Algorithms for Parsing Search Queries in Inverted File Document Retrieval Systems

by

Jane W. S. Liu

November, 1975
Algorithms for Parsing Search Queries in Inverted File Document Retrieval Systems

by

Jane W. S. Liu

November, 1975

This work was supported by the National Science Foundation under Grants NSF DCR 73-07980 and NSF DCR 72-03740 A01.
Abstract

In an inverted file document retrieval system, a query is in the form of a Boolean expression of index terms. In response to a query, the system accesses the inverted lists corresponding to the index terms, merges them and selects from the merged list those documents that satisfy the search logic. In this paper, we consider the problem of determining a Boolean expression which leads to the minimum total merge time among all Boolean expressions that are equivalent to the expression given in the query. This problem is the same as finding an optimal merge tree among all trees that realize the truth function determined by the Boolean expression in the query. Several algorithms which generate optimal merge trees, when the lengths of overlaps between different lists are small compared with the length of the lists, are described. These algorithms are no longer optimal when the lengths of overlaps cannot be neglected. In this case, it is possible to bound the performance of these algorithms in some instances in terms of the maximum overlap between lists. The performance bounds are discussed.
Acknowledgment

The author wishes to thank Drs. D. J. Kuck and W. Stellhron for their comments and suggestions.
I. Introduction

In this paper, we consider the problem of parsing search queries in inverted file document retrieval systems. In such systems, the index file contains an entry for each of the index terms selected as descriptors for the documents in the data file. Each entry in the index file contains a pointer to an inverted list of pointers in the postings file. The pointers in this list in turn point to all the documents in the data file that contain the corresponding index term. This file organization has been studied extensively and is used in many well known systems [1-4].

A query to an inverted file document retrieval system is in the form of a Boolean expression of index terms. For example, to request information on scheduling or resource management policies in time-shared systems, a user may present to the system a query

"Time Shared" \cdot ("Scheduling Policy" + "Resource Management Policy")  \tag{1-1}

where \cdot and + are the AND and OR operators, respectively. In response to a request, the system accesses the inverted lists in the posting files corresponding to the index terms in the query, merges them and selects from the merged list those documents that satisfy the search logic. In our example, the union of the lists corresponding to the index terms "Scheduling Policy" and "Resource Management Policy" is the list of pointers to the documents on scheduling or resource management policies. Let us denote this list by A. The list A is obtained by merging the two lists with duplicated entries deleted. We call the process of merging two lists to obtain their union an OR merge. The intersection of two lists is obtained by merging them and deleting from the merged list all entries except the
duplicated ones. We call the process of merging two lists to obtain their intersection an AND merge. In this example, pointers to documents to be selected as response to the query in (1-1) are obtained by AND merging the list A with the list corresponding to the index term "Time Shared." In other words, these three lists are merged in the order specified by the tree in Fig. 1-la. We label the leaves of the tree by the index terms and the internal nodes by the Boolean operators corresponding to the merges.

We note that the query in (1-1) is equivalent to the query

"Scheduling Policy"."Time Shared"+"Resource Management"."Time Shared"

To answer this query, the corresponding lists are merged in the order specified by the tree in Fig. 1-lb. Clearly, the times required to produce the response might be different for the two equivalent queries written in different Boolean forms.

We are concerned here with the problem of determining a Boolean expression which leads to the minimum total merge time among all Boolean expressions that are equivalent to the expression given in the query. For this purpose, we describe the inverted file document retrieval system schematically as shown in Fig. 1-2. To process a query, the lists corresponding to the index terms in the query are read into the buffer memory and are merged after they have been placed in the buffer. The total retrieval time is equal to the sum of the

†The total merge time computed here is equal to the amount of time required of the merge processor to process the query. The merge order that minimizes this time does not minimize the total retrieval time in general. However, keeping the total merge time minimized becomes important in multiuser systems in which many users share a large buffer memory. While the lists corresponding to the index terms in the query of a user are being merged, the lists specific in the queries of other users can be loaded into the buffer.
"scheduling" policy

"resource" management

"time shared"

(a)

"scheduling" policy

"time" shared "resource" management "time" shared

(b)

Figure 1-1
time required to access the lists from the secondary memory and the merge
time of the lists. In this paper, no attempt is made to minimize the
former. (The average access time of the lists from the secondary memory
has been estimated elsewhere [5,6]. The dependence of list access time
on the access algorithm and buffer management scheme used in the system
is the subject of a separate study [7].) Furthermore, only the case of
two-way merges is considered here.

In Section II, we introduce the terminologies and notations
necessary in our discussion. In Section III, we assume that the lengths
of overlaps between different lists are very small compared with the lengths
of the lists. Hence, in computation of the total merge time, lengths of
overlaps between lists can be neglected. Several algorithms are described
in Section III. These algorithms allow us to find optimal Boolean
expressions when queries are written as nested Boolean expressions in which
(i) all variables are distinct, and (ii) the complement of any variable, \( \bar{B} \),
can appear only in product terms, such as \( A_1 \cdot A_2 \cdot \ldots \cdot A_n \cdot \bar{B} \), with at
least one of the variables \( A_i \) being uncomplemented. These algorithms are
no longer optimal when the lengths of overlap between lists cannot be
neglected. In this case, it is possible to bound the performance of these
algorithms in some instances in terms of the maximum overlap between lists.
We discuss these bounds in Section IV.
II. Notations

Throughout our discussions, we use upper case letters (e.g., A, B, C, D) to denote both index terms and their corresponding inverted lists. Lower case letters are used to denote the lengths of the corresponding lists (e.g., a, b, c and d denote the lengths of the lists A, B, C, and D, respectively). Let $\sigma(A,B)$ denote the lengths of overlap between lists A and B. The lengths of the resultant lists obtained by AND merging and OR merging the lists A and B are $\sigma(A,B)$ and $a+b-\sigma(A,B)$, respectively. Let $\overline{A}$ denote the complement of the index term A. The list corresponding to $\overline{A} \cdot B$ is obtained by merging the lists A and B and selecting from the merged list those entries that are in list B but not in list A. We call this merging process an AND NOT merge. Clearly, the length of the list obtained by AND NOT merging A and B is equal to $b-\sigma(A,B)$. We shall not be concerned with Boolean expressions containing terms of the forms $\overline{A}+B$ and $\overline{A} \cdot \overline{B}$. This restriction leads to no loss of generality since Boolean expressions of these forms are explicitly ruled out in most installations.

Consider a query written as a Boolean expression of the index terms $A_1, A_2, \ldots, A_n$. Let $Q(A_1, A_2, \ldots, A_n)$ denote the truth function determined by this expression.\(^\dagger\) (A truth function is usually represented by a truth table for the Boolean expression.) We note that the truth function $Q(A_1, A_2, \ldots, A_n)$

\(^\dagger\)More precisely, a Boolean expression of the index terms $A_1, A_2, \ldots, A_n$ specifies an element in the free Boolean algebra with n generators. This element in the free Boolean algebra in turn specifies the truth function $Q(A_1, A_2, \ldots, A_n)$. 
determines a unique list of pointers. This list is the valid response to all queries written as Boolean expressions that determine this truth function and this list can be obtained by merging the lists \( A_1, A_2, \ldots, A_n \). Let \( T(F(A_1, A_2, \ldots, A_n)) \) denote a tree specifying the merge order of lists \( A_1, A_2, \ldots, A_n \) corresponding to the Boolean expression \( F(A_1, A_2, \ldots, A_n) \). Again, the leaves of the tree are labeled with the names of the corresponding lists while the internal nodes are marked by the Boolean operators corresponding to the merges. (Examples of such trees are shown in Fig. 1-1.) We say that the tree \( T(F(A_1, A_2, \ldots, A_n)) \) realizes the truth function \( Q(A_1, A_2, \ldots, A_n) \) if \( Q(A_1, A_2, \ldots, A_n) \) is the truth function determined by \( F(A_1, A_2, \ldots, A_n) \). Since there are many ways to parenthesize a Boolean expression (e.g., \( A \cdot (B + (C + D)) \) and \( A \cdot ((B + C) + D) \) are two different ways to parenthesize the expression \( A \cdot (B + C + D) \)), corresponding to a Boolean expression, there are many different binary merge trees. We distinguish them by using different subscripts. (For example, \( T_1(A \cdot (B + C + D)) \) and \( T_2(A \cdot (B + C + D)) \) denote two different binary merge trees for the expression \( A \cdot (B + C + D) \)). When there is no possible confusion, we also refer to the tree \( T(F(A_1, A_2, \ldots, A_n)) \) simply as \( T \).

The time required to merge two lists is proportional to the sum of their lengths for all three types of merges. (To be specific, let the proportional constant be 1.) The cost of the tree \( T(F(A_1, A_2, \ldots, A_n)) \), denoted \( C(T) \), is equal to the total merge time of the lists \( A_1, A_2, \ldots, A_n \) when the order of the merges is specified by \( T \). A tree is said to be optimal when its cost is minimum among all trees which realize the truth function \( Q(A_1, A_2, \ldots, A_n) \) determined by the Boolean expression \( F(A_1, A_2, \ldots, A_n) \) in the query. With a slight abuse of the notation, we denote the optimal tree by \( T_0(F(A_1, A_2, \ldots, A_n)) \). Hence the problem of finding a Boolean expression corresponding to the minimum total merge time is the same as that of finding
an optimal merge tree among all trees that realize the truth function $Q(A_1, A_2, \ldots, A_n)$. 
III. Algorithms for Determining Optimal Merge Trees

In this section, we assume that the lengths of overlap between different lists are very small compared with the lengths of the lists. Hence, in computation of total merge time, lengths of overlaps can be neglected. In this case, the length of the resultant list obtained by OR merging list A and B is equal to a+b. The length of the resultant list obtained by AND merging A and B is very small compared to a or b. In the computation of merge time, it is assumed to be zero. The length of the list obtained by AND NOT merging A and B is approximately equal to b.

The lengths of overlaps between lists being negligibly small, an optimal tree for OR merging the lists \( A_1,A_2,...,A_n \) to realize the truth function determined by the Boolean expression \( A_1+A_2+...+A_n \) is one with minimum weighted path length where the weights of the leaves are the lengths of the lists and the weights of internal nodes are zero. Such a tree can be constructed using the Huffman's procedure. We call the resultant tree a Huffman tree for \( A_1,A_2,...,A_n \) and denote it by \( T_0(A_1+A_2+...+A_n) \) [9].

A property of an optimal tree is that all its subtrees are optimal. It follows that if \( T_{10} \) and \( T_{20} \) are two subtrees of an optimal tree \( T_0(A_1+A_2+...+A_n) \) obtained by removing the root of \( T_0 \), then \( C(T_{10}) + C(T_{20}) \) is minimum among all possible two subtrees of an arbitrary tree \( T(A_1+A_2+...+A_n) \). Similarly, let \( T_{10},T_{20},...,T_{m0} \) denote m subtrees of \( T_0(A_1+A_2+...+A_n) \) obtained by removing the roots of larger subtrees of \( T_0 \). Then

\[
C(T_{10}) + C(T_{20}) + ... + C(T_{m0}) \leq C(T_1) + C(T_2) + ... + C(T_m)
\]  

(3-1)
where $T_1, T_2, \ldots, T_m$ are $m$ subtrees of an arbitrary tree $T(A_1 + A_2 + \ldots + A_n)$. We call the subtrees $T_{10}, T_{20}, \ldots, T_{m0}$ Huffman subtrees.

Optimal Merge Trees for Boolean Expressions of the Form $(A_1 + A_2 + \ldots + A_n) \cdot B$

Let $A_1, A_2, \ldots, A_n$ and $B$ be $n+1$ lists with lengths $a_1, a_2, \ldots, a_n$ and $b$, respectively. To find an algorithm which yields an optimal form of the Boolean expression $(A_1 + A_2 + \ldots + A_n) \cdot B$, we consider the tree shown in Fig. 3-1 where $T_1, T_2, \ldots, T_m$ are $m$ subtrees of $T(A_1 + A_2 + \ldots + A_n)$. We note that

Lemma 3-1. If the tree in Fig. 3-1 is an optimal merge tree among all trees that realize the truth function determined by the expression $(A_1 + A_2 + \ldots + A_n) \cdot B$, then $T_1, T_2, \ldots, T_m$ are Huffman subtrees of $T_0(A_1 + A_2 + \ldots + A_n)$.

Proof: The cost of the tree in Fig. 3-1 is

$$ C(T) = C(T_1) + C(T_2) + \ldots + C(T_m) + \sum_{i=1}^{n} a_i + mb $$

Because of (3-1), this cost is minimized when $T_1, T_2, \ldots, T_m$ are Huffman subtrees.

As a consequence of Lemma 3-1, to find an optimal tree which realizes the truth function determined by the Boolean expression $(A_1 + A_2 + \ldots + A_n) \cdot B$, we need to consider only trees corresponding to expressions of the form

$$ (\sum_{A_1 \in S_1} A_1) \cdot B + (\sum_{A_1 \in S_2} A_1) \cdot B + \ldots + (\sum_{A_1 \in S_m} A_1) \cdot B $$

(3-2)

where $S_j$ ($1 \leq j \leq m$) is the set of leaves of the $j$th Huffman subtree of $T_0(A_1 + A_2 + \ldots + A_n)$. Furthermore, an optimal merge tree that realizes the truth function determined by $(A_1 + A_2 + \ldots + A_n) \cdot B$ can be obtained from
Figure 3-1
Theorem 3-1

\[(A_1+A_2+\ldots+A_n) \cdot B\] is an optimal Boolean form if and only if

\[\sum_{i=1}^{n} a_i \leq b\]

Hence an optimal merge tree for \((A_1+A_2+\ldots+A_n) \cdot B\) is the tree, \(T_u\), shown in Fig. 3-2a, where \(T_0(A_1+A_2+\ldots+A_n)\) is the Huffman tree for the lists \(A_1,A_2,\ldots,A_n\).

Proof: Let \(T_{10}\) and \(T_{20}\) be two Huffman subtrees of \(T_0(A_1+A_2+\ldots+A_n)\) and \(S\) be a subset of \(\{A_1,A_2,\ldots,A_n\}\). Because of Lemma 3-1, the tree in Fig. 3-2b, \(T_d\), will have the minimum cost among all trees \(T(F(B,A_1,A_2,\ldots,A_n))\) corresponding to Boolean expressions of the form

\[F(B,A_1,A_2,\ldots,A_n) = (\Sigma A_i) \cdot B + (\Sigma_{A_i \in S} A_i) \cdot B + (\Sigma_{A_i \notin S} A_i) \cdot B\]

The cost of \(T_d\) is

\[C_d = C(T_{10}) + C(T_{20}) + \sum_{i=1}^{n} a_i + 2b\]

\[= C(T_0) + 2b\]

while the cost of \(T_u\) is

\[C_u = C(T_0) + \sum_{i=1}^{n} a_i + b\]

Therefore, we have

\[C_d \geq C_u\]

if and only if

\[\sum_{i=1}^{n} a_i \leq b\]
Figure 3-2
We now show that the inequality in (3-3) implies that the tree $T_u$ in Fig. 3-2a is indeed optimal. Again, because of Lemma 3-1, it suffices for us to show that (3-2) implies that the cost of the tree $T_m$ shown in Fig. 3-3a, $C_m$, is less than the cost of the tree $T_{(m+1)}$ shown in Fig. 3-3b, $C_{m+1}$, where $T_{j_{10}}$ and $T_{j_{20}}$ are two Huffman subtrees of $T_{j_{0}}$. We note that

$$C_{m+1} = \sum_{k=1}^{m} C(T_{k0}) + C(T_{j_{10}}) + C(T_{j_{20}}) + \sum_{i=1}^{n} a_i + (m+1) b$$

and

$$C_m = \sum_{k=1}^{m} C(T_{k0}) + \sum_{i=1}^{n} a_i + mb$$

But

$$C(T_{j_{0}}) = C(T_{j_{10}}) + C(T_{j_{20}}) + \sum_{A_i \in S_j} a_i$$

where $S_j$ is the set of leaves of $T_{j_{0}}$. That

$$C_{m+1} - C_m = b - \sum_{A_i \in S_j} a_i$$

is equal to or larger than zero is clearly implied by the inequality (3-3).

From Theorem 3-1, we have algorithm A for finding an optimal merge tree for $(A_1, A_2, \ldots, A_n) \cdot B$.

**Algorithm A**

1. a. If $b \geq \sum_{i=1}^{n} a_i$, $(A_1, A_2, \ldots, A_n) \cdot B$ is an optimal Boolean form and an optimal merge tree is $T_u$ shown in Fig. 3-2a where $T_0(A_1, A_2, \ldots, A_n)$ is the Huffman tree for the lists $A_1, A_2, \ldots, A_n$. 
Figure 3-3
b. If \( b < \sum_{i=1}^{n} a_i \), choose the merge tree \( T_d \) shown in Fig. 3-2b where \( T_{10} \) and \( T_{20} \) are two Huffman subtrees of \( T_0(A_1+A_2+...+A_n) \). The corresponding Boolean expression is

\[
( \sum_{A_1} A_1 ) \cdot B + ( \sum_{A_1} A_1 ) \cdot B
\]

where \( S_1 \) and \( S_2 \) are the sets of leaves of \( T_{10} \) and \( T_{20} \).

2. An optimal tree can be obtained by repeating step 1 for each of the terms \( (\sum_{A_1} A_1 ) \cdot B \).

Consider the example shown in Fig. 3-4. The lengths of the lists \( B, A_1, A_2, A_3 \) and \( A_4 \) are 5, 1, 2, 5 and 10, respectively. The Huffman tree for \( A_1+A_2+A_3+A_4 \) is shown in Fig. 3-4a together with its two subtrees \( T_{10} \) and \( T_{20} \). Since \( (a_1+a_2+a_3+a_4) = 18 > b \), the cost of the tree \( T_d \) in Fig. 3-4b is less than the cost of any tree \( T(B \cdot (A_1+A_2+A_3+A_4)) \). (\( C(T_d) = 39 \) and 
\[
C(T(B \cdot (A_1+A_2+A_3+A_4))) \geq 52.
\]) Hence, we choose the Boolean expression corresponding to the tree \( T_d \):

\[
B \cdot (A_1+A_2+A_3) + B \cdot A_4
\]

instead of \( B \cdot (A_1+A_2+A_3+A_4) \). Repeating step 1 for the term \( B \cdot (A_1+A_2+A_3) \), we choose to distribute \( \cdot \) operation with respect to \( + \) operation and obtain

\[
B \cdot (A_1+A_2) + B \cdot A_3 + B \cdot A_4
\]

The corresponding merge tree is \( T'_d \) shown in Fig. 3-4c. Since \( a_1 + a_2 < b \), we conclude that \( T'_d \) is an optimal merge tree among all trees that realize the truth function determined by the expression \( B \cdot (A_1+A_2+A_3+A_4) \). Indeed, 
\[
C(T'_d) = 36 \text{ and } C(T(B \cdot A_1+B \cdot A_2+B \cdot A_3+B \cdot A_4)) = 38.
\]
$T_0(A_1+A_2+A_3+A_4)$

(a)

$T_10$

(b)

Boolean expression $B \cdot (A_1+A_2+A_3+A_4)$

(c)

Figure 3-4
Optimal Merge Trees for Boolean Expressions of the Form \((A_1 + A_2 + ... + A_n) \cdot (B_1 + B_2 + ... + B_m)\)

Lemma 3-1 and Theorem 3-1 can be generalized to the case when the Boolean expression specified in the query is of the form

\[(A_1 + A_2 + ... + A_n) \cdot (B_1 + B_2 + ... + B_m)\]

To do so, let us consider the tree \(T_k\) shown in Fig. 3-5 where \(T_{A1}, T_{A2}, ..., T_{Aj}\) are subtrees of \(T(A_1 + A_2 + ... + A_n)\) and \(T_{B1}, T_{B2}, ..., T_{Bk}\) are subtrees of \(T(B_1 + B_2 + ... + B_m)\). We state without proof:

**Lemma 3-2.** If the merge tree \(T_k\) is optimal for the Boolean expression \((A_1 + A_2 + ... + A_n) \cdot (B_1 + B_2 + ... + B_m)\), then \(T_{A1}, T_{A2}, ..., T_{Aj}\) are Huffman subtrees of \(T(A_1 + A_2 + ... + A_n)\) and \(T_{B1}, T_{B2}, ..., T_{Bk}\) are Huffman subtrees of \(T(B_1 + B_2 + ... + B_m)\).

In this case, we have

**Theorem 3-2.**

\[(A_1 + A_2 + ... + A_n) \cdot (B_1 + B_2 + ... + B_m)\] is an optimal Boolean expression if and only if

\[\sum_{i=1}^{n} a_i = \sum_{i=1}^{m} b_i\]

Hence an optimal merge tree for \((A_1 + A_2 + ... + A_n) \cdot (B_1 + B_2 + ... + B_m)\) is the tree \(T_u\) in Fig. 3-6 where \(T_A\) and \(T_B\) are Huffman trees for \(A_1, A_2, ..., A_n\) and \(B_1, B_2, ..., B_m\), respectively.

**Proof:** We compare the cost of the tree \(T_u\) in Fig. 3-6a with that of \(T_{dA}\) and \(T_{dB}\) in Fig. 3-6b and 3-6c. The Boolean expressions corresponding to the tree \(T_{dA}\) and \(T_{dB}\) are
Figure 3-5
Figure 3-6
\[
( \sum A_i \cdot (B_1 + B_2 + \ldots + B_m) + ( \sum A_i \cdot (B_1 + B_2 + \ldots + B_m) )
\]

\[
\quad A_i \in S_{1A} \quad A_i \notin S_{1A}
\]

and

\[
( \sum B_i \cdot (A_1 + A_2 + \ldots + A_n) + ( \sum B_i \cdot (A_1 + A_2 + \ldots + A_n) )
\]

\[
\quad B_i \in S_{1B} \quad B_i \notin S_{1B}
\]

respectively, where \( S_{1A} \) and \( S_{1B} \) are sets of leaves of \( T_{A1} \) and \( T_{B1} \), respectively. Let \( C_u \) and \( C_{dA} \) be the costs of the trees \( T_u \) and \( T_{dA} \), respectively.

\[
C_u = C(T_A) + C(T_B) + \sum_{i=1}^{n} \sum_{i=1}^{m} a_i + b_i
\]

and

\[
C_{dA} = C(T_{A1}) + C(T_{A2}) + C(T_B) + 2 \sum_{i=1}^{n} \sum_{i=1}^{m} b_i + \sum_{i=1}^{n} a_i
\]

But

\[
C(T_A) = C(T_{A1}) + C(T_{A2}) + \sum_{i=1}^{n} a_i
\]

Hence

\[
C_u - C_{dA} = \sum_{i=1}^{n} a_i - \sum_{i=1}^{m} b_i
\]

which is less than or equal to zero if and only if

\[
\sum_{i=1}^{n} a_i \leq \sum_{i=1}^{m} b_i
\]

Similarly,

\[
\sum_{i=1}^{m} b_i \leq \sum_{i=1}^{n} a_i
\]
implies

\[ C_u - C_{dB} \leq 0 \]

where \( C_{dB} \) is the cost of the tree \( T_{dB} \). In other words,

\[ C_u = C_{dB} = C_{dA} \]

when

\[
\sum_{i=1}^{n} a_i = \sum_{i=1}^{m} b_i
\]  

(3-4)

We need to show that because of Eq. (3-4), \( C_u \) is no greater than any other tree which realizes the truth function determined by the Boolean expression \((A_1 + A_2 + \ldots + A_n) \cdot (B_1 + \ldots + B_m)\). Let \( T_k \) shown in Fig. 3-5 be such a tree. Again, because of Lemma 3-2, \( T_{A1}, T_{A2}, \ldots, T_{Aj} \) are \( j \) Huffman subtrees of \( T_A \) and \( T_{B1}, T_{B2}, \ldots, T_{Bk} \) are \( k \) Huffman subtrees of \( T_B \). Let \( S_{Ap} \) be the set of leaves of \( T_{Ap} (p = 1, 2, \ldots, j) \) and \( S_{Bq} \) be the set of leaves of \( T_{Bq} (q = 1, 2, \ldots, k) \). We note that the Boolean expression corresponding to the tree \( T_k \) is

\[
\sum_{p=1}^{j} \sum_{q=1}^{k} (\sum_{A_i \in S_{Ap}} A_i \cdot (\sum_{B_i \in S_{Bq}} B_i))
\]  

(3-5)

Let \( T_{Bq1} \) and \( T_{Bq2} \) be two Huffman subtrees of \( T_{Bq} \) with \( S_{Bq1} \) and \( S_{Bq2} \) be the sets of leaves, respectively. Let \( T_{k+1}(l; r) \) denote a tree corresponding to the Boolean expression

\[
\sum_{p=1}^{j} \sum_{q=2}^{k} (\sum_{A_i \in S_{Ap}} A_i \cdot (\sum_{B_i \in S_{Bq}} B_i)) + \sum_{p=r+1}^{j} (\sum_{A_i \in S_{Ap}} A_i \cdot (\sum_{B_i \in S_{Bq}} B_i))
\]

\[
+ \sum_{p=1}^{r} (\sum_{A_i \in S_{Ap}} A_i \cdot (\sum_{B_i \in S_{Bl1}} B_i)) + \sum_{p=1}^{r} (\sum_{A_i \in S_{Ap}} A_i \cdot (\sum_{B_i \in S_{Bl2}} B_i))
\]
obtained by further distributing the operation with respect to + operation in the sum term \( \sum_{B_i \in S_{B1}} B_i \) in (3-5). We note that for any \( 1 \leq r < j \),

\[
C(T_{k+1}(l; r)) > C(T_k)
\]

This inequality follows from

\[
C(T_k) = \sum_{p=1}^{j} C(T_{Ap}) + \sum_{q=1}^{k} C(T_{Bq}) + k \sum_{i=1}^{n} a_i + j \sum_{i=1}^{n} b_i
\]

while

\[
C(T_{k+1}(l; r)) = \sum_{p=1}^{j} C(T_{Ap}) + \sum_{q=1}^{k} C(T_{Bq}) + k \sum_{i=1}^{n} a_i + j \sum_{i=1}^{n} b_i
\]

\[+ \sum_{r} \sum_{p=1}^{j} A_i \in S_{Ap} \]

When \( r=j \), we have

\[
C(T_{k+1}(l; j)) = \sum_{p=1}^{j} C(T_{Ap}) + \sum_{q=2}^{k} C(T_{Bq})
\]

\[+ C(T_{B11}) + C(T_{B12}) + (k+1) \sum_{i=1}^{n} a_i + j \sum_{i=1}^{m} b_i\]

But

\[
C(T_{B1}) = C(T_{B11}) + C(T_{B12}) + \sum_{B_i \in S_{B1}} b_i
\]

Hence

\[
C(T_{k+1}(l; j)) - C(T_k) = \sum_{i=1}^{n} a_i - \sum_{B_i \in S_{B1}} b_i \quad (3-6)
\]
which is larger than or equal to zero when Eq. (3-4) is valid. In general, let \( T_{k+1}(t; j) \) be the tree corresponding to the Boolean expression

\[
\sum_{p=1}^{j} \left( \sum_{q=t+1}^{k} A_i \in S_{Ap} \left( \sum_{B_i \in S_{Bj}} B_i \right) + \sum_{q=1}^{t} \left[ \sum_{A_i \in S_{Ap}} A_i \right] \left( \sum_{B_i \in S_{Bq}} B_i \right) \right)
\]

for \( 1 \leq t \leq k \) and its cost \( C(T_{k+1}(t; j)) \). We have,

\[
C(T_{k+1}(t; j)) = C(T_k) = t \sum_{i=1}^{n} a_i - \sum_{q=1}^{t} \sum_{i \in B_i \in S_{Bq}} b_i
\]

which is equal to or larger than zero when Eq. (3-4) is valid.

When \( \sum_{i=1}^{n} a_i < \sum_{j=1}^{m} b_j \), the tree \( T_u \) is no longer optimal. In this case, let \( S_{Bj}, 1 < j \leq k \) be the set of leaves of \( k \) Huffman subtrees of \( T_B \) such that their union is \( \{ B_1, B_2, \ldots, B_m \} \). Moreover,

\[
\sum_{B_i \in S_{Bj}} b_i \leq \sum_{i=1}^{n} a_i, \quad j = 1, 2, \ldots, k
\]

but

\[
\sum_{B_i \in S_{Bj}} b_i > \sum_{i=1}^{n} a_i, \quad j = 1, 2, \ldots, k \text{ and } j \neq j'.
\]

It follows from Eq. (3-6) that
Corollary 3-1

When $\sum_{i=1}^{n} a_i < \sum_{j=1}^{m} b_j$, an optimal tree corresponding to the Boolean expression

$$\sum_{j=1}^{k} (A_1 + A_2 + \ldots + A_n) \cdot \left( \sum_{B_i \in S_B} B_i \right)$$

has the minimum cost among all trees that realize the truth function determined by $(A_1 + A_2 + \ldots + A_n) \cdot (B_1 + B_2 + \ldots + B_m)$.

Similarly, for the case of $\sum_{j=1}^{m} b_j < \sum_{i=1}^{n} a_i$, let $S_{A_j}$, $1 < j \leq k$, be the set of leaves of $k$ Huffman subtrees of $T_A$ such that their union is $(A_1, A_2, \ldots, A_n)$, and

$$\sum_{A_i \in S_{A_j}} a_i \leq \sum_{j=1}^{m} b_j \quad j = 1, 2, \ldots, k$$

but

$$\sum_{A_i \in S_{A_j} \cup S_{A_{j'}}} a_i > \sum_{j=1}^{m} b_j \quad j, j' = 1, 2, \ldots, k \text{ and } j \neq j'$$

We have

Corollary 3-2

When $\sum_{i=1}^{n} a_i > \sum_{j=1}^{m} b_j$, an optimal tree corresponding to the Boolean expression

$$\sum_{j=1}^{k} (B_1 + B_2 + \ldots + B_m) \cdot \left( \sum_{A_i \in S_{A_j}} A_i \right)$$
has the minimum cost among all trees that realize the truth function
determined by \((A_1 + A_2 + \ldots + A_n) \cdot (B_1 + B_2 + \ldots + B_m)\).

An algorithm to determine an optimal tree in this case is

**Algorithm B**

1. If \(\sum_{i=1}^{n} a_i = \sum_{i=1}^{m} b_i\), \((A_1 + A_2 + \ldots + A_n) \cdot (B_1 + B_2 + \ldots + B_m)\) is an
   optimal expression. An optimal tree is \(T_u\) shown in Fig. 3-6a where \(T_A\) and
   \(T_B\) are Huffman trees of \(A_1, A_2, \ldots, A_n\) and \(B_1, B_2, \ldots, B_m\), respectively.

2. If \(\sum_{i=1}^{n} a_i < \sum_{i=1}^{m} b_i\),
   
   a. Choose the Boolean expression
   
   \[
   (A_1 + A_2 + \ldots + A_n) \cdot \left( \sum_{B_i \in S_{B_1}} B_i \right) + (A_1 + A_2 + \ldots + A_n) \cdot \left( \sum_{B_i \in S_{B_2}} B_i \right)
   \]
   
   where \(S_{B_1}\) and \(S_{B_2}\) are sets of leaves of the two Huffman subtrees of \(T_B\).
   
   The corresponding merge tree is \(T_{dB}\) in Fig. 3.6c.
   
   b. For each of the terms \((A_1 + A_2 + \ldots + A_n) \cdot \left( \sum_{B_i \in S_{B_j}} B_i \right)\), if
      \[
      \sum_{i=1}^{n} a_i < \sum_{i=1}^{m} b_i,
      \]
      distribute \(\cdot\) with respect to the sum of the \(B_i\)'s such that
      \[
      B_i \in S_{B_j},
      \]
      the \(B_i\)'s in each of the sum terms are elements of sets of leaves of smaller
      Huffman subtrees.
   
   c. The process in step 2-b terminates either when
      \[
      \sum_{i=1}^{n} a_i \geq \sum_{i=1}^{m} b_i
      \]
      or when we obtain terms of the form \((\sum_{i=1}^{n} A_i) \cdot B_j\).
3. If \( \sum_{i=1}^{n} a_i > \sum_{i=1}^{m} b_i \),
   
   a. Choose the Boolean expression
   \[
   (B_1 + B_2 + \ldots + B_m) \cdot (\sum_{A_1 \in S_{A1}} A_1) + (B_1 + B_2 + \ldots + B_m) \cdot (\sum_{A_1 \in S_{A2}} A_1)
   \]
   where \( S_{A1} \) and \( S_{A2} \) are the sets of leaves of the Huffman subtrees of \( T_A \).
   The corresponding merge tree is \( T_{dA} \) in Fig. 3.6b.

   b. For each of the terms \((B_1 + B_2 + \ldots + B_m) \cdot (\sum_{A_1 \in S_{A1}} A_1)\), if
   \[
   \sum_{i=1}^{m} b_i < \sum_{A_1 \in S_{A1}} a_i
   \]
   distribute \( \cdot \) with respect to sum of \( A_i \)'s in each of the sum terms such that the \( A_i \)'s in each of the sum terms are elements of leaves of smaller Huffman subtrees.

   (c) The process in 3b terminates either \( \sum_{i=1}^{m} b_i > \sum_{A_1 \in S_{A1}} a_i \) or
   \[
   \sum_{i=1}^{m} b_i < \sum_{A_1 \in S_{A1}} a_i
   \]
   when we obtain terms of the form \((\sum_{i=1}^{m} B_i) A_j\).

   We illustrate Algorithm B by an example. Consider the expression
   \[
   (A_1 + A_2 + A_3 + A_4) \cdot (B_1 + B_2).
   \]
   The lengths of the corresponding lists are
   \( a_1 = 1, a_2 = 2, a_3 = 3, a_4 = 6, b_1 = 2 \) and \( b_2 = 2 \). The Huffman trees for
   the lists \( A_1, A_2, A_3 \) and \( A_4 \) and for lists \( B_1 \) and \( B_2 \) are shown in Fig. 3-7a.
   Since \( a_1 + a_2 + a_3 + a_4 > b_1 + b_2 \), the cost of the merge tree \( T_1 \) in Fig. 3-7b
   corresponding to the Boolean expression
   \[
   F_1 = (B_1 + B_2) \cdot (A_1 + A_2 + A_3) + (B_1 + B_2) \cdot A_4
   \]
   is less than the cost of the tree \( T((A_1 + A_2 + A_3 + A_4) \cdot (B_1 + B_2)) \). Moreover, since
   \( a_1 + a_2 + a_3 > b_1 + b_2 \), the cost of the tree \( T_2 \) in Fig. 3-7c is less than the cost
   of \( T_1 \). Indeed, \( T_2 \) is an optimal tree and the Boolean expression corresponding
   to \( T_2 \) is
(a) $T_A$

(b) $T_1$

(c) $T_2$

Figure 3-7
\[
F_2 = (B_1 + B_2) \cdot (A_1 + A_2) + (B_1 + B_2) \cdot A_3 + (B_1 + B_2) \cdot A_4 \\
(C(T(A_1 + A_2 + A_3 + A_4) \cdot (B_1 + B_2))) \geq 41, C(T_1) = 33, C(T_2) = 31.
\]

**Further Generalizations**

To find an optimal merge tree when the Boolean expression specified in the query is a product of sum terms

\[
P_m = \prod_{j=1}^{m} (A_{j1} + A_{j2} + \ldots + A_{jn_j})
\]

and \(A_{j1}\) are all distinct, let \(C(k)\) denote the cost of the merge tree \(T(P_m)\).

Suppose that we first complete all those merges corresponding to the Boolean expression

\[
P_{m-1} = \prod_{j=1}^{m-1} (A_{j1} + A_{j2} + \ldots + A_{jn_j})
\]

Since the lengths of overlaps between lists are assumed to be negligibly small, it follows from Theorem 3-1 that the cost

\[
C(m) = \sum_{i=1}^{n_m} a_{mi} + C(m-1)
\]

is minimum in this case and the corresponding Boolean expression is

\[
P_m = A_{m1} \cdot P_{m-1} + A_{m2} \cdot P_{m-1} + \ldots + A_{mn_m} \cdot P_{m-1}
\]

Similarly, we have

\[
C(m) = \sum_{j=1}^{m} \sum_{i=1}^{n_j} a_{ji} + C(T((A_{11} + A_{12} + \ldots + A_{1n_1}) \cdot (A_{21} + A_{22} + \ldots + A_{2n_2})))
\]

Suppose that the indices are chosen so that
\[
C(T((A_{11} + A_{12} + \ldots + A_{1n_1}) \cdot (A_{21} + A_{22} + \ldots + A_{2n_2}))) = \sum_{i=1}^{n_1} a_i \quad \sum_{i=1}^{n_2} a_i 
\]

\[
\leq C(T((A_{j1} + A_{j2} + \ldots + A_{jn_j}) \cdot (A_{k1} + A_{k2} + \ldots + A_{kn_k}))) = \sum_{i=1}^{n_j} a_i - \sum_{i=1}^{n_k} a_i
\]

for all \( j, k = 3, 4, \ldots, m \).

**Corollary 3-3**

If \( \sum_{j=1}^{n_j} a_{ji} \) are equal for all \( j = 1, 2, \ldots, m \), the cost of the optimal tree corresponding to the Boolean expression in (3-7) is

\[
C_0(m) = \sum_{j=1}^{m} \sum_{i=1}^{n_j} a_{ji} + C(T(A_{11} + A_{12} + \ldots + A_{1n_1})) + C(T(A_{21} + A_{22} + \ldots + A_{2n_2}))
\]

Furthermore, when \( \sum_{i=1}^{n_j} a_{ji} \) are not all equal, the cost of the optimal tree is given by

**Corollary 3-4**

\[
C_0(m) = \sum_{j=1}^{m} \sum_{i=1}^{n_j} a_{ji} + C(T((A_{11} + A_{12} + \ldots + A_{1n_1}) \cdot (A_{21} + A_{22} + \ldots + A_{2n_2}))) \quad (3-8)
\]

When the Boolean expression given in the query can be written as a sum of product terms

\[
S_m = \sum_{j=1}^{m} A_{j1} \cdot A_{j2} \ldots \cdot A_{jn_j}
\]

with all \( A_{ji} \) being distinct, we note that distribution of the \( + \) operation with respect to the \( \cdot \) operation will lead to increase in the total merge time. Hence, the optimal cost in this case is
Optimal Merge Trees for Boolean Expressions Containing Complements of Variables

To generalize the results discussed above to Boolean expressions containing complements of variables, we consider expressions of the form

\[(A_1 + A_2 + \ldots + A_n) \cdot \overline{B}\]  

(3-9)

In this case, we have

**Theorem 3-3**

\[(A_1 + A_2 + \ldots + A_n) \cdot \overline{B}\] is an optimal expression among all expressions equivalent to it. Hence an optimal merge tree is \(T_u\) shown in Fig. 3-8 where the symbol \(-\) is used to denote an AND NOT merge.

**Proof:** The cost of the merge tree corresponding to the Boolean expression in (3-9) is

\[C_u = C(T_A) + \sum_{i=1}^{n} a_i + b\]

where \(T_A\) is a Huffman tree for the lists \(A_1, A_2, \ldots, A_n\). Let \(T_{A1}\) and \(T_{A2}\) be two subtrees of \(T(A_1 + A_2 + \ldots + A_n)\) with sets of leaves \(S_{A1}\) and \(S_{A2}\) respectively. The cost of the merge tree corresponding to the Boolean expression

\[\left( \sum_{A_i \in S_{A1}} A_i \right) \cdot \overline{B} + \left( \sum_{A_i \in S_{A2}} A_i \right) \cdot \overline{B}\]

is
\[ T_0(A_1 + A_2 + \ldots + A_n) \]

\[ T_u \]

Fig. 3-8
\[ C_d = C(T_{A1}) + C(T_{A2}) + \sum_{i=1}^{n} a_i + 2b + \sum_{i=1}^{n} a_i \]

\[ \geq C(T_A) + \sum_{i=1}^{n} a_i + 2b \]

\[ = C_u + b \]

Similarly, let \( T_{Al} \) for \( i = 1, 2, \ldots, m \) be \( m \) subtrees of \( T_A \) and \( S_{Ai} \) be the sets of their leaves,

\[ C_{dm} = C(T_A) + mb + \sum_{i=1}^{n} a_i \]

\[ > C_u \]

We further observe that the cost of the merge tree corresponding to the Boolean expression

\[ B \cdot \underbrace{A_1 + A_2 + \ldots + A_n}_{=} \]

is equal to that of \( B \cdot (A_1 + A_2 + \ldots + A_n) \). Again, let \( T_{Al} \) and \( T_{A2} \) be two Huffman subtrees of \( T_A(A_1 + A_2 + \ldots + A_n) \) with their sets of leaves being \( S_{Al} \) and \( S_{A2} \), respectively. The cost of the merge tree corresponding to the Boolean expression \( B \cdot (A_1 + A_2 + \ldots + A_n) \) is equal to

\[ C(T_{A1}) + C(T_{A2}) + \sum_{i=1}^{n} a_i + 2b \]

\[ \uparrow \text{We note that in response to a query of the form } B \cdot \overline{F_1} \cdot \overline{F_2}, \text{ the system will parenthesis the expression as } (B \cdot \overline{F_1}) \cdot \overline{F_2}, \text{ or rewrite it as } B \cdot \overline{F_1 + F_2}. \]
same as the cost of the tree corresponding to
\[ B \cdot (\sum_{A_1 \in S_{A_1}} A_1) + B \cdot (\sum_{A_1 \in S_{A_2}} A_1) \]

Hence we can use the result in Theorem 3-1 and obtain

**Corollary 3-5**

When \( b \geq \sum_{i=1}^{n} a_i \), \( B \cdot A_1 + A_2 + ... + A_n \) is optimal among all Boolean expressions equivalent to it.

Algorithm A can be modified to determine an optimal equivalent form when \( b < \sum_{i=1}^{n} a_i \).

**Algorithm A1**

1. If \( b \geq \sum_{i=1}^{n} a_i \), \( B \cdot A_1 + A_2 + ... + A_n \) is optimal among all Boolean expressions equivalent to it and an optimal merge tree is \( T_u \) as shown in Fig. 3-9a.

Otherwise, we have

2. a. \( b < \sum_{i=1}^{n} a_i \). Choose the Boolean expression

\[
B \cdot \sum_{A_1 \in S_{A_1}} A_1 \cdot \sum_{A_1 \in S_{A_2}} A_1
\]

and the corresponding tree is \( T_d \) shown in Fig. 3-9b.

b. An optimal tree can be obtained by repeating steps 1 and/or 2a for \( B \cdot \sum_{A_1 \in S_{A_1}} A_1 \) and then for \( (B \cdot \sum_{A_1 \in S_{A_1}} A_1) \cdot \sum_{A_1 \in S_{A_2}} A_1 \).
Figure 3-9
Similarly, from Theorem 3-2, we have

**Corollary 3-6**

When \( \sum_{i=1}^{m} b_i = \sum_{i=1}^{n} a_i \), an optimal form of the Boolean expression

\[
(B_1 + B_2 + \ldots + B_m) \cdot \overline{A_1 + A_2 + \ldots + A_n}
\]

is

\[
(B_1 + B_2 + \ldots + B_m) \cdot (A_1 + A_2 + \ldots + A_n)
\]

The algorithm BL determines an optimal merge tree when \( \sum_{i=1}^{m} b_i \neq \sum_{i=1}^{n} a_i \).

**Algorithm BL**

1. If \( \sum_{i=1}^{n} a_i = \sum_{i=1}^{m} b_i \), the optimal merge tree is \( T_u \) in Fig. 3-10a where \( T_A \) and \( T_B \) are Huffman trees of \( A_1, A_2, \ldots, A_n \) and \( B_1, B_2, \ldots, B_m \).

Otherwise, we have

2. If \( \sum_{i=1}^{n} a_i > \sum_{i=1}^{m} b_i \)

   a. Choose the expression

   \[
   \left( \sum_{A_i \in S_{A1}} A_i \right) \cdot B_1 + B_2 + \ldots + B_m + \left( \sum_{A_i \in S_{A2}} A_i \right) \cdot B_1 + B_2 + \ldots + B_m
   \]

   The corresponding merge tree is \( T_{dA} \) in Fig. 3-10b.

   b. For each of the terms \( \left( \sum_{A_i \in S_{A1}} A_i \right) \cdot B_1 + B_2 + \ldots + B_m \), if \( \sum_{A_i \in S_{A1}} A_i > \sum_{A_i \in S_{A2}} A_i \), rewrite the term as

   \[
   \left( \sum_{A_i \in S_{A1}} A_i \right) \cdot B_1 + B_2 + \ldots + B_m + \left( \sum_{A_i \in S_{A2}} A_i \right) \cdot B_1 + B_2 + \ldots + B_m
   \]

   where \( S_{A1} \) and \( S_{A2} \) are the sets of leaves of the two Huffman subtrees of \( T_{A1} \).
Figure 3-10
c. Repeat Step 2b until either \( \sum_{i=1}^{n} a_i < \sum_{i=1}^{m} b_i \) or the term becomes
\[
A_j \cdot \overline{B_1 + B_2 + \ldots + B_m}.
\]

3. If \( \sum_{i=1}^{n} a_i < \sum_{i=1}^{m} b_i \),

a. Choose the Boolean expression
\[
(A_1 + A_2 + \ldots + A_n) \cdot \left( \sum_{B_i \in S_{B_1}} B_i \right) \cdot \left( \sum_{B_i \in S_{B_2}} B_i \right)
\]
and the corresponding merge tree is \( T_{dB} \) as shown in Fig. 3-10c.

b. For each of the terms \( \sum_{B_i \in S_{B_j}} B_i \), if \( \sum_{i=1}^{n} a_i < \sum_{B_i \in S_{B_j}} b_i \), rewrite the term as
\[
\left( \sum_{B_i \in S_{B_{j1}}} B_i \right) \cdot \left( \sum_{B_i \in S_{B_{j2}}} B_i \right)
\]
where \( S_{B_{j1}} \) and \( S_{B_{j2}} \) are sets of leaves of the Huffman subtrees of \( T_{B_j} \).

c. Repeat Step 3b until either \( \sum_{i=1}^{n} a_i \geq \sum_{B_i \in S_{B_j}} b_i \) or when \( S_{B_j} \) contains one single term.

Furthermore, we have

**Corollary 3-7**

The cost of an optimal merge tree corresponding to the expression
\[
\sum_{j=1}^{m} (A_{j1} + A_{j2} + \ldots + A_{j_{n_j}}) \cdot \sum_{k=1}^{\ell} (B_{k1} + B_{k2} + \ldots + B_{k_{P_k}})
\]
is
\[
C_0(m) + \sum_{k=1}^{\ell} \sum_{i=1}^{P_k} b_{k1}
\]
where \( C_0(m) \) is the cost for merging the lists \( A_{i_j} \) in Eq. (3-8).
IV. Bounds on Suboptimal Parsing Algorithms

When the lengths of overlaps cannot be neglected in the computation of merge times, the algorithms described in Section III are no longer optimal. In this section, we derive bounds on the performance of these algorithms in terms of the maximum overlap between lists. Again, let \( \sigma(A, B) \) denote the length of overlap between lists \( A \) and \( B \). The length of the resultant list obtained by OR merging lists \( A \) and \( B \) is equal to \( a + b - \sigma(A, B) \). The lengths of the resultant lists are \( \sigma(A, B) \) and \( a - \sigma(A, B) \) when the lists \( A \) and \( B \) are AND merged and AND NOT merged, respectively.

Consider a set of lists \( S = \{A_1, A_2, \ldots, A_n\} \). We say that the maximum length of overlap between them is \( \hat{\sigma} \) if \( \sigma(A_i, A_j) \leq \hat{\sigma} \) for all \( A_i \) and \( A_j \) in \( S \). Moreover, let \( S_1 \) and \( S_2 \) be any two disjoint subsets of \( S \), and \( R_1 \) and \( R_2 \) be the two lists obtained by OR merging lists in \( S_1 \) and \( S_2 \), respectively. Then, \( \sigma(R_1, R_2) \leq \hat{\sigma} \). Hence, \( \hat{\sigma} \) is an absolute measure of the maximum length of overlap. It is a meaningful measure when the lists in \( S \) are of comparable lengths and that their intersections are relatively small compared to their lengths.

In practice, the lengths of the inverted lists may differ by several orders of magnitude.\(^\dagger\) The length of overlap between any pair of lists is often measured in terms of a fraction of the length of the shorter list. Let \( \phi(A_i, A_j) \) denote the fraction such that

\[
\sigma(A_i, A_j) = \phi(A_i, A_j) \min(a_i, a_j)
\]

\(^\dagger\) For example, from MEDLAR Master Mesh, we found that the length of the list corresponding to the index term HUMAN is 493, 599 while that corresponding to LUROVIN is only 4.
for any $A_1$, $A_j$ in $S = \{A_1,A_2,\ldots,A_n\}$. Let $\hat{\sigma}$ denote the maximum overlap for the set $S$. That is, $\sigma(A_i,A_j) \leq \hat{\sigma} \min[a_i,a_j]$ and $\sigma(R_1,R_2) \leq \hat{\sigma} \min[r_1,r_2]$ where $r_1$ and $r_2$ are the lengths of $R_1$ and $R_2$, respectively. With slight abuse of the term, we also call $\hat{\sigma}$ the maximum length of overlap.

Again let $C(T)$ denote the cost of a merge tree $T(A_1+A_2+\ldots+A_n)$. Since $C(T)$ is the total merge time of $A_1,A_2,\ldots,A_n$ when their merge order is specified by $T$, $C(T)$ depends on the lengths of $A_1,A_2,\ldots,A_n$ as well as overlaps between them. Let $P(T)$ denote the weighted path length of the tree $T$. As discussed in Section III, $P(T)$ is the cost of the tree $T$ when the corresponding Boolean expression is $A_1+A_2+\ldots+A_n$ and the length of overlap between the lists are zero.

**Bounds on Cost of Huffman Tree**

Let $T_0(A_1+A_2+\ldots+A_n)$ be an optimal merge tree for the lists $A_1,A_2,\ldots,A_n$ corresponding to the Boolean expression $A_1+A_2+\ldots+A_n$. As demonstrated by the example in Fig. 4-1, $T_0$ cannot be obtained using Huffman's procedure in general. Let $T_H$ denote the Huffman tree for $A_1,A_2,\ldots,A_n$. We now bound the cost of the Huffman tree $T_H$.

**Lemma 4.1.**

Let $S = \{A_1,A_2,\ldots,A_k\}$ be a set of lists and $R_k$ be the list obtained by OR merging all the $A_i$ in $S$. The length of the list $R_k$, $r_k'$, is such that

$$r_k' \geq \sum_{i=1}^{k} a_i - (k-1) \hat{\sigma}$$

where $\hat{\sigma}$ is the maximum overlap between the lists in $S$. Moreover, the bound is tight.
\[ C(T_H) = 2(2+3) + 4 = 14 \]

(a) \( T_H \), Huffman tree

\[ C(T_0) = 3 + 4 + 4 + 2 = 13 \]

(b) \( T_0 \), Optimal tree

\[ \sigma(A_1,A_2) = 3, \quad \sigma(A_1,A_3) = 0, \quad \sigma(A_2,A_3) = 0 \]

\[ a_1 = 4, \quad a_2 = 3, \quad a_3 = 2 \]

Figure 4-1
Proof: Let us consider any list $A_1'$ in the set $S$. Without loss of
generality, suppose that $A_1 \cap A_1', A_1 \cap A_2', \ldots A_1 \cap A_k'$
(where $\cap$ denotes set intersection). By definition of $\wedge$, the total number of
elements in $A_1 \cap A_1', A_1 \cap A_2', \ldots A_1 \cap A_k'$ is at most $\wedge$.

Let

$I(A_1) = \phi$

where $\phi$ is the null set and

$I(A_i) = A_1 \cap (A_1 \cup A_2 \cup \ldots \cup A_{i-1})$ $i = 2, 3, \ldots, k$

We note that the lists

$A_1' = A_1$

$A_2' = A_2 - I(A_2)$

$\vdots$

$A_k' = A_k - I(A_k)$

are disjoint. Moreover, their lengths $a_1', a_2', \ldots, a_k'$ are such that

$a_1' = a_1$

$a_2' \geq a_2 - \wedge$

$\vdots$

$a_k' \geq a_k - \wedge$

Since the list corresponding to $A_1' + A_2' + \ldots + A_n'$ is $R_k$, we have

$r_k = \sum_{i=1}^{k} a_i' \geq \sum_{i=1}^{k} a_i - (k-1) \wedge$

We point out here that throughout our discussion, by a list, we mean a
sorted list of distinct elements. Hence, we may also regard a list as a
set whenever the order in which the elements appear in it is irrelevant.
That the bound is tight can be demonstrated by the example:

\[ A_1 = (\alpha, \beta, \gamma, x_1, \ldots, x_n), \ A_2 = (\alpha, \beta, \gamma, y_1, \ldots, y_n), \ldots \ A_k = (\alpha, \beta, \gamma, z_1, \ldots, z_k) \]

where \( x_i, y_i, z_i \) are all distinct. \( \sigma = 3 \) and the list \( R_k \) is

\[ (\alpha, \beta, \gamma, x_1, \ldots, x_n, y_1, \ldots, y_n, \ldots, z_1, \ldots, z_k) \]

with length \( k \sum \frac{\sigma}{n} - 3(k-1). \)

In terms of \( \sigma \), we have the following bound on the weighted path length of a Huffman tree for a set of lists \( S = \{A_1, A_2, \ldots, A_n\} \), \( p(T_H) \).

**Theorem 4-1**

\[ p(T_H) \leq C(T) + \frac{n^2 + n - 1}{2} \frac{n - 1}{n} \sigma \quad (4-1) \]

where \( \sigma \) is the maximum overlap for lists \( A_1, A_2, \ldots, A_n \) and \( C(T) \) is the cost of any tree \( T \) for these lists corresponding to the Boolean expression \( A_1 + A_2 + \ldots + A_n \).

**Proof:** Let \( T \) be an arbitrary tree for the set of lists \( \{A_1, A_2, \ldots, A_n\} \) corresponding to the expression \( A_1 + A_2 + \ldots + A_n \). Suppose that in \( T \) the leaf \( A_i \) is at level \( l_i, i = 1, 2, \ldots, n \). Let \( T' \) be a tree obtained from \( T \) when the weight of \( A_i \) is replaced by \( a_i - \frac{n-1}{n} \sigma \). We claim that

\[ p(T') \leq C(T) \quad (4-2) \]

That is,

\[ \sum_{i=1}^{n} l_i (a_i - \frac{n-1}{n} \sigma) \leq C(T) \]

Since a Huffman tree has the minimum weighted path length and \( \sum_{i=1}^{n} l_i \leq \frac{1}{2} (n^2 + n - 1) \), we have

\[ p(T_H) - \frac{1}{2} (n^2 + n - 1) \cdot \frac{n-1}{n} \sigma \leq C(T) \]

and hence Eq. (4-1).
To show that the inequality in (4-2) is valid, let A be an arbitrary list in the set \( \{A_1, A_2, \ldots, A_n\} \). Suppose that in the tree \( T \), A is at level \( \ell \) as shown in Fig. 4-2, where \( T_1, T_2, \ldots, T_{\ell-1}, T_{\ell} \) are subtrees of \( T \). Let \( S_1, S_2, \ldots, S_{\ell} \) be the sets of leaves and \( R_1, R_2, \ldots, R_{\ell} \) be the lists corresponding to the roots of \( T_1, T_2, \ldots, T_{\ell} \), respectively. We note that the weight of the node \( j \) in \( T', \tau_j' \), is

\[
\tau_j' = a + \sum_{k=j+1}^{\ell} \sum_{A_i \in S_k} a_i - \frac{n-1}{n} \sum_{k=j+1}^{\ell} |S_k| + 1 \tag{4-3}
\]

On the other hand, from Lemma 4-1, the length of the list \( r_{j+1}', r_{j+1} \), is such that

\[
r_{j+1} \geq \sum_{A_i \in S_{j+1}} a_i - (|S_{j+1}| - 1) \hat{\sigma}
\]

and length of the list corresponding to the node \((j+1)\) is lower bounded by

\[
a + \sum_{k=j+2}^{\ell} \sum_{A_i \in S_k} a_i - \hat{\rho} |S_k| \]

Hence merge time required to generate the list corresponding to the node \( j \) in the tree \( T, \tau_j \), is such that

\[
\tau_j \geq a + \sum_{k=j+1}^{\ell} \sum_{A_i \in S_k} a_i - \sum_{k=j+1}^{\ell} |S_k| - 1 \hat{\sigma}
\]

Since

\[
\frac{n-1}{n} \sum_{k=j+1}^{\ell} |S_k| + 1 \geq \left( \sum_{k=j+1}^{\ell} |S_k| - 1 \right) \]

\[\text{The inequality } \frac{n-1}{n}(x+1) \geq x - 1 \text{ is equivalent to } \]

\[2n \geq x + 1 \]

Since we have \( x \leq n \) and \( n > 1 \), the inequality is always valid.
Figure 4-2
for any \( n > 1 \), for all \( j = 0, 1, \ldots, \ell-1 \), \( r_j \leq r_j^\prime \) and the inequality in (4-2) follows.

It follows from Theorem 4-1 that the cost of a Huffman tree is upper bounded by

\[
C(T_H) \leq C(T_0) + \frac{1}{2}(n^2+n-1) \frac{n-1}{\ell} \hat{\sigma}
\]

(4-4)

where the tree \( T_0 \) is optimal.

Although the bound in (4-4) is not tight, it does indicate that in most cases of practical interest, the cost of the Huffman tree does not differ substantially from that of an optimal tree for processing a query of the form \( A_1+A_2+\ldots+A_n \). For example, when the length of the lists are approximately equal.

\[
C(T_0) \sim \ell(1 - \frac{\hat{\sigma}}{\ell}) n \log_2 n
\]

where \( \ell \) is the approximate list length. Since \( \frac{1}{2}(n^2+n-1) \frac{n-1}{n} \hat{\sigma} \) is approximately equal to \( \frac{1}{2} n^2 \hat{\sigma} \) for large \( n \), its value becomes comparable to \( C(T_0) \) only when:

\[
\frac{1}{2} \frac{n}{\log_2 n} \sim \frac{\ell}{\hat{\sigma}} (1 - \frac{\hat{\sigma}}{\ell})
\]

(For \( n = 64 \), \( \frac{\hat{\sigma}}{\ell} \approx 0.20 \); for \( n = 8 \), \( \frac{\hat{\sigma}}{\ell} \approx 0.43 \). That is, only for every large overlaps.

Similarly, we bound the cost of a Huffman tree in terms of the maximum overlap \( \hat{\phi} \).
Lemma 4-2

\[ r_k \geq (1 - \frac{\phi}{k}) \sum_{i=1}^{k} a_i \]

Again, \( r_k \) is the length of the list corresponding to the expression \( A_1 + A_2 + \ldots + A_k \).

Proof: Let \( A_k \) be the shortest list among \( A_1, A_2, \ldots, A_k \) and \( R_{k-1} \) be the list corresponding to the expression \( A_1 + A_2 + \ldots + A_{k-1} \). The length of \( R_{k-1} \) \( r_{k-1} \) is clearly larger than the length of any particular list \( A_1, A_2, \ldots, A_{k-1} \). Hence

\[ r_k \geq \sum_{i=1}^{k} a_i - a_k \phi \]

\[ \geq \sum_{i=1}^{k} a_i (1 - \frac{\phi}{k}) \]

Hence, we obtain the following bound for the cost of a Huffman tree for the lists \( A_1, A_2, \ldots, A_n \).

Theorem 4-2

\[ C(T_H) \leq P(T_H) \leq \frac{1}{1 - \frac{1}{2} \phi} C(T) \] (4-5)

where \( T \) is any arbitrary tree for \( A_1 + A_2 + \ldots + A_n \).

Proof: The proof is similar to that of Theorem 4-1. Suppose that \( T' \) is a tree obtained from \( T \) when the weight of \( A_1 \) is replaced by \( a_1(1 - \frac{1}{2} \phi) \).

We claim that

\[ P(T') \leq C(T) \] (4-6)

and hence, the inequalities in (4-5) follows.
To show that the inequality (4-6) is valid, we again look at the tree in Fig. 4-2. The weight of the node \( j \) in the tree \( T' \), \( \tau'_j \), is such that

\[
\tau'_j \geq (a + \sum_{k=j+1}^{\ell} \sum_{A_1 \in S_k} a_1)(1 - \frac{1}{2} \hat{\phi})
\]

On the other hand, the length of the list \( r_{j+1}' \), \( r_{j+1}' \), is such that

\[
r_{j+1}' \geq \sum_{A_1 \in S_{j+1}} a_1(1 - \frac{\hat{\phi}}{|S_{j+1}|})
\]

and the length of the list corresponding to the node \((j+1)\) is lower bounded by

\[
(a + \sum_{k=j+2}^{\ell} \sum_{A_1 \in S_k} a_1)(1 - \frac{\hat{\phi}}{1 + \sum_{k=j+2}^{\ell} |S_k|})
\]

Hence, the merge time required to generate the list corresponding to the node, \( \tau'_j \), is such that

\[
\tau_j \geq (a + \sum_{k=j+1}^{\ell} \sum_{A_1 \in S_k} a_1) \min \left[ (1 - \frac{\hat{\phi}}{|S_{j+1}|}), \left(1 - \frac{\hat{\phi}}{1 + \sum_{k=j+2}^{\ell} |S_k|}\right) \right]
\]

\[
\geq \tau'_j
\]

The inequality in (4-6) follows immediately from this expression.

We again note that the bound in (4-5) is not tight. However, we can conclude from it that for most cases of practical interest \((\hat{\phi} < 0.1)\) the total merge time is quite close to the minimum when the merge order is specified by a Huffman tree.
Performance of Suboptimal Algorithms

Clearly, when the length of overlap is not negligible, the algorithms described in Section III are no longer optimal. However, we have the following special cases:

**Theorem 4-3**

When $\sum_{i=1}^{n} a_i \leq b$, $(A_1 + A_2 + \ldots + A_n) \cdot B$ is optimal among all Boolean expressions equivalent to it. Hence, an optimal merge tree is $T_u$ as shown in Fig. 4-3a when $T$ is an optimal merge tree for $(A_1 + A_2 + \ldots + A_n)$.

Proof: Let $C_1(T)$ denote the cost of the tree $T_u$ in Fig. 4-3a. Suppose that $T(A_1 + A_2 + \ldots + A_n)$ is an arbitrary tree for the lists $A_1, A_2, \ldots, A_n$. Let $R$ be the list corresponding to the Boolean expression $A_1 + A_2 + \ldots + A_n$ and $r$ be its length. Clearly,

$$C_1(T) = C(T) + r + b$$

We need to show that $C_1(T)$ is less than or equal to any tree $T_{m+1}$ shown in Fig. 4-3b. Let $R_1, R_2, \ldots, R_{m-1}, R_m, R_{m+1}$ be the lists corresponding to the roots of $T_1, T_2, \ldots, T_{m-1}, T_m, T_{m+1}$ which are the m+1 subtrees of $T(A_1 + A_2 + \ldots + A_n)$ obtained by removing the roots of larger subtrees of $T$.

Let $r_1, r_2, \ldots, r_{m-1}, r_m, r_{m+1}$ be the lengths of these lists, respectively. Furthermore, let $D_1, D_2, \ldots, D_{m-1}, D_m, D_{m+1}$ be the lists obtained by AND merging $R_1, R_2, \ldots, R_{m-1}, R_m, R_{m+1}$, respectively, with the list $B$.

The cost of the tree $T_{m+1}$, $C_{m+1}$, is

$$C_{m+1} = \sum_{i=1}^{m} [C(T_i) + r_i] + C(T_m) + C(T_{m+1}) + r_m + r_{m+1} + (m+1) b$$

$$+ C(T(D_1 + D_2 + \ldots + D_{m-1} + D_m + D_{m+1}))$$
\[ T(A_1 + A_2 + \ldots + A_n) \]

(a)

\[ T_{u} \]

(b)

Fig. 4-3
Let $T_m$ be a subtree of $T$ obtained by OR merging $R_{m1}$ and $R_{m2}$ together. We have

$$
C(T_m) = \sum_{i=1}^{m} [C(T_i)+r_i] + mb + C(T(D_1+D_2+\ldots+D_m))
$$

where $R_m$ is the root of $T_m$ and $D_m$ is the list obtained by AND merging $R_m$ with $B$.

Since

$$
C(T_m) = C(T_{m1}) + C(T_{m2}) + r_{m1} + r_{m2}
$$

$$
C(T_{m+1}) - C(T_m) = b - (r_{m1}+r_{m2}-r_m)
$$

$$
+ C(T(D_1+D_2+\ldots+D_{m-1}+D_{m1}+D_{m2}))
$$

$$
- C(T(D_1+D_2+\ldots+D_m))
$$

That this difference is larger than or equal to zero follows from

$$
r_{m1} + r_{m2} - r_m \geq 0
$$

$$
b \geq \sum_{i=1}^{n} s_i \geq (r_{m1}+r_{m2}-r_m)
$$

and that $D_m$ is a list obtained by OR merging the lists $D_{m1}$ and $D_{m2}$. Hence, $C(T_m)$ is a monotone nondecreasing function of $m$. In other words, for any tree $T_m$, there is a tree $T_u$ whose cost is less than $C_m$. Hence, we have the statement of the theorem.
V. Summary

The results in Section III allows us to find an optimal form of any nested Boolean expression in which (i) all variables are distinct, and (ii) the complement of any variable,  \( \overline{B} \), (or any expression) can appear only in the form

\[
A_1 \cdot A_2 \cdot \ldots \cdot A_n \cdot \overline{B}
\]

with at least one of the \( A_i \)'s not complemented. We already noted that the restriction (ii) leads to no real loss of generality since it indeed is a restriction imposed on the form of the query in most inverted list document retrieval systems.

The problem of finding the optimal forms of Boolean expressions in which not all variables are distinct is more difficult than that of finding the minimum gate realization of the Boolean expression (when fan-in of gates are 2). Our problem is further complicated by the fact that if one of the terms \( A+B, A\cdot B, \overline{A}\cdot B, \overline{A}\cdot B \) is generated by merging lists A and B, the other three terms are also obtained at no additional cost.

When the lengths of overlaps between lists cannot be neglected in computing the total merge time, the Boolean expression determined by the algorithms described in Section III are no longer optimal. The performance of these algorithms can be bounded in terms of the maximum overlap between the lists as done in Section IV.
References


In an inverted file document retrieval system, a query is in the form of a Boolean expression of index terms. In response to a query, the system accesses the inverted lists corresponding to the index terms, merges them and selects from the merged lists those documents that satisfy the search logic. In this paper, we consider the problem of determining a Boolean expression which leads to the minimum total merge time among all Boolean expressions that are equivalent to the expression given in the query. This problem is the same as finding an optimal merge tree among all trees that realize the truth function determined by the Boolean expression in the query. Several algorithms which generate optimal merge trees, when the sizes of overlaps between different lists are small compared with the length of the lists, are described. These algorithms are no longer optimal when the lengths of overlaps cannot be neglected. In this case, it is possible to bound the performance of these algorithms in some instances in terms of the maximum overlap between lists. The performance bounds are discussed.