New Algorithms for Block Segregated Multiple Key Search Strategy

AHMED TAREK
California University of Pennsylvania
Department of Math and Computer Science
250 University Avenue, California 15419
UNITED STATES OF AMERICA
tarek@cup.edu

Abstract: In this paper, two new algorithms for list segregated multiple key search strategies are discussed. In block search, a given list of elements, which are in sorted order is subdivided into a number of equal sized blocks, and the given key element is compared to the upper bounding element in each block until a bounding element is found, which is either equal to or greater than the key. If the bounding element is greater, then the search is confined in the corresponding block to locate the correct key position. In the proposed algorithms, the block search effort is confined to identify multiple key elements in a given list. The search efforts are optimized after identifying each individual key position by discarding a part of the search space (a portion of the list elements). Here, the list of keys are assumed to be sorted in ascending order, though it is possible to consider a descending list of keys with an ascending list of elements. Once a key index position is identified, the successive keys are searched for at the same or at the higher level blocks containing larger elements. This strategy considerably reduces the search efforts with multiple keys with increasing number of keys in the list. To search for the current key element, the algorithm simply discards the blocks containing the previous smaller keys, and starts at the bounding element of the latest block holding the immediately preceding key. Computational complexity of the proposed algorithms are also considered. Another modification uses the binary search instead of the linear search to identify the correct key position within the block, since the list is in sorted order. With large block sizes, this strategy considerably improves the search time. Therefore, the performance issues of the proposed algorithms are also taken into consideration.

Key–Words: Multiple keys, Ordered list, Block search, Blocking level, Multi-key block search, Bounding element, Multi-key binary search, Binary search, Computational complexity.

1 Introduction

The idea pertaining to the Block Search was first introduced in [2]. Since then this computational paradigm remained neglected, and there was no significant research in this arena. The author has identified block search as a potentially efficient technique to explore, blended with his multiple key search strategies. With the Block Search, also known as the Jump Search, jumps are from block to block until the algorithm precisely identifies the exact bounding block which contains the key element. Therefore, block search may efficiently be applied to multiple key elements. Once the exact bounding block containing the current key index position is identified, a binary search may be applied to find out the exact key index position, and all previous blocks may be discarded from further considerations with the rest of the keys inside the list. This improves the computational efficacy of the Multiple Key Block Search (MLK BS) strategy.

This complex search algorithm deals with two different lists simultaneously. One is usually a much longer list, \( L_1 \) holding the list elements to which, the multiple key search algorithm will be applied to. The other list, \( L_2 \) is usually much shorter compared to \( L_1 \). The algorithm requires both \( L_1 \) and \( L_2 \) to be sorted. In this paper, both \( L_1 \) and \( L_2 \) are assumed to be sorted in ascending order. However, it is also possible to consider a descending list of keys with an ascending list of elements with minor modifications. Alternatively, an ascending list of keys may be search for in a descending list of elements or a descending list of keys may be explored in a descending list of elements. The required modifications for the alternative variations are also outlined.

The algorithm subdivides the list \( L_1 \) with \( n \) elements into \( b \) segregated blocks of sublists. Therefore, each block contains approximately, \( c = \lceil \frac{n}{b} \rceil \) elements. Starting with the first key, \( k_1 \), the key is compared to the 1st element in \( L_1 \). If the list \( L_1 \) element is smaller, \( k_1 \) is compared to the cth element. If the cth element is still smaller, \( k_1 \) is compared with each successive \( p \times c \) elements until a \( p = d \) is found such that \( k_1 \leq d \times c \)th element. Here, \( p = 2, 3, \ldots, b \). If \( k_1 \)
= d \times cth element, the upper bounding element at the dth position contains the key, k_1. If k_1 < d \times cth element, the search for k_1 is confined to the dth block within the range of ((d-1) \times c)th element through the d \times cth element. A binary search is used to identify the exact position within the dth block such that the algorithm exhibits considerable efficiency with larger sub-list sizes in each block. If k_1 > b \times c, then this key and also the successive keys, k_j, j = 2, 3, \ldots, m do not exist in the given list, L_1. Here, m is the size of the list L_2. This strategy is applied for each successive keys, k_j, j = 2, 3, \ldots, m. It may be the case that g out of the m keys exist in L_1, and the rest of the (m - g) keys, do not exist within list, L_1. The algorithm perfectly identifies the keys existing within the given list, and may display appropriate messages regarding the non-existing keys in L_1.

The paper is organized as follows. Section 2 introduces the terms and notations used in this paper. Section 3 is the multiple key block search algorithms. This section also compares the alternative block search strategies and explains the pros and the cons associated with each one of them. The possible combinations for the list of keys and the list of elements are also discussed. Section 4 deals with the computational complexity issues. Section 5 pertains to the algorithmic analysis. Section 6 is the application.

2 Terms and Notations

n: Total number of elements in the larger list.
b: Number of blocks containing n elements in total.
m: Number of keys in the given list of keys.
r: Ratio between the number of list elements to the number of keys, which is \( \frac{n}{m} \).
\( n_i \): The ith block of elements. Here 1 \leq i \leq b.
c: Size of each block in terms of the number of elements within the block. k_j: The jth key in the list of keys. Here, 1 \leq j \leq m.
c: Number of elements in each block = \( \frac{n}{c} \).
\( L_1 \): List containing the n given elements.
\( L_2 \): List containing the m different keys.
t_j: Index position of the jth key, k_j within the list.

Definition 1. Keygap, g: The number of elements in between two successive key elements in a list is known as the Keygap. For instance, suppose that the number of elements in between the key, k_j and the key k_{j+1}, is a. Then the Keygap in between these two keys is, g = a elements.

Definition 2. Inter-key Space Elements, I: Elements in between all successive pair of keys is collectively known as the Inter-key Space Elements (IKSE). This is denoted by I. Then, \( I = \sum_{j=1}^{m-1}(t_{j+1} - t_j) \). If a key is absent from the given list, then the Inter-key Space Elements measures the number of elements between the first previous key existing within the list and the next succeeding key existing within the list. If none of the m keys exist within the list \( L_1 \), \( I = 0 \). If there is only 1 key, say the jth key, \( k_j \) is existent within the list, \( L_1 \), then \( I = (n - t_j) \). If only the first key, \( k_1 \) and the last key, \( k_m \) exist within the list, \( L_1 \), then \( I = (t_m - t_1) \). If only the hth and the sth keys exist within the list \( L_1 \) such that \( 1 \leq h < s \leq m \), then \( I = t_s - t_h \). Now, IKSE, I for the other combinations may also be analyzed similarly.

3 Multi-key Block Search Technique

The recursive version of the multi-key block search algorithm is considered first, which is completely a new paradigm in multi-key block search research.

Algorithm MultiKey_BKLS2key

Purpose: This algorithm performs 2-key recursive block search technique.

The supplied parameters are: array arr[], position of the 1st element: left, position of the last element: right, smaller_key, and larger_key.

2-key block search returns the array key_pos[] (in Java) with key_pos[0] and key_pos[1] corresponding to the smaller and the larger key index positions, respectively.

Require: smaller_key < larger_key

Ensure: Proper index positions are identified.

```java
int n = (right - left + 1) \{The number of elements within the array.\}
int k = \lfloor \sqrt(n) \rfloor
int t = \frac{n}{k}
int blk_arr[] = new int[k + 1] \{Define an array to hold the Block Marker Values.\}

for all j such that 0 \leq j < k do
    blk_arr[j] = j * i
end for

while smaller_key > arr[blk_arr[j]] do
    j ++
end while

left = blk_arr[j - 1]
right = blk_arr[j]
key_pos[0] = binary_search (arr, left, right, smaller_key)
key_pos[1] = binary_search (arr, left, right, larger_key)

while larger_key > arr[blk_arr[j]] do
    j ++
end while
```

```java
```
\[\text{left} = \text{blk}_\text{arr}[j - 1]\]
\[\text{right} = \text{blk}_\text{arr}[j]\]
\[\text{key}_\text{pos}[1] = \text{binary_search} (\text{arr, left, right, larger_key})\]
\[\text{return} \text{ key}_\text{pos}\]

Following is the modified Multiple Key Block Search algorithm for 2 keys. For efficiency reasons, it makes use of the 2-key binary search algorithm (multiple key binary search) as proposed in [1].

**Algorithm AltMultiKey_BLKS2key**

**Purpose:** This algorithm performs 2-key modified block search technique.

The supplied parameters are: array arr[], position of the 1st element: left,
position of the last element (\(n\)th element): right, smaller_key, and larger_key.

2-key block search returns the array key_pos[] (in Java) with key_pos[0] and key_pos[1] corresponding to the smaller and the larger key index positions, respectively.

**Require:** smaller_key < larger_key

**Ensure:** Proper index positions are identified.

\[\text{int} \ n = (\text{right} - \text{left} + 1) \{\text{Total number of elements within the larger array.}\}\]
\[\text{int} \ k = \lfloor \sqrt{n} \rfloor + 1 \{\text{Define the number of blocks within the array.}\}\]
\[\text{int} \ i = \frac{n}{k} \{\text{Find out the number of elements in each block.}\}\]
\[\text{int} \ \text{blk}_\text{arr}[i] = \text{new int}[k + 1] \{\text{Define an array to hold the Block marker values.}\}\]

**for all** \(j\) such that \(0 \leq j < k\) do

\[\text{blk}_\text{arr}[j] = j + i\]

**end for**

\[\text{blk}_\text{arr}[i] = \text{right}\]
\[j = 0\]

**while** smaller_key > arr[blk_arr[j]] do

\[j++\]

**end while**

\[\text{left} = \text{blk}_\text{arr}[j - 1]\]

**while** larger_key > arr[blk_arr[j]] do

\[j++\]

**end while**

\[\text{right} = \text{blk}_\text{arr}[j]\]

\[\text{key}_\text{pos}[1] = \text{BinarySearch}_{2\text{key}} (\text{arr, left, right, smaller_key, larger_key})\]

\[\text{return} \text{ key}_\text{pos}\]

The 2-key binary search algorithm [1] also makes use of the classical 1-key version.

**Algorithm BinarySearch_{2key}**

**Purpose:** This algorithm performs 2-key binary search.

The supplied parameters are: array arr[], position of the first element: left,
position of the last element: right, smaller key, and larger key.

The 2-key search finds out small_pos, large_pos for the smaller and the larger keys.

**Require:** small_key < large_key

**Ensure:** left > right or the keys found

**while** left ≤ right do

\[\text{middle} = \text{left} + (\text{right} - \text{left})/2\]

\[\text{if} \ \text{arr}[\text{middle}] < \text{small_key} \text{ then}\]

\[\text{BinarySearch}_{2\text{key}} (\text{arr, (middle+1), right, small_key, large_key, small_pos, large_pos})\]

\{Recursively call BinarySearch_{2key}\}

**else if** \(\text{arr}[\text{middle}] = \text{small_key}\) then

\[\text{small_pos} = \text{middle}\]

\[\text{large_pos} = \text{BinarySearch(arr, middle+1, right, large_key)}\]

\[\text{return}\]

**else if** \(\text{arr}[\text{middle}] > \text{small_key and arr[middle]} < \text{large_key}\) then

\[\text{small_pos} = \text{BinarySearch(arr, left, middle-1, small_key)}\]

\[\text{large_pos} = \text{BinarySearch(arr, middle+1, right, large_key)}\]

\[\text{return}\]

**else if** \(\text{arr}[\text{middle}] = \text{large_key}\) then

\[\text{small_pos} = \text{middle}\]

\[\text{large_pos} = \text{BinarySearch(arr, left, middle-1, small_key)}\]

\[\text{return}\]

**else if** \(\text{arr}[\text{middle}] > \text{large_key}\) then

\[\text{BinarySearch}_{2\text{key}} (\text{arr, left, middle-1, small_key, large_key, small_pos, large_pos})\]

\[\text{end if}\]

**end while**

\[\text{small_pos} = -1\]

\[\text{large_pos} = -1\]

\[\text{return}\]

Following is the classical 1-key binary search algorithm used in the computation.

**Algorithm binary_search**

**Purpose:** This algorithm performs 1-key recursive binary search.

**while** right ≥ left do

\[\text{middle} = \text{left} + (\text{right} - \text{left})/2\]

\[\text{if} \ \text{arr}[\text{middle}] = \text{key}_\text{element} \text{ then}\]

\[\text{return} \text{ middle}\]

**else if** \(\text{arr}[\text{middle}] > \text{key}_\text{element}\) then

\[\text{return} \text{ binary_search (arr, left, middle-1, key}_\text{element})\] \{recursive call to binary_search\}

**else**

\[\text{return} \text{ binary_search (arr, middle+1, right, key}_\text{element})\]

**end if**
end while
return −1

For the above single level multiple key block search algorithms, the number of blocks, \( b = \sqrt{n} \). According to [2], with only 1 blocking level, efficiency of the block search strategy will be maximum whenever the block size is the square root to the \( L_1 \) list size, which is \( n \).

3.1 The Pros and Cons of the Approaches in Multiple Key Block Search

With multiple key block search strategy, there are 2 variations. With the first one, the appropriate block containing the smallest key is identified. Then within that block, the exact position of the smallest key is identified using a basic binary search algorithm. Next starting with this block, the exact block containing the next larger key is identified, and a basic binary search is again applied within that block for the exact key position. This process is continued until the last or the \( m \)-th key index position is identified.

The alternative approach at first identifies the 2 blocks \( p \) and \( q \) such that \( p \times j \leq k_1 \) and \( q \times j \geq k_m \). Here, \( j = n^{\frac{1}{2}} \) is the size of each block. As for the next step, the Multiple Key Binary Search technique, as proposed in [1] is applied, and the exact locations for the \( m \) different keys are identified in one single execution of the Multi-key Binary Search algorithm. This algorithm in fact combines the block searching with the multiple key binary search algorithm to create a Multi-key Block Search variation.

3.2 List Variations in the Multi-key Block Search Strategy

Assume that the list \( L_1 \) contains the given elements and the list \( L_2 \) contains the given keys. With minor modifications in the multi-key block search strategy, following four search combinations of \( L_1 \) and \( L_2 \) are possible.

The first combination is the one that has been considered in this paper for the rest of the sections. Here, both the lists \( L_1 \) and \( L_2 \) are in ascending order. If the \( j \)-th key \( k_j \) is identified within the \( i \)-th block \( n_i \), then search for the rest of the \( (m-j) \) keys ranging from \( k_{j+1} \) through \( k_m \) is limited only within the blocks \( n_i \) through \( n_b \). Here, \( i \leq b \). Whenever the keys are concentrated within the few initial blocks, the search algorithm will have better performance, if it starts the computation with the last key, \( k_m \), and if the bounding block, \( n_i \) that contains the key \( k_m \) is identified first. In that event, the search for the 1st through the \( (m-1) \)-th keys, which are smaller than the \( m \)-th key may be confined only within the blocks \( n_1 \) through \( n_i \). Therefore, the blocks \( n_{i+1} \) through \( n_b \) holding a total of \( (b-i) \times \frac{n^2}{b} \) list elements may be discarded.

The list \( L_1 \) is in descending order and the list \( L_2 \) is in ascending order. For this combination, if the \( j \)-th key \( k_j \) is identified at the \( i \)-th block \( n_i \), then compute the search for the rest of the keys \( (keys \ k_{j+1} \ through \ k_m) \) only within the blocks \( n_i \) through \( n_b \). Therefore, the remaining \( (b-i) \) blocks of elements starting with the block \( n_{i+1} \) through block \( n_b \) may be discarded safely. If each block contains an equal number of elements, then suppose that each block contains \( c \) elements. Therefore, this search strategy saves \( \frac{(b-i) \times c}{b \times c} = \frac{(b-i)}{b} \) part of the total effort in computing the keys \( k_{j+1} \ through \ k_m \) index positions within the list. The best possible scenario will be when the key \( k_1 \) position is identified within the block \( n_1 \). In that event, if the remaining \((n-1)\) keys exist within the list, they will also exist within the block, \( n_1 \). Therefore, the block search strategy will save \( \frac{(b-i)}{b} \) part of the total block searching computational efforts.

Next consider when both the lists \( L_1 \) and \( L_2 \) are in descending order. If the \( j \)-th key \( k_j \) is identified at the \( i \)-th block \( n_i \), then search for the rest of the \((m-j)\) keys is confined within the blocks \( n_i \) through \( n_b \). This case is pretty similar to the one when both the lists \( L_1 \) and \( L_2 \) are in ascending order.

Consider the case when the list \( L_1 \) is in ascending order, and the list \( L_2 \) is in descending order. If the key, \( k_j \) is identified at the \( i \)-th block \( n_i \), then search for the keys \( k_{j+1} \ through \ k_m \) is restricted only within the blocks \( n_1 \) through \( n_i \). This case is pretty similar to the one with the list \( L_1 \) is in descending and the list \( L_2 \) is in ascending order.

4 Complexity

The proposed algorithms are nearly as good as or exactly as good as the multiple key binary search technique. The algorithms proposed at their different stages of computation make use of the binary search and the multi-key binary search techniques as outlined in [1]. The analysis pertaining to complexity is presented next.

At first, consider the block search technique with only 1 blocking level. An analysis following the one in [2] shows that the best possible result is encountered with the first version of the Multi-key Block Search technique with the number of blocks = \( n^{\frac{1}{2}} \), and each block size = \( \frac{n}{2} \), and the number of keys, \( m \leq b \) or \( m \leq n^{\frac{1}{2}} \). Here \( n \) is the total number of elements within the list, \( L_1 \). For the best possible case, each key, \( k_i \) exists at the middle index position of the
i-th block. With the best case, 2 comparisons are required for the 1st block and 1 comparison for each of the rest \((m-1)\) blocks to determine the exact block for each key. This requires a total of \((m-1+2) = (m+1)\) comparisons. Next, for applying the basic binary search for the \(m\) different keys, each key exists at the middle of the corresponding block and requires only 1 comparison. Hence, altogether, the time complexity function, \(f(n, m) = 1 \times m + (m+1) = 2m+1\). Thus, the big-oh notational complexity is, \(O(m)\). For the second approach, both the smallest and the largest key exist within the same block (the first block), and requires a total of, \(2+1 = 3\) comparisons. The application of the multi-key binary search algorithm requires a total of \(m\) comparisons. Hence, for this approach also, the best case complexity is, \(O(m)\).

Next consider the worst possible case. Here, for \(n^{\frac{1}{2}}\) individual blocks, each key requires \((n^{\frac{1}{2}}+1)\) comparisons and all the keys are within the \(n^{\frac{1}{2}}\)-th block. Each block contains \(\frac{n}{n^{\frac{1}{2}}}\) elements. In the worst possible case, each key requires comparisons in the order of \(\log(\frac{n}{n^{\frac{1}{2}}}) = \log(n^{\frac{1}{2}})\). Hence, in the worst case, \(f(n, m) = m(n^{\frac{1}{2}} + 1) + m\log(n^{\frac{1}{2}}) = mn^{\frac{1}{2}} + m + \frac{m}{2}\log(n)\). The complexity is nonlinear, which is, \(O(mn^{\frac{1}{2}})\). For the average case with the first algorithm, all the keys are within the middle block, which is the \(n^{\frac{1}{2}}\)-th block. Each key requires \(n^{\frac{1}{2}}+1\) comparisons to find out it’s exact block, and within the block, each key requires \(\log(n^{\frac{1}{2}})\) comparisons. Hence, for the average case, \(f(n, m) = m \times (n^{\frac{1}{2}} + 1) + m \times \log(n^{\frac{1}{2}}) = m \times (n^{\frac{1}{2}}) + m + \frac{1}{2}m\log(n)\). Hence the average case complexity is also nonlinear, which is, \(O(mn^{\frac{1}{2}})\). In the complexity hierarchy, \(\log(n) < n^{\frac{1}{2}}\). This is true, since for sufficiently large values of \(n\), \(n^{\frac{1}{2}}\) will supersede \(\log(n)\). Only the larger values of \(n\) come into play in determining the computational complexity. Hence, with large enough values of \(n\), \(m \times (n^{\frac{1}{2}})\) will be larger than \(m\log(n)\).

5 Analysis

5.1 Block Partitioning

Following result on block partitions holds true with the proposed \(m\)-key Block Search Technique.

**Theorem 3.** Proposed Multi-key Block Search technique has a time complexity, which is \(O(\log(\sqrt{n}))\), where the number of elements, \(n \gg\) the number of keys, \(m\).

**Proof:** With the Multi-key Block Search technique, a list containing \(n\) elements are subdivided into a number of blocks. Each block contains \(\lfloor \sqrt{(n)} \rfloor\) elements. Let \(n \simeq k^2\), where \(k\) is an integer. Therefore, there are \(k = \sqrt{(n)}\) elements in each block. Also the total number of blocks is, \(k\). For the 1st block, the bounding index positions are, \(\text{left}_1 = 0\), and \(\text{right}_1 = (k – 1)\). For the 2nd block, \(\text{left}_2 = k\), \(\text{right}_2 = (2k – 1)\). Proceeding in this way, for the \(k\)th block, \(\text{right}_k = (k \times (k – 1) + k\). For performing a single key search, the average and the worst-case complexities using the block-search technique is, \(O(\log_2 k)\). With a total of \(m\) keys, the average and the worst-case complexities are, \(O(m \log_2 k)\). If \(k = \sqrt{(n)} \gg m\), the overall complexity of the proposed multi-key block search technique is, \(O(\log(k))\) or \(O(\sqrt{(n)})\).

**Corollary 4.** If the number of elements in a list is kept constant at \(n\), then the following results hold true.

1. The number of blocks, \(b\) is inversely proportional to the size of each block, \(c\) in terms of the number of elements with no nested blocking.

2. If the size of each block without nesting is, \(c = n^{\frac{1}{2}}\), then the total number of blocks is, \(b = n^{1 \frac{k-1}{k}}\).

**Proof:** (1) Suppose that for the block search strategy without nesting, the size of each block is, \(c_y\). For another strategy without nesting, the size of each block is, \(c_y\). Since there is no labeling within the block search, following equation holds true:

\[
\text{block size}, c \times \text{total number of blocks}, b = \text{number of elements}, n \quad (1)
\]

Using equation 1, \(c_x \times b_x = c_y \times b_y\). This provides, \(\frac{c_x}{c_y} = \frac{b_y}{b_x}\). This means that, in general, \(c \propto \frac{1}{b}\).

(2) From the equation 1, \(c \times b = n\). But \(c = n^{\frac{1}{2}}\). Therefore, \(n^{\frac{1}{2}} \times b = n^{1}\). Hence, \(b = n^{1 \frac{k-1}{k}} = n^{\frac{1}{k}} = n^{\frac{1}{k-1}}\).

6 Application

The multiple key block search algorithms outlined in this paper may effectively be used to perform database searching with multiple key elements to compare among the corresponding records. For example, a single table containing a list of possible country names could be used to hold all of the approximately 190 entries. To compare among the population attributes of different countries in an efficient way, which is stored inside this table, the multi-key block search technique may be used effectively. The 26 upper-case letters from the alphabet will serve as the \(m\) keys in this search. The sorted list is split alphabetically into 26 blocks of unequal size using the countries first letter.
as an index to 26 pointers (26 boundary values) to start of each block. For comparing among multiple differ-
ent countries using the block search, their names also come from a sorted list. Hence, once a country be-
ing with the smaller key (upper-case alphabetical letter having a smaller value in the lexicographic se-
quence) is identified in a particular block, the search for the following countries will only be restricted in this and the next higher blocks with the following letters inside the alphabet. Therefore, the multi-key
block search will considerably improve the search ef-
ficiency. In the worst case, whenever a country’s name starts with the letter 'Z', it will require only 26 com-
parisons to identify it’s correct block.

7 Conclusion

This paper proposes two new algorithms for the list sub-
division based multiple key search strategies, and analyzes performance to judge their computational
effectiveness. The concept pertaining to the Block Search for a single key was introduced in [2]. In Com-
puter Science, a jump search or block search refers to a search algorithm to find the key index position inside
of an ordered list. It works by first checking all the bounding elements, \( k \times c \), where \( k = 0, 1, 2, \ldots, b \), and \( c \) is the block size, until an item \( q \times c \) is found
that is larger than the search key. To find the exact
position of the search key within the list, a traditional
linear search is performed on the sublist bounded by
the items \((q - 1) \times c \) and \( q \times c \). With just a single layer
and no sub-layer within, the optimal value of \( c \) is \( n^{\frac{1}{2}} \).
Here, \( n \) is the length of the list, \( L_{1} \). The advantage
of block search over the binary search is that a jump search only requires to jump backwards once, while a
binary search can jump backwards up to \( \log(n) \) times. This makes significant difference if a backwards jump takes significantly more time than a forward jump [2].

The block search algorithm may be modified by performing multiple levels of jump search on the sub-
lists, before finally performing the linear search. For a \( k \)-level block search, the optimum block size, \( c_{l} \) for
the \( l \)-th level counting from the 1st level is, \( n^{\left(\frac{k-l}{k}\right)} \). The modified algorithm will perform \( k \) backwards
jumps. For the single key block search, if there are
2 levels of jumping, the optimum is, \( c = n^{\frac{3}{2}} \) for the
first level of blocks, and \( c = n^{\frac{5}{4}} \) for the second level of
blocks [2].

With the proposed algorithms, two new strategies
are considered to find out the positions of \( m \) different
keys in a given sorted list. With the first strategy, at
first the most appropriate block holding the smallest
key is found. Next within the appropriate block, basic
binary search technique is applied to find out the ex-
act key location. For the next larger key, starting with
the block containing the smallest key, other blocks
holding larger elements are searched for to identify
the most appropriate block containing that next larger
key. Then again, binary search is applied to determine
the exact key position. This technique is carried over
for the rest of the keys. With the alternative algorithm,
the blocks containing the smallest and the largest keys
are found first. Then the blocks that are surrounded
by these two extreme blocks that include the smallest
and the largest keys are considered as one contiguous
list of elements. This also includes the two extreme
blocks holding the smallest and the largest keys. As
the next step, the Multi Key Binary Search Tech-
nique, as proposed by the author in [1] is applied to
the optimized list, and the exact locations of \( m \) dif-
f erent keys are identified in one single execution of
the Multi-key Binary Search algorithm. This tech-
nique integrates the flavor of the Block Search tech-
nique with multiple element binary search. Another
advantage with this technique is that, if all the keys
are concentrated within a relatively fewer number of
consecutive blocks, it is also possible to avoid the
wastage due to computation within the other out-of-
the-range blocks. Also, for the average and the worst
case multi-key binary search cases, multi-key block
search will demonstrate much better performance in
general. With both the multiple key block search algo-
rithms proposed in this paper, only one level of block-
ing is considered. However, it is also possible to con-
sider multiple level of blocking with both the algo-
rithms. This remains as a future research issue.

Acknowledgements: This research was supported by
the Department of Mathematics and Computer Science
at California University of Pennsylvania.

References:

Multi-Key Binary Search, Proceedings of the 8th
World Multiconference on Systemics, Cybernet-
ics and Informatics SCI 2004, Volume II, Pages
222-227, Orlando, Florida, U.S.A.

Sequential Search Technique, Communications
of the ACM Transaction (CACM) 21(10), 1978,
pp. 831–834.

[3] Richard F. Gilberg and Behrouz A. Forouzan,
Data Structures-A Pseudocode Approach with
C++, Brooks/Cole, Thomson Learning, Califor-

[4] Sartaj Sahni, Data Structures, Algorithms and