
  divisor := 2^(1+no);
  nogps := (N*2^Cur_level) div divisor;
  if (nogps ≤ 31) or (divisor = 128) then
    okay := true;
  else
    okay := false;
until okay;
group_size := divisor;
```

Algorithm 1. Computing the group size.

If a given overflow block is insufficient to meet the group requests, then a larger block is allocated to the group. The contents from the former block are copied to a new larger overflow block. The block size doubles whenever the overflow block is insufficient to meet the requests.

Algorithm 2 computes the group number given the page number, current level, and group size. In the first line, x increases even if the page number does not change provided that the current level increases. The actual group number is obtained by y **mod** 31.

```
x := N*2^Cur_level + page_no - 1;
y := x div group_size;
group_no := y mod 31;
```

Algorithm 2. Computing the group number for a given page and group size.

A bit pattern in the node must indicate the status of the page. Since it is beneficial to have a large separator, 8 bits are allocated for it; then the displacement field is used to indicate the page status. A value of 255 indicates the page is empty; 254 indicates the page is neither empty nor full, and 253 indicates the page is full. Therefore, the values 0 to 252 are possible for page displacements. Figure 1 shows the nodes representing primary pages 0, 1, and 2. The displacement fields of 255 mean that the pages are empty. The SIG field is the page separator.

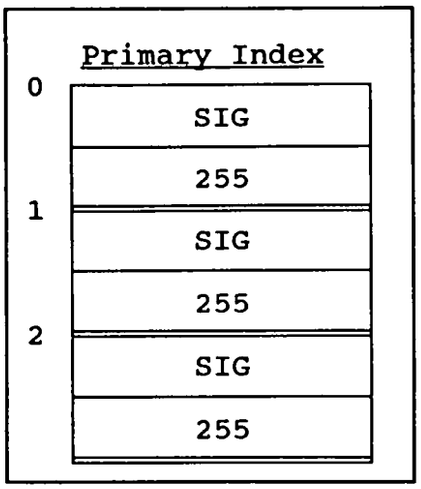


Figure 1. Primary nodes.

Group Overflow Table

Group no.	Starting page
1	16
0	20
dummy	24

Figure 2. The group overflow table.

We assign the largest key signature in a page as the separator in contrast to LHS where the smallest signature displaced from the page becomes the separator. The overflow blocks are assigned based on the sequence of requests. Therefore, a table is needed to indicate which block is allocated to what group. This table, the group overflow table, is shown in Figure 2. It has two fields; the group number and the starting page of the overflow block allocated to that group. According to Figure 2, the group 1 overflow block starts with page 16, and the group 0 overflow block starts with page 20. Also notice that subtracting 16 from 20 gives the size of the overflow block allocated to group 1. The dummy entry indicates that pages 24 onwards are free. Subtracting 20 from 24 gives the overflow block size of group 0. An advantage of the dummy entry is that the overflow block size of group 0 can be computed. The dummy entry also indicates where the next overflow block begins. Assuming that group 2 requests overflow pages, and the overflow block size is four, then the dummy entry is replaced by the group 2 entry. Another dummy entry is appended with a starting page of 28 because pages 24–27 are assigned to group 2. The next overflow block would start from page 28. Figure 3 shows the addition of group 2.

Now if group 0 returns all its overflow pages, then the group 0 entry is replaced by a dummy entry as shown in Figure 4. With the dummy entry starting from page 20 now, the overflow block size of group 1 can still be computed (20–16). The dummy entry also indicates that a hole of size four (24–20) exists starting from page 20.

Group Overflow Table

Group no.	Starting page
1	16
0	20
2	24
dummy	28

Figure 3. Group overflow table with group 2 added.

Group Overflow Table

Group no.	Starting page
1	16
dummy	20
2	24
dummy	28

Figure 4. Group overflow table with group 0 returned.

This hole will be allocated to the next request for which its size is sufficient. The dummy entry will then be replaced by the new group number.

Since not every overflow page in an overflow block may be used, a bit is used to represent an overflow page. A '0' indicates that the page is not used, and a '1' indicates that it is. Referring to Figure 2, let us assume that group 1 only uses overflow page 16, and group 0 only uses overflow page 20; then the respective page indicators are set to 1. All other page indicators are set to 0 as shown in Figure 5. Notice that only the first and fifth bits are set to '1' because the first bit represents the first page in the first overflow block which is page 16, and the fifth bit represents the fifth page from 16 which is page 20. The relative positioning of the 1s also indicates the relative positioning of the respective secondary nodes. The overflow pages are represented by secondary nodes using the same format as for the primary nodes. In the scenario in Figure 5, the secondary index should have only 2 nodes, 1 node to represent page 16 and the other to represent page 20.

Page Indicators

16	17	18	19	20	21	22	23
1	0	0	0	1	0	0	0

Figure 5. Page indicators showing that overflow pages 16 and 20 are being used.

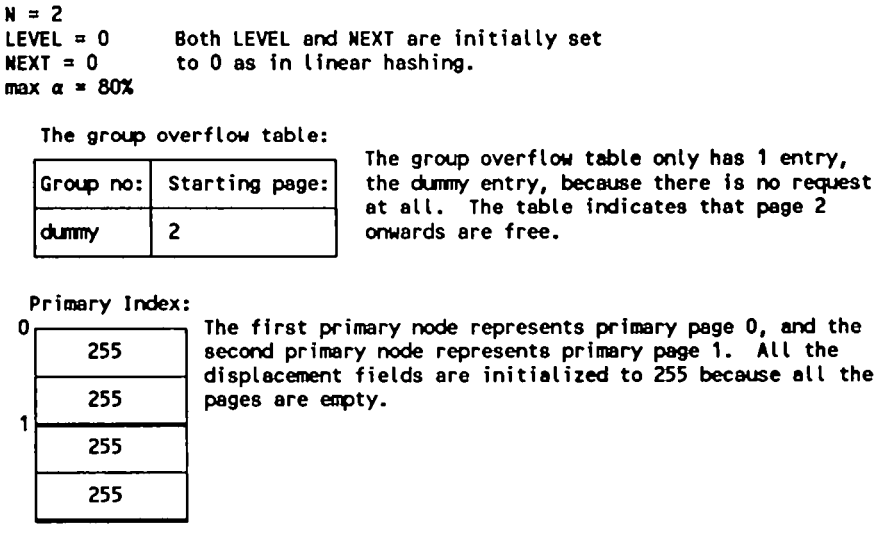
2.1 Insertion

Next we consider insertion via an example. Let us assume that the initial file size is 2 pages, the primary page can accommodate 5 records, the overflow page can hold 2 records, and the upper bound for α is 80%. The file initially appears as Scheme 1.

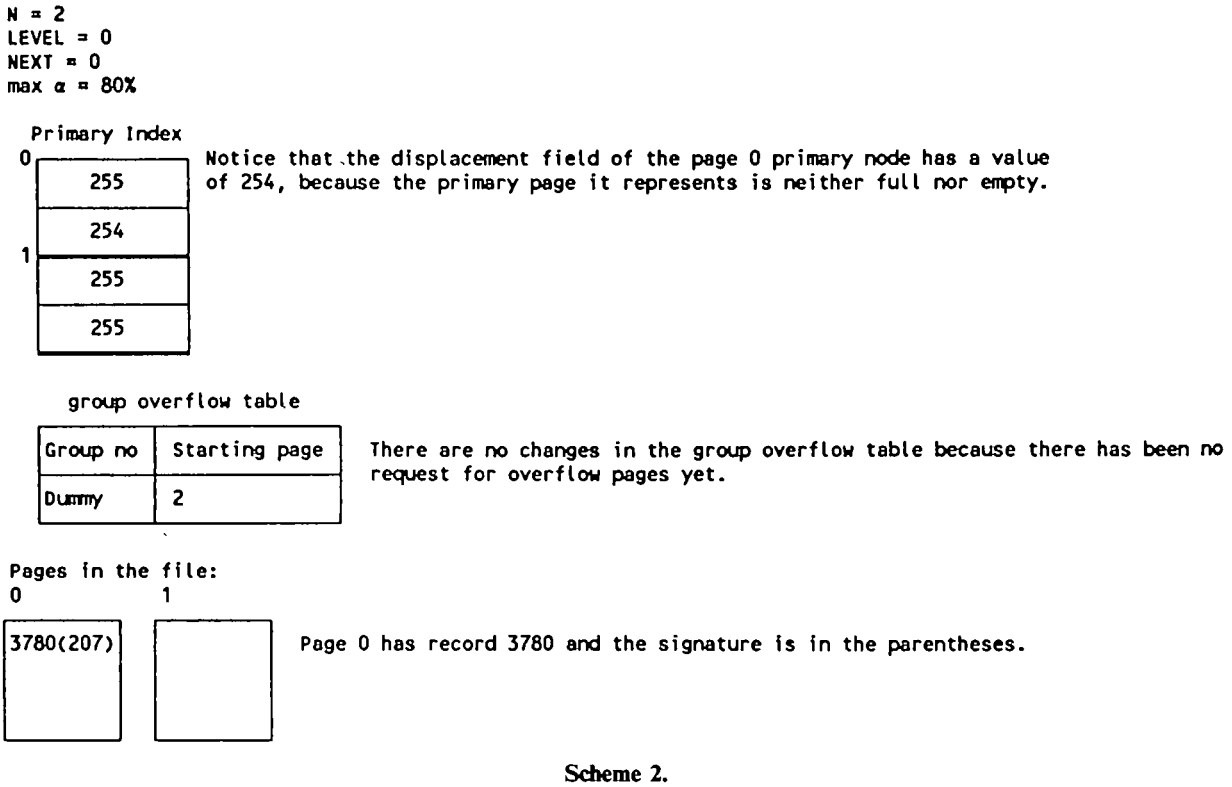
We try inserting the record with key 3780 and signature 207. We used Larson's recommendation in Ref. 7 of generating signatures using a random number generator with the hash value of the record key as the seed. Like linear hashing, the hash function used is *key* **mod** $N*(2^{LEVEL}) = 3780$ **mod** $2*2^0 = 0$. Instead of accessing page 0 straight away, we first examine the page 0 primary node which is the first node in the primary index. Since the displacement field of the page 0 primary node has a value of 255, page 0 must be empty. Therefore, record 3780 is stored in page 0. At this stage, the value of the separator for page 0 is unimportant because the page is empty; the only time that we need to use the signature value is when the page is full. Next the displacement field for the page 0 primary node is set to 254 because page 0 is no longer empty. After inserting record 3780, the α is computed. In this case, the α is 10% which is not more than 80% so there is no need to expand the file. The file now becomes Scheme 2.

2.2 Overflow pages

To demonstrate how overflow pages are processed, we insert record 8304 with signature value 214 into the file of



Scheme 1.



Scheme 2.

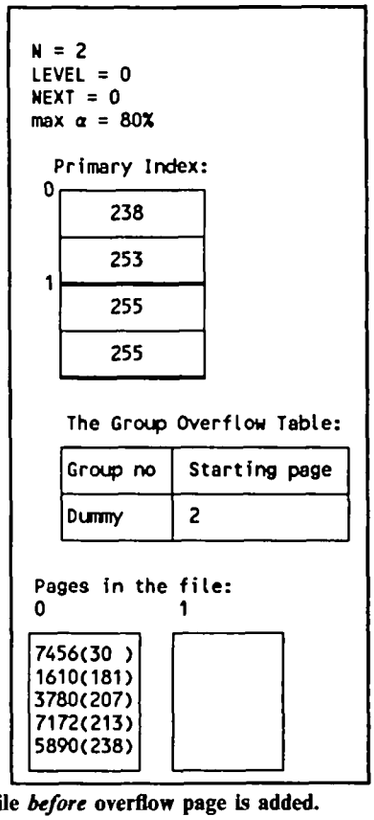


Figure 6. File *before* overflow page is added.

Figure 6. Record 8304 hashes to page 0 (8304 **mod** $2*2^0$). The value of 253 in the displacement field of the page 0 primary node indicates that the page is full. The page 0 separator has a value of 238 which is larger than 214, the signature of record 8304, so record 8304 must be placed in page 0. The record with the largest signature in page 0, 5890, is temporarily removed to make way for record 8304. If multiple records have the maximum signature value among all the key signatures in the page, then all those records must be temporarily removed. After inserting record 8304 into page 0, a new separator is chosen. According to our convention, the new separator for page 0 is 214 since it is the largest signature on the page.

The next task is to reposition record 5890 into an overflow page. According to Algorithm 2, page 0 belongs to group 0; according to Algorithm 1, the page 0 overflow block size is 4. Therefore, an entry for group 0 is added to the group overflow table. Its overflow block starts from page 2 where the dummy entry was. A new dummy entry is added after the group 0 entry with a starting page of 6. The page indicators for all the overflow pages assigned to group 0, that is, pages 2 to 5, are established as 0 because none of them is being used currently. Out of the four overflow pages assigned to group 0, page 2 is selected because it is the first unused page. The page indicator for page 2 is then set to 1.

A secondary node *representing* page 2 is included in the secondary index. Since page 2 is the only page which is being used, its node is the only node in the secondary index. All the fields in the page 2 secondary node are set to 255 because page 2 is empty. Now page 2 is accessed and record 5890 is inserted. The displacement field of the page 2 secondary node is changed to 254 because page 2 is no longer empty. Next, overflow page 2 must be linked

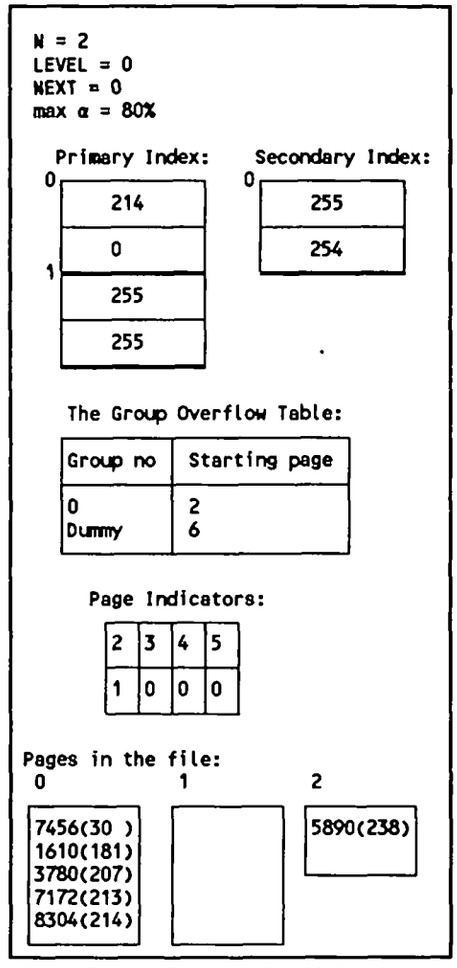


Figure 7. File *after* overflow page is added.

from page 0. The linking is done by updating the displacement field of the page 0 primary node to 0 because page 2 is the first overflow page in the overflow block assigned to group 0. Again the α is computed to ensure that it does not go beyond 80%. In this case, the α is $6/12 = 50\%$. The status of the file is shown in Figure 7.

2.3 Using separators

To demonstrate how the separator can improve the insertion performance, let us try inserting record 1796 with signature 253 into Figure 8. Record 1796 hashes to page 0. The displacement field in the page 0 primary node having a value less than 253 indicates that page 0 is full and there is an overflow page following it. In checking the signature field, it is apparent that the separator for page 0 is less than the signature of record 1796. Therefore, record 1796 cannot be placed in page 0. The next overflow page is checked. According to the page 0 primary node, the displacement field has a value of 0. From Algorithm 2, page 0 belongs to group 0; Algorithm 1 tells us that page 0 has an overflow block size of 4. Therefore, a displacement value of 0 implies overflow page 2 because the overflow block assigned to group 0 starts from page 2 as shown in the group overflow table.

According to the page indicators, page 2 is the only overflow page being used, so the page 2 secondary node

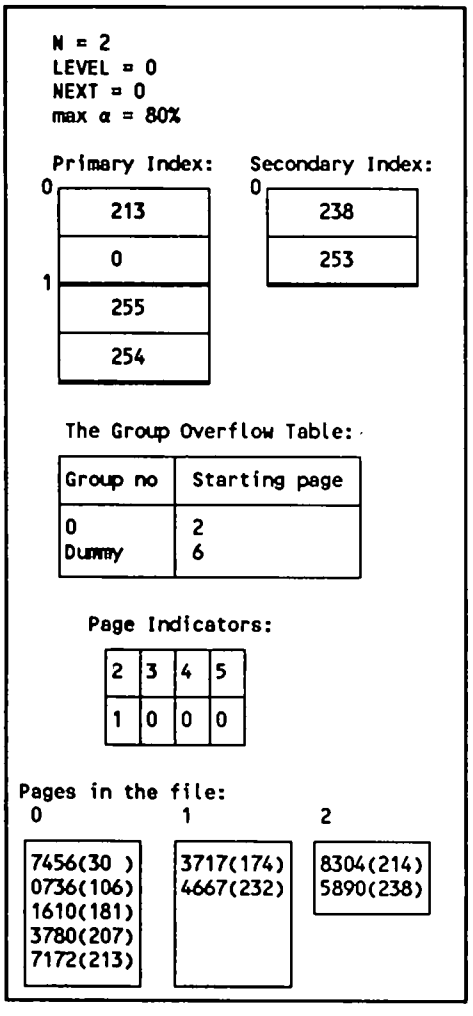


Figure 8. Use of separators; file *before* insertion of record.

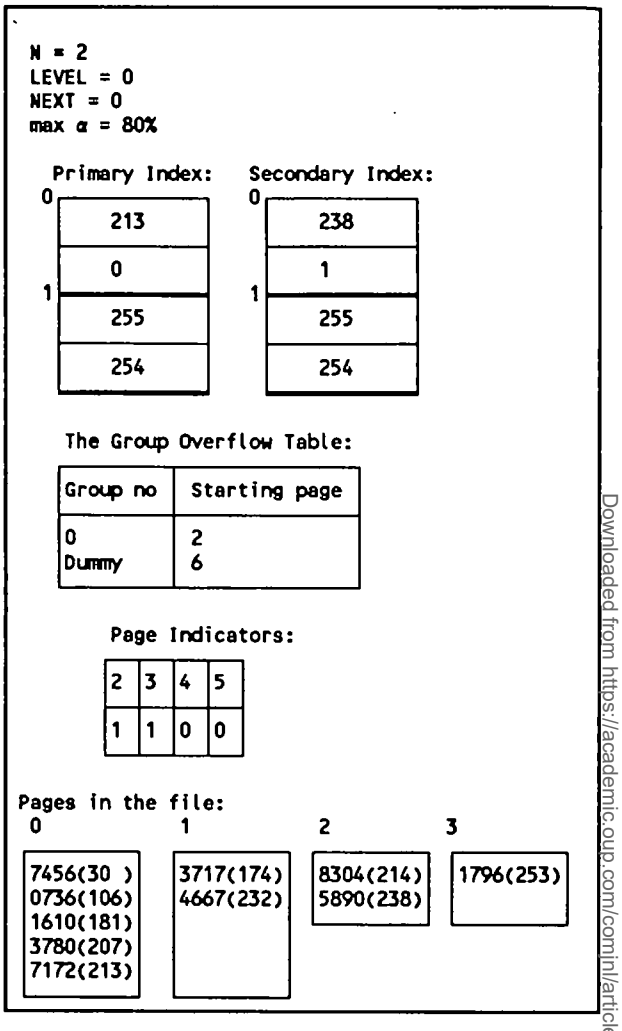


Figure 9. Use of separators; file *after* insertion of record; second overflow page added.

must be the only one in the secondary index. Accessing the page 2 secondary node, we know that page 2 is full because the displacement field has a value of 253. Again the separator for page 2 is examined. Because the signature for record 1796 is larger than the separator, record 1796 must be placed in the next overflow page. However, the displacement field for the page 2 node has a value of 253 which indicates that there are no overflow pages following page 2. Since page 0 belongs to group 0, and the group 0 overflow block consists of pages 2 to 5, the page indicators tell us that page 3 is the next available overflow page; therefore it is selected. The page indicator for page 3 is set to 1. A secondary node representing page 3 is added directly below the page 2 secondary node. All the fields of the page 3 secondary node are set to 255. Page 3 is accessed and record 1796 is placed there. The displacement field of the page 3 secondary node is set to 254 because page 3 is no longer empty.

We then need to link all the overflow pages belonging to page 0. This is done by updating the displacement field of the page 2 secondary node to 1 because page 3 has a displacement of 1 from page 2. Again the α is computed and since it does not exceed 80%, there is no expansion. The resulting file appears in Figure 9. By using separators, pages 0 and 2 do not need to be accessed when inserting record 1796; only page 3 does.

2.4 Expansion

To demonstrate expansion, we assume the file in Figure 10 which has an $\alpha > 80\%$. Because NEXT is 0, chain 0 is split. All the records in chain 0 must be temporarily stored in the record pool. The displacement field in the page 0 primary node tells us that page 0 is full and has an overflow page at a displacement of 0. Since page 0 belongs to group 0, and the group 0 overflow block begins with page 2, the first overflow page must refer to page 2. According to the page indicators, page 2 is the first of the used pages; therefore, its node must be the first node in the secondary index. The page 2 secondary node indicates that page 2 is full and has a successor overflow page at a displacement of 1 which implies page 3. The page 3 secondary node must be directly below the page 2 secondary node. The page 3 secondary node indicates that it is full, and there is no overflow page following it because its displacement field is 253. After the records have been moved to the record pool, the page 2 and 3 secondary nodes can be eliminated because the corresponding overflow pages are not being used. Actually the group 0 entry in the group overflow table can be eliminated because none of the overflow pages allocated to group 0 are used. For the same reason, the page indicators for the pages allocated to group 0 can be

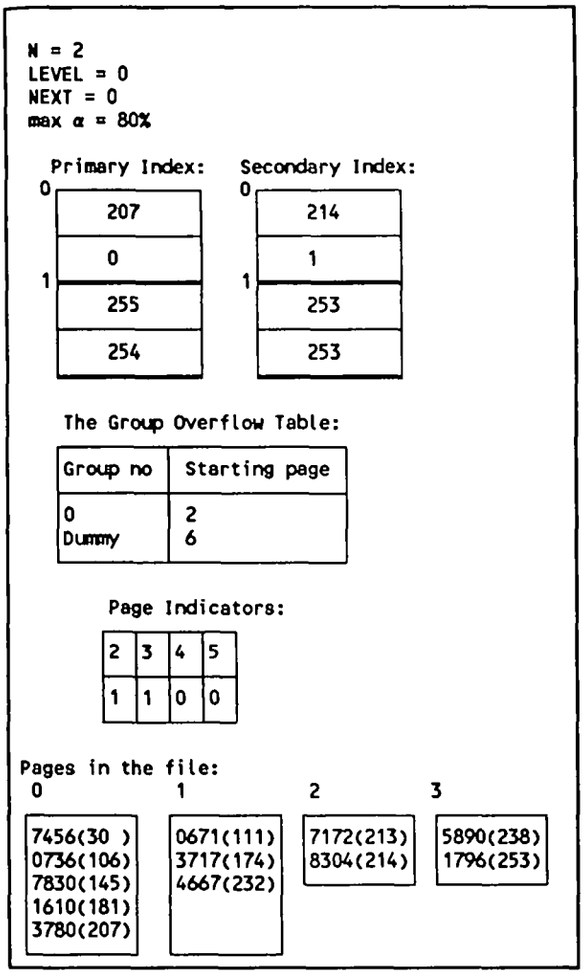


Figure 10. File *before* expansion of chain 0.

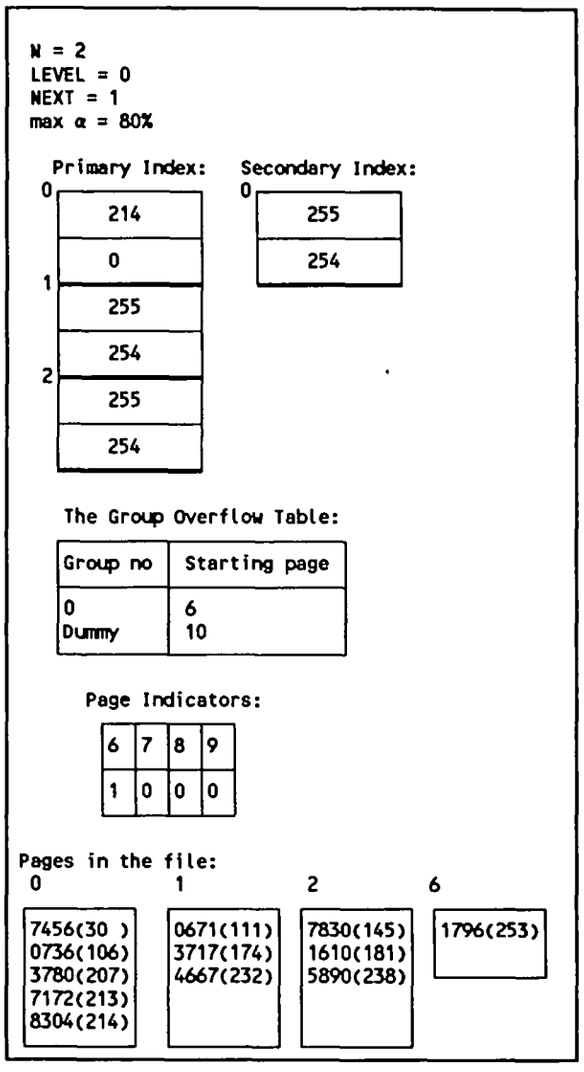


Figure 11. File *after* expansion of chain 0; overflow page for chain 0.

removed. After eliminating the group 0 entry, the group overflow table only has a dummy entry with a starting page of 6.

Now we are ready to compute the newly added page. Like linear hashing, it is $\mathrm{NEXT} + \mathrm{N}*2^{\mathrm{LEVEL}} = 0 + 2*2^0 = 2$. We need to check the group overflow table to determine if page 2 is being used as an overflow page. In this case, the table only has a dummy entry with a starting page of 6, which is beyond 2; therefore, page 2 is not being used as an overflow page. If page 2 were being used as an overflow page, then its respective block would have to be copied onto a new location to free page 2. A new node representing page 2 with its displacement field initialised to 255 is appended to the primary index. Now all the records in the record pool are ready for rehashing. Like linear hashing, these records are rehashed with the function *key* **mod** $\mathrm{N}*2^{\mathrm{LEVEL}+1}$, that is, *key* **mod** $2*2^1 =$ *key* **mod** 4. This rehashing is similar to inserting.

The record with key 7456 hashes to page 0; likewise for the record with key 736. Keys 7830 and 1610 hash to page 2. Keys 3780, 7172, and 8304 hash to page 0. Page 0 is now full. Key 5890 hashes to chain 2. Key 1796 hashes to page 0. Because key 1796 has a signature value of 253, which is greater than the largest signature of 214 for page 0, it cannot be placed in page 0. Instead we need to request an overflow page.

Algorithm 2 tells us that page 0 belongs to group 0 and Algorithm 1 tells us that the group size is four. A request for an overflow block of four pages is made. An entry for group 0 is added to the group overflow table. The overflow block assigned to group 0 consists of pages 6 to 9. The dummy entry now has a starting page of 10. Four page indicators, for pages 6–9, are added and set to 0 to indicate that none of them are used yet. There are only four page indicators because there is only one overflow block active. Page 6 is selected from the overflow pages.

A secondary node representing overflow page 6 is placed in the secondary index. We need to update the displacement field of the primary node of page 0 to 0 because its overflow page is the first one in the overflow block assigned to group 0. Now page 6 is accessed, and record 1796 is stored there. We then update the displacement field of the secondary node of page 6 to 254.

Once all the records in the record pool are rehashed, the variable NEXT is incremented by 1. If the new value of NEXT equals $\mathrm{N}*2^{\mathrm{LEVEL}}$, all the pages in that level have been split; LEVEL would be incremented by 1, and NEXT would be reset to 0. The new value of NEXT does not equal $\mathrm{N}*2^{\mathrm{LEVEL}}$; therefore, NEXT remains as 1. Because the α is $< 80\%$, another expansion is not needed. Figure 11 shows the file resulting from the expansion of chain 0.

2.5 Retrieval

For retrieval, let us try retrieving record 5890 with signature 238 from the file in Figure 12. The hash function is *key* **mod** $N*2^{LEVEL}$, which implies 4890 **mod** $2*2^1 = 5890$ **mod** $4 = 2$. Like linear hashing, we check if 2 is < NEXT; if it is, then we would apply the hash function *key* **mod** $N*2^{LEVEL+1}$. In this case, 2 is > NEXT, so page 2 is the proper address. Comparing the separator of the page 2 primary node and the signature of record 5890, we note that the signature of record 5890 is larger which implies that record 5890 cannot be in page 2. According to Algorithm 2, page 2 belongs to group 1, and according to the group overflow table, its overflow block begins with page 10. The displacement field value of 0 for the page 2 primary node refers to overflow page 10.

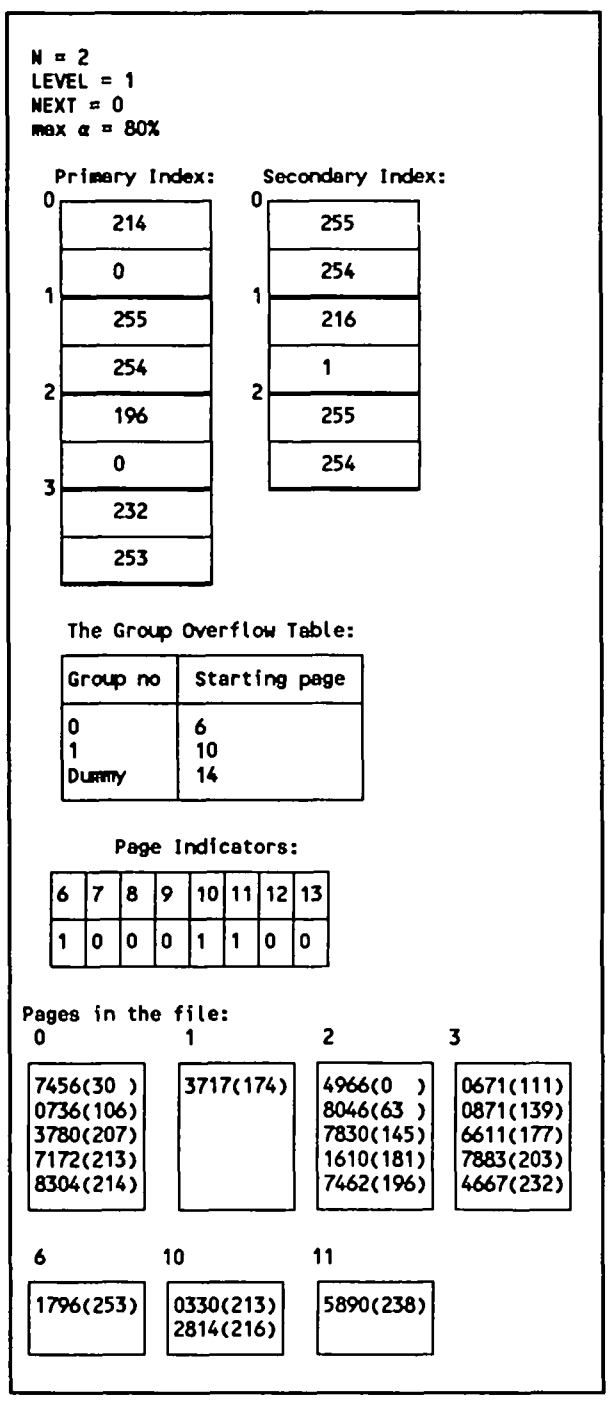


Figure 12. File for retrieval example.

The group overflow table indicates that group 0 is the only group before group 1. The group 0 block of overflow pages starts on page 6, and the group 1 block of overflow pages starts on page 10. The page 10 indicator must be the fifth indicator in the page indicators. In checking the page indicators, we realize that page 10 is the second overflow page which is used; therefore, the page 10 secondary node must be the second node in the secondary index. That node reveals that page 10 is full and the largest signature is 216 which is less than 238, the signature of record 5890. Therefore, record 5890 is not in page 10. The next overflow page is checked. According to the secondary node of page 10, the next overflow page has a displacement value of 1 which implies page 11. The page 11 secondary node is directly below the page 10 secondary node. Page 11, which is not full because it has a displacement value of 254, is accessed to search for record 5890. This example demonstrates one auxiliary memory access per successful retrieval.

For an example of unsuccessful retrieval, we try retrieving record 5203 with signature 255. Record 5203 **mod** $4 = 3$ which is larger than NEXT so page 3 is the proper address. From the page 3 primary node, we know page 3 is full, and its separator is 232 which is less than 255, the signature of record 5203. This comparison indicates that record 5203 is not in page 3, neither is it in the file because there is no overflow page following page 3. *No* probe of auxiliary memory is needed for this unsuccessful retrieval.

2.6 Deletion

Deletion consists of two steps: retrieving the record to be deleted and then physically removing it and updating the indexes, group overflow table, and page indicators as necessary. If the page has only one record before deletion, then after deletion it is empty and its displacement field is updated to 255. If the page is an overflow page, then the page can be freed, its page indicator reset to 0, and its corresponding node removed. Then the displacement field of the predecessor node is set to 253 because that page must be full. The entire overflow block can be freed if all its pages are not used. If this freed overflow block is the first overflow block among all the overflow blocks, then its entry can be removed from the group overflow table; otherwise, its group number can be replaced by the dummy entry.

If the page is not full before or after deletion, then no updating is necessary because the status of the page has not changed.

If the deleted record falls on a full page, then it is necessary to determine whether there are any overflow pages following it. If the displacement field value is less than 253, then there is an overflow page following. The record with the smallest signature on the following overflow page needs to be moved into the space vacated by the deleted record and the signature field of the node representing the page with a deletion needs to be updated. If there is no overflow page following, then after deleting the record, it is only necessary to update the displacement field to 254 because the page would be partially full. If the overflow page following the page of the deletion has more than one record with the same minimum signature,

it would not be possible to move multiple records to the page of the deletion because there is only room for one. In that case, a new separator must be selected from the remaining records in that page. If the deleted record is not the separator for the page, then the signature field is not affected since there are no new records moving into the page.

Next the vacancy on the following overflow page needs to be filled. This page is handled like the one with the deleted record. This process continues until the last page of the chain.

Because linear hashing is often used for applications where deletion is infrequent, the overall performance of HSUSPR would not be strongly affected by the deletion cost. HSUSPR has lower deletion costs than an algorithm which uses linear probing such as LHS.

After updating the indexes, group overflow table, page indicators, and after returning the overflow page for reuse, the storage utilisation is checked to determine if it is below the limit set by the user. If it is, the file size contracts to contain one fewer chain. All the records belonging to the most-recently-added chain are moved to the record pool; its primary and secondary nodes are removed and the data pages are returned for reuse. The value of NEXT is decremented; if it is negative, then LEVEL is decremented. NEXT is then set to $N*2^{LEVEL}-1$. Then all the records in the record pool are rehashed as with insertion.

3. EXPERIMENTAL RESULTS

To ascertain its effectiveness, we tested HSUSPR with a file of 20000 authors' names. Both the primary and overflow page sizes were 10 records, α was 80%, the separator length was one byte, the maximum group size was 128, and all data were collected at level 8 (more than 1024 pages). The data were shuffled and run 4 times. The averages plus 95% and 99% confidence levels for one sided t-tests[8] for the true means are presented in Tables 2, 3, and 4.

Table 2 presents the total insertion costs which include the cost of insertion, expansion, and moving overflow pages due both to overflow pages due both to overflow block expansion and overflow pages being included as a primary page. The average total insertion cost was 2.70

Table 2. Average insertion costs, 95%, and 99% confidence measurements for an α of 80%

Costs	Average: no. probes	Upper bound 95% confidence no. probes	Upper bound 99% confidence no. probes
INSERTION	1.960	1.963	1.966
EXPANSION	0.39	0.40	0.41
PGMOBKEXP/RE	0.174	0.180	0.186
PGMOFLEXP/RE	0.0017	0.0057	0.0094
TOTAL COST	2.70	2.73	2.75

INSERTION: The insertion cost per record inserted at level 8.

EXPANSION: The expansion cost per record inserted at level 8.

PGMOBKEXP/RE: The number of overflow pages moved due to a block expansion per record inserted at level 8.

PGMOFLEXP/RE: The number of overflow pages moved due to overflow pages being included as the primary page per record inserted at level 8.

TOTAL COST: The total cost for inserting a record at level 8. This cost includes the insertion cost, expansion cost, and the cost of moving overflow pages, both due to overflow block expansion, and overflow pages being included as a primary page.

Table 3. Average, 95% and 99% confidence measurements for the following: record pool size, maximum number of page indicators, and maximum number of overflow pages which are used for an α of 80%

Costs	Average: no. records	Upper bound 95% confidence no. records	Upper bound 99% confidence no. records
RECPL	27.25	27.34	27.42

Costs	Average: no. pages	Upper bound 95% confident: no. pages	Upper bound 99% confidence: no. pages
MPAGEARR	3008	3008	3008
MUSEPAGE	1252.25	1284.23	1313.97

RECPL: The record pool which temporarily holds all the records during an expansion. The value is in number of records.

MPAGEARR: The maximum total number of page indicators needed at level 8. It is the same as the total number of overflow pages allocated at level 8 because each overflow page allocated has one page indicator.

MUSEPAGE: The maximum total number of overflow pages that are used at level 8.

Table 4. Main memory storage needed in number of bits per chain for 80% α; averages, 95%, and 99% confidence measurements

Costs	Average: no. bits	Upper bound 95% confidence no. bits	Upper bound 99% confidence no. bits
MOLTENTRY/CN	0.25	0.26	0.27
MPAGEARR/CN	1.32	1.34	1.36
STOARGEINBIT	1.57	1.60	1.62
TOTALSTORAGE	17.57	17.60	17.62

MOLTENTRY/CN: The maximum number of entries in the group overflow table per chain in terms of bits. A chain includes the primary pages, and the maximum overflow pages used at level 8.

MPAGEARR/CN: The number of page indicators shared by a chain at level 8. A chain includes the primary pages, and the maximum overflow pages used at level 8.

STORAGEINBIT: The storage in bits per chain at level 8. This value is obtained by adding MOLTENTRY/CN, and MPAGEARR/CN.

TOTALSTORAGE: The total storage per chain in terms of bits at level 8. This value is obtained by adding 16 to STORAGEINBIT. The value 16 is added because the index needs two bytes per chain.

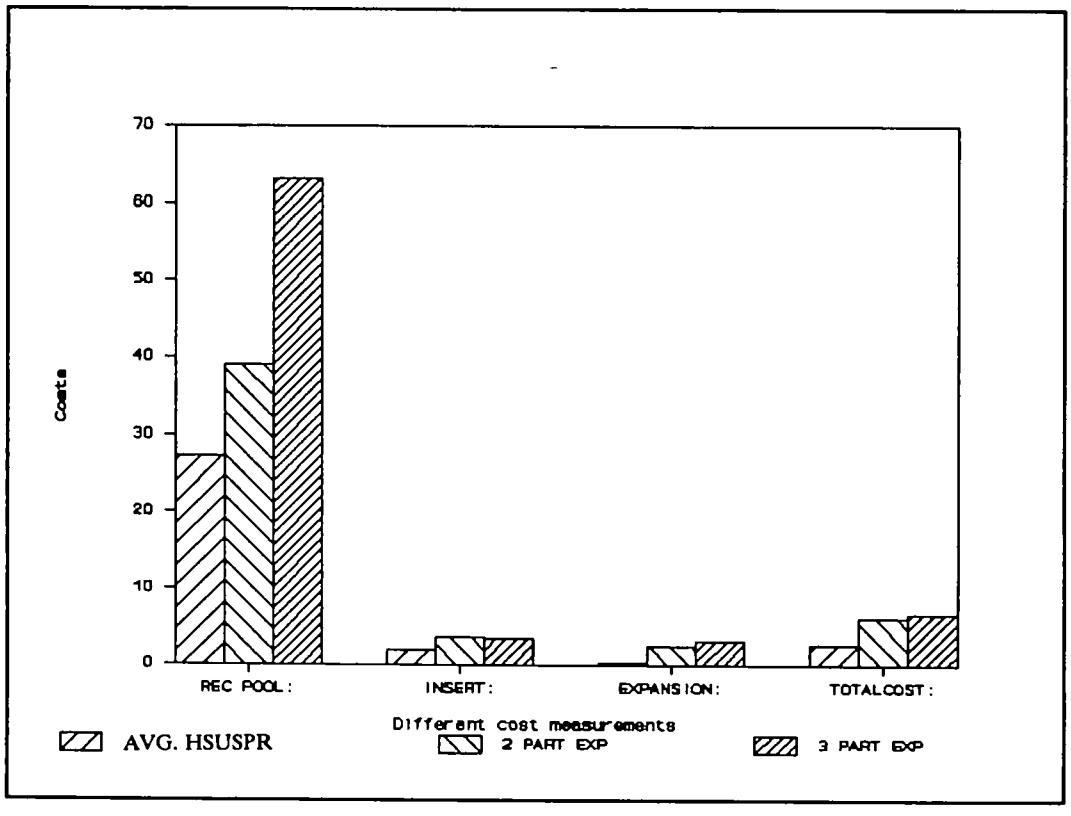


Figure 13. Comparison of costs for LHS versus HSUSPR.

probes, and with a 99% confidence that no more than 2.75 probes would be required.

Table 3 presents the record pool size, the maximum number of page indicators, and the maximum number of overflow pages used. The average record pool size was 27.25 records per expansion, and with a 99% confidence fewer than 27.42 records per expansion.

Table 4 presents the total primary storage needed per chain which includes the maximum number of entries in the group overflow table per chain, and the number of page indicators per chain where the chain includes the primary pages and the maximum number of overflow pages. In addition, the storage needed for the indexes is added. With a 99% confidence, each chain required no more than 18 bits.

We compare the HSUSPR linear hashing results with those of Linear Hashing with Separators (LHS) for an α of 80%. Figure 13 contrasts the averages of HSUSPR with those of LHS for two and three partial expansions. Figure 14 does the same for 95% and 99% confidence measurements. Both figures contrast the record pool size, insertion, expansion, and total insertion costs. Figures 13 and 14 indicate that the HSUSPR record pool size per expansion is about half of that needed by three partial expansions, and abut 70% of that needed by two partial expansions.

Typically the average, 95%, and 99% confidence measurements for the record pool size were 27.25, 27.34 and 27.42 respectively versus 39, and 63.2 records for two and three partial expansions respectively. The computation for obtaining the average record pool size for two and three partial expansions is shown in Ref. 6. It is also apparent from Figures 13 and 14 that the total insertion cost is less than half of that needed by LHS. According to Table 2, the total insertion cost for HSUSPR was 2.73 and 2.75 probes with 95% and 99% confidence respectively versus 6.1 and 6.6 for two and three partial expansions respectively. A second test was performed with alphabetical data (authors' names) for an α of 90%, an initial file size of two pages, a primary and overflow page size of 10, a maximum group size of 128, and data collected at level 7 (more than 512 pages). The results are shown in Tables 5, 6, and 7.

Table 5 presents the insertion cost, expansion cost, the cost of moving overflow pages due to overflow block expansion and the overflow pages being included as primary pages, and the total insertion costs. Table 6 contains the record pool size, the maximum number of page indicators, and the maximum number of overflow pages. Table 7 has the maximum number of entries in the group overflow table per page, the maximum number of page indicators per page, and the total storage needed per page. All these results are in terms of bits per chain where the chain includes the primary pages and the maximum overflow pages used. The total storage needed per chain includes the storage for the indexes. For an α of 90%, the overflow block needs to be more than 256 pages but fewer than 512 pages; therefore, the displacement field needs to be 9 bits instead of 8. Hence, a node requires 17 bits. The total storage needed per page is fewer than 19 bits.

Larson did not record the LHS results for an α beyond 85% because he said that the performance was so unfavourable. Therefore we cannot *directly* compare HSUSPR with LHS; however, we do compare HSUSPR results with an α of 90% with those of LHS for an α of *only* 85%. Despite HSUSPR being tested with a higher α, its performance was better than that of LHS. The comparisons appear in Figures 15 and 16. Figure 15 contrasts the HSUSPR averages with LHS of 2 and 3

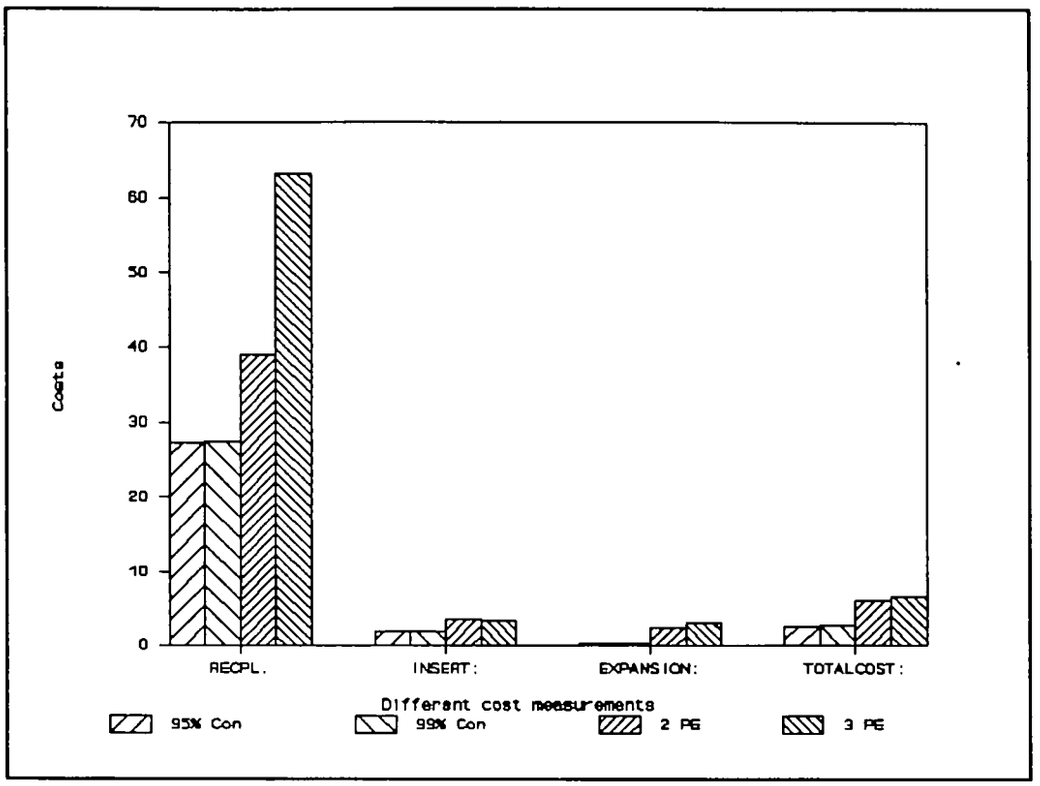


Figure 14. Comparison of costs from LHS for two and three partial expansions with those from HSUSPR with the 95% and 99% confidence measurements.

partial expansions. Figure 16 contrasts the 95% and 99% confidence measurements for HSUSPR with those of 2 and 3 partial expansions for LHS.

Figures 15 and 16 show that the record pool size for our 90% α is only about 60% of the size required for 3 partial expansions, and about 90% of the size needed for 2 partial expansions with an α of only 85%. Our average, 95%, and 99% confidence readings are 63.5, 64.0, and 64.4 records respectively versus 71.8 and 107.6 records for 2 and 3 partial expansions with an 85% α.

Our total insertion cost was only about half of what was needed for 2 and 3 partial expansions *despite* our

Table 5. Average insertion costs, 95%, and 99% confidence measurements for an α of 90%

Costs	Average: no. probes	Upper bound 95% confident: no. probes	Upper bound 99% confident: no. probes
INSERTION	3.32	3.36	3.40
EXPANSION	0.31	0.34	0.37
PGMOBKEXP/RE	0.18	0.19	0.21
PGMOFLEXP/RE	0.00	0.00	0.00
TOTAL COST	3.98	4.07	4.15

INSERTION: The insertion cost per record inserted at level 7.

EXPANSION: The expansion cost per record inserted at level 7.

PGMOBKEXP/RE: The number of overflow pages moved due to a block expansion per record inserted at level 7.

PGMOFLEXP/RE: The number of overflow pages moved due to overflow pages being included as the primary page per record inserted at level 7.

TOTAL COST: The total cost for inserting a record at level 7. This cost includes the insertion cost, expansion cost, and the cost of moving overflow pages, both due to overflow block expansion, and overflow pages being included as a primary page.

Table 6. Average, 95%, and 99% confidence measurements for the following: record pool size, maximum number of page indicators, and maximum number of overflow pages for a 90% α

Costs	Average: no. records	Upper bound 95% confidence: no. records	Upper bound 99% confident: no. records
RECPL	63.54	64.01	64.44

Costs	Average: no. pages	Upper bound 95% confident: no. pages	Upper bound 99% confident: no. pages
MPAGEARR	3200	3344.18	3478.25
MUSEPAGE	1953	1992.17	2028.60

RECPL: The record pool which temporarily holds all the records during an expansion. The value is in number of records.

MPAGEARR: The maximum total number of page indicators needed at level 7. It is the same as the total number of overflow pages allocated at level 7 because each overflow page allocated has one page indicator.

MUSEPAGE: The maximum total number of overflow pages that are used at level 7.

Table 7. Main memory storage needed in number of bits per page for an α of 90%: averages, 95%, and 99% confidence measurements

Costs	Average: no. bits	Upper bound 95% confident: no. bits	Upper bound 99% confident: no. bits
MOLTENTRY/PG	0.35	0.37	0.40
MPAGEARR/PG	1.30	1.35	1.39
STOARGEINBIT	1.65	1.70	1.74
TOTALSTORAGE	18.65	18.70	18.74

MOLTENTRY/CN: The maximum number of entries in the group overflow table per chain in terms of bits. A chain includes the primary pages, and the maximum overflow pages used at level 7.

MPAGEARR/CN: The number of page indicators shared by a chain at level 7. A chain includes the primary pages, and the maximum used overflow pages at level 7.

STORAGEINBIT: The storage in bits per chain at level 7. This value is obtained by adding MOLTENTRY/PG, and MPAGEARR/PG.

TOTALSTORAGE: The total storage per chain in terms of bits at level 7. This value is obtained by adding 17 to STORAGEINBIT. The value 17 is added because our index needs 17 bits per chain.

α being 5% higher. The total insertion cost for 2 and 3 partial expansions were 8.94 and 9.38 probes per record respectively versus 3.98, 4.07, and 4.15 probes per record for our averages, 95% and 99% confidence measurements.

Next we consider the effect of different distributions of key data on performance. Four sets of 25000 numerical values each were created. Each set of data had a different distribution: one set had a heavy concentration in the beginning range of numbers, another set had a heavy concentration in the middle range of numbers, the third set had a heavy concentration in the high number range, and the fourth set had a heavy concentration in both the beginning and ending range of numbers. These sets of data were tested with the following parameters: initial file size of 2, primary and overflow page size of 10, α of 80%, and a maximum group size of 128. Each set of data was tested 4 times. The results collected from the 16 runs were tested with the Single-Factor Analysis of Variance test.[8] The test results indicated that the four distributions of data did not affect the performance of HSUSPR.

In general, our simulation suggests three major points:

1. By removing secondary clustering, the insertion, expansion, and the total insertion costs were less, and the record pool size was smaller.
2. HSUSPR linear hashing has high storage utilisation, and our performance results with 90% storage utilisation are more favourable than those of Linear Hashing with Separators for a storage utilisation of 85%.
3. The performance of HSUSPR linear hashing is independent of the key distributions.

4. THE ADVANTAGES OF HSUSPR LINEAR HASHING

The advantages of HSUSPR linear hashing include:

1. One access to secondary memory for successful retrieval.
2. Maximum of one access to secondary memory for an unsuccessful retrieval.

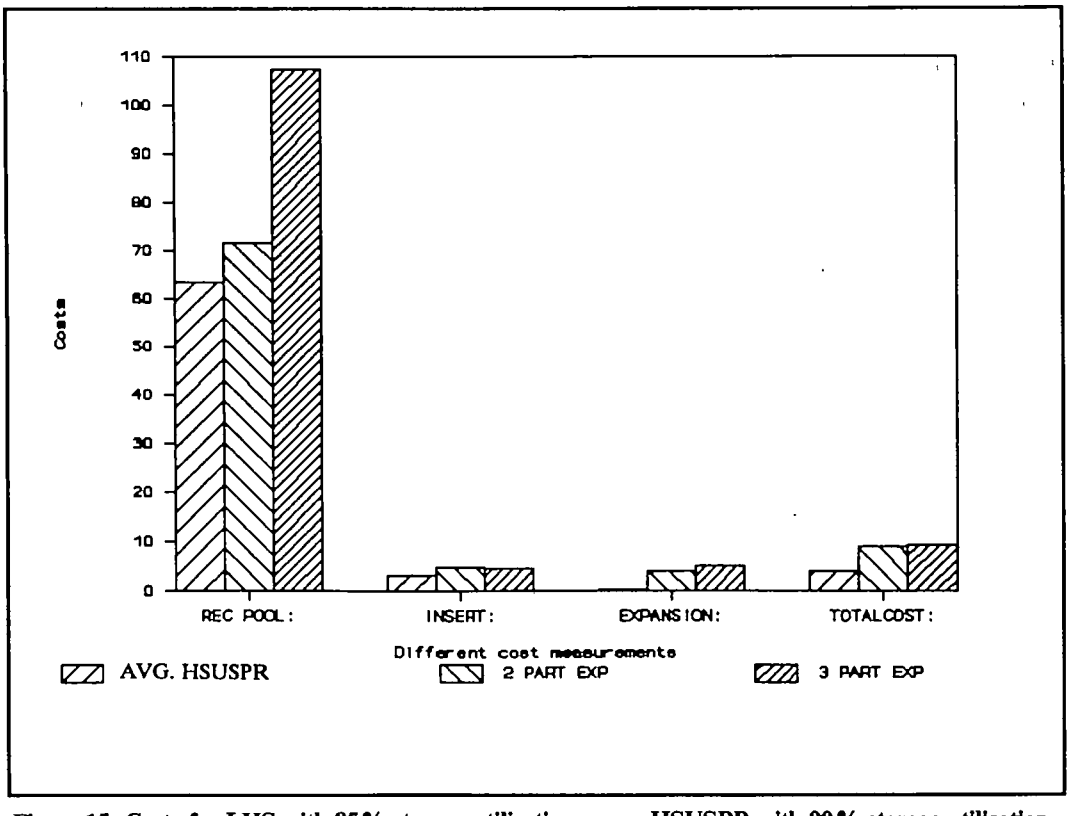


Figure 15. Costs for LHS with 85% storage utilisation versus HSUSPR with 90% storage utilisation.

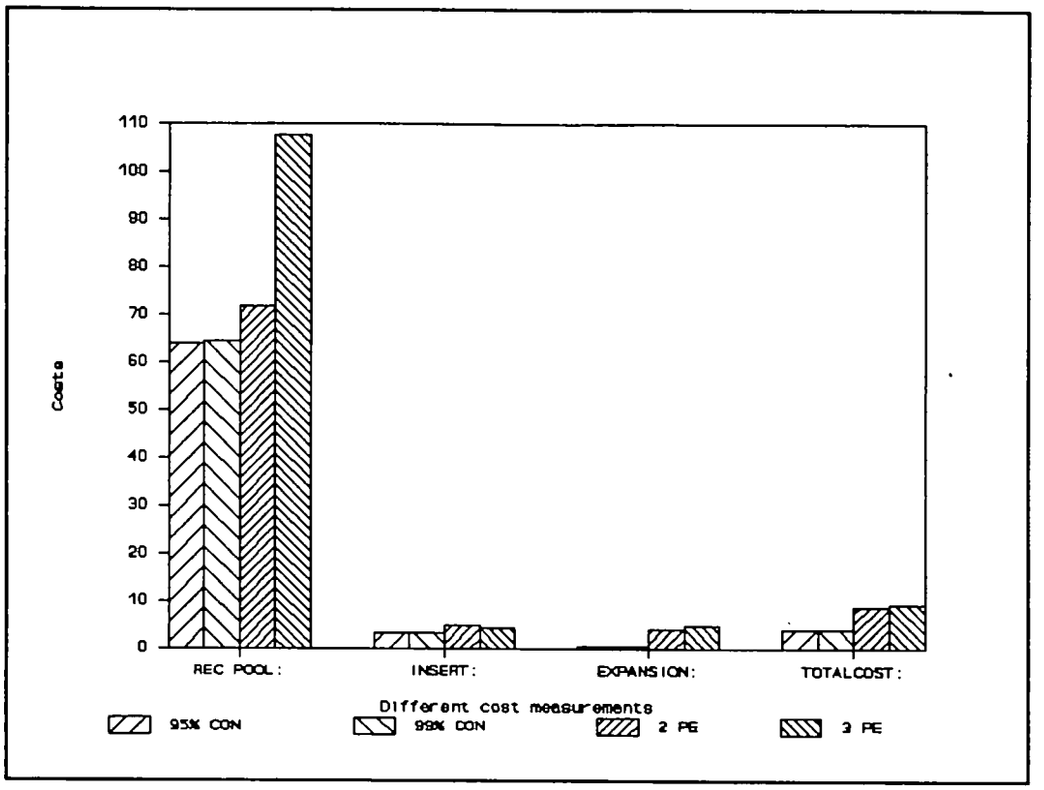


Figure 16. Costs for LHS with 85% storage utilisation versus HSUSPR with 90% storage utilisation for 95%, and 99% confidence measurements.

3. Stability and predictability, since we can guarantee successful retrieval in one access or a maximum of one access for unsuccessful retrieval.
4. Support for extendibility.
5. Direct and simple computation of addresses as compared with Linear Hashing with Separators.
6. Lower insertion and expansion costs since there is no secondary clustering as compared with Linear Hashing with Separators.
7. A relatively small record pool for holding records during an expansion because there is no secondary clustering.
8. About two byte nodes: one byte for the separator field, and the next byte for the displacement field. These sizes agree with one of our goals.
9. Chaining without secondary clustering. Since there is no secondary clustering, all the records in a page must belong to a particular chain. Then each chain tends to have fewer records. With fewer records in a chain, there is a lower chance for the file to 'wander away'[4] because there is a lower chance for b records to have the same signature values, where b is the capacity of the page.
10. Controllable storage utilisation.
11. High storage utilisation. Storage utilisation can be set as high as 90%.

5. THE COSTS OF HSUSPR LINEAR HASHING

The good performance of HSUSPR linear hashing is not without costs. They include:

1. The current version of the modified algorithm is more complex because we need to handle multiple overflow blocks as a result of limiting the displacement field to only 256 unique values.
2. Holes may be created within an overflow block. This phenomenon exists because we break up the overflow pages into blocks so that the displacement field of 8 bits can reach out for all the overflow pages in a block.
3. During an insertion, records may propagate. However, it must be remembered that this propagation is the cost of obtaining one access per successful retrieval, and at most one access per unsuccessful retrieval.
4. During an expansion, when one of the overflow pages in an overflow block is to be included as a primary page, the whole overflow block must be reallocated which means, we may need to do multiple copying. The worst case is, if the whole overflow block is full then all the overflow pages within that block must be copied.
5. A relatively small amount of storage is needed for the page indicators: one bit per overflow page.
6. Memory space is needed for the group overflow table. The size of the group overflow table depends on the size of the entry in the table and the number of entries in the whole table. In our simulation, each entry was allocated 24 bits.

6. CONCLUSIONS

In conclusion, HSUSPR linear hashing is recommended for applications where insertion costs need to be low, and retrieval time needs to be fast and constant. Examples of such applications are: medical information systems, databases for books, articles, and albums, databases for

selected groups of the population such as prisoners, drivers, and drug addicts, databases used by compilers and text editors, and information systems which are accessible by networks.

Acknowledgement

The authors wish to thank an anonymous referee for the helpful suggestions for improving the presentation of the algorithm.

REFERENCES

1. R. Fagin, J. Nievergelt, N. Pippenger and H. R. Strong, Extendible hashing – a fast access method for dynamic files, *ACM Transactions on Database Systems*, **4** (3), 315–344 (1979).
2. P. A. Larson, Dynamic hashing, *BIT*, **18** (2), 184–201 (1978).
3. Witold Litwin, Linear Hashing: a new tool for file and table addressing, *Proc. 6th International Conference on Very Large Database*, Montreal, 212–223 (1980).
4. P. A. Larson, Linear hashing with separators – a dynamic hashing scheme achieving one-access retrieval, *ACM Transactions on Database Systems*, **13** (3), 366–388 (1988).
5. Alan L. Tharp, *File Organization and Processing*, Wiley, New York 398 (1988).
6. Seng Fuat Ou, HSUFSP linear hashing, Internal Report, N.C. State University, Raleigh NC, U.S.A. (1990).
7. P. A. Larson, Personal Communication (1990).
8. Devore Jay L. *Probability and Statistics for Engineering and the Sciences*, Brooks/Cole Publishing Company, Monterey, California (1982).

Book Review

GORDON BLAIR, JOHN GALLAGHER, DAVID HUTCHISON and DOUG SHEPHERD
Object-Oriented Languages, Systems and Applications
Pitman Publishing, London, £22.95
0-273-03132-5

Object-Oriented Languages, Systems and Applications is a collection of chapters by various authors covering a variety of object-oriented topics. It aims to provide comprehensive coverage of the object-oriented paradigm and its applications, rather than concentrating on one application area.

The book is logically divided into four parts. The first five chapters (Part I) are concerned with basic concepts. The first chapter introduces the notion of 'object-orientation' and describes the structure of the book. A 'dependency chart' is provided showing the interrelationships between the chapters, and a guide is provided for readers who may wish to work through the book in stages or from different viewpoints, rather than from beginning to end. After this introductory chapter, there are three chapters on basic concepts. The first of these introduces the well-known object-oriented concepts of encapsulated objects, classes and inheritance. Chapter 3 discusses some variations on the traditional approaches. It introduces a number of techniques for behaviour sharing and evolution, including delegation, actors and photocopying, which are associated with class-less systems. Chapter 4 introduces concepts associated with abstract data types (ADTs), and includes a useful section on the differences between types and classes. Chapter 5 then asks the question: 'What *are* Object-Oriented Systems?' A model of object-oriented computing, embracing four dimensions – encapsulation, classification, polymorphism and interpretation – is discussed. This is followed by an introduction to formal approaches to object orientation which seems a little out of place; since the subject is re-introduced in the final chapter, this material might have been better placed there.

The next five chapters (Part II) discuss the application of the object-oriented approach to the following areas respectively: programming languages, database systems, design methods, distributed systems and interactive user interfaces. These serve to illustrate the wide-ranging applicability of the object-oriented paradigm. Although these chapters are structured in different ways (having been written by different authors), they all introduce and compare different systems/languages, after covering general aspects and issues in the relevant application areas. Chapters 11 and 13 (Part III) describe three specific object-oriented products: an object-oriented processor, REKURSIV; an object-oriented language, BETA; and an object-oriented database system, Iris. The final chapter (Part IV) considers future directions for research. The authors stress the importance of embarking on a period of consolidation, obtaining practical experience and establishing consistent terminology and semantics. It is interesting to note that in chapter 5 the model of object-oriented computing which is presented is said to provide a 'more general interpretation [which] widens the scope of object orientation'. Some would argue that the scope should be narrowed, not widened! But perhaps this widening in scope is an essential prerequisite for an effective period of consolidation and subsequent narrowing of concepts and terminology. In this final chapter, specific areas of importance for the future are singled out: software engineering methodologies, formal methods, sharing and distribution, ODP standardization work and multimedia objects.

The book is oriented towards postgraduate computer scientists and computer professionals; with its broad coverage of object-oriented topics, it is certainly important reading for researchers in the field. The authors suggest that it could form the basis of a final-year undergraduate option course, but I feel that some awareness of the widely recognised object-oriented concepts is essential for a full appreciation of the more specialised material and the issues addressed.

A large number of authors have contributed to the book. Although this could have resulted in a very disjointed production, the editors have attempted to ensure that the coverage of particular topics is balanced, and have provided a guide to the structure of the work. However, while there are certain advantages in having different authors for the chapters on application areas and specific products, it might have been more appropriate for the chapters on basic concepts to be written by a single author. These chapters are not as integrated as they might be. It must be said though, that there is a wealth of material in this book for anyone with a basic knowledge of object-oriented computing wishing to extend that knowledge.

ELIZABETH OXBORROW
Canterbury

Announcement

International Journal of Applied Intelligence

Special issue on
Distributed AI in Manufacturing. Guest editor Mark S. Fox, Carnegie Mellon University.

Papers are solicited for this issue on any topic that demonstrates the use of DAI in any facet of industrial organisation such as design, planning, production, distribution, field service, sales, marketing, finance, etc. The following are examples of topics.

- Concurrent design
- Control of autonomous automated guided vehicles
- Enterprise integration
- Group sales forecasting
- Distributed field service
- Distributed planning and scheduling

Dates to note: deadline for paper submissions March 1992; scheduled for publication 1 July 1992.

Please send your papers to:
Karen S. Cullen,
Kluwer Academic Publishers,
101 Philip Drive,
Norwell, MA 02061, USA