SIAM J. COMPUT.
Vol. 9, No. 1, February 1980

# ON THE COMPLEXITY OF SEARCHING A SET OF VECTORS*

D. S. HIRSCHBERG†

**Abstract.** The vector searching problem is, given $k$-vector $A$ (a $k$-vector is a vector that has $k$ components, over the integers) and given a set $\bar{B}$ of $n$ distinct $k$-vectors, to determine whether or not $A$ is a member of set $\bar{B}$. Comparisons between components yielding "greater than-equal-less than" results are permitted. If the vectors in $\bar{B}$ are unordered then $nk$ comparisons are necessary and sufficient. In the case when the vectors in $\bar{B}$ are ordered, it is shown that $\lfloor \log n \rfloor + k$ comparisons are necessary and, for $n \geqq 4k$, $k\lceil \log (n/k) \rceil + 2k - 1$ comparisons are sufficient.

**Key words.** searching, vector, lower bounds, complexity

**Searching an unordered set.** We first consider the case in which the vectors in $\bar{B}$ are *not* ordered. In this case, an upper bound of $nk$ comparisons can be easily demonstrated. A simple adversary can be constructed to show that $nk$ comparisons are also necessary.

A nontrivial dynamic adversary can be used to construct an oracle to demonstrate that $nk$ comparisons are necessary even if we allow comparisons between elements of the vectors in $\bar{B}$ as well as comparisons between elements of $A$ and vectors in $\bar{B}$ [3].

Recently, Stockmeyer and Wong have demonstrated upper and lower bounds that are within a small factor from one another for the more general problem of determining the intersection of two sets of vectors [7].

For a review of the use of oracles to derive lower bounds, the reader is referred to [1], [4], [6].

**Searching an ordered set.** We now consider the case in which preprocessing of the set $\bar{B}$ is permitted. That is, we can assume that $\bar{B}$ is in some prearranged order, such as lexicographic order.

In the discussions that follow, all logarithms are assumed to be base 2.

A lower bound of $\lfloor \log n \rfloor + k$ comparisons can be seen by observing that $\lfloor \log n \rfloor + 1$ comparisons are required to determine if there is any vector having the correct value of one component, and $k-1$ comparisons are required to verify the agreement of the remaining components.

The oracle for distinguishing a path (which will be of length at least $\lfloor \log n \rfloor + k$) in each decision tree that solves this problem is as follows.

Initially, define *low* $= 1$ and *high* $= n$.

Let the next comparison presented to the oracle be $a_j : b_{ij}$.

```
    mid ← (low + high)/2
    **If** low ≠ high **then**:
        **if** $i <$ low **then** return>
        **if** low $\leqq i \leqq$ mid **then** [low ← $i + 1$; return>]
        **if** mid $< i \leqq$ high **then** [high ← $i - 1$; return<]
        **if** high $< i$ **then** return<
    **Else** (low = high) return=
```

*low* will not equal *high* until after at least $\lfloor \log n \rfloor$ comparisons. During these comparisons, vector $A$ could equal any of the vectors $B_{\text{low}} \cdots B_{\text{high}}$.

When low = high, until *all* components of $B_{\text{low}}$ have been compared with $A$, $B_{\text{low}}$ may equal $A$ but it is also possible that one component of $B_{\text{low}}$ will be less than the

* Received by the editors January 12, 1978, and in revised form March 30, 1979. This research was supported by The National Science Foundation under Grant MCS-76-3933.

† Department of Electrical Engineering, Rice University, Houston, Texas 77001.

corresponding component of $A$ and, in that case, it is possible that none of the vectors in $\bar{B}$ are equal to $A$.

Thus $\lfloor \log n \rfloor + k$ comparisons are necessary to solve this problem.

In the above analysis, we assumed that all comparisons are between a component of $A$ and the corresponding component of a vector in $\bar{B}$. It is straightforward to generalize and allow comparisons between components of vectors both of which are elements of $\bar{B}$.

Having demonstrated a lower bound, we now consider upper bounds for this problem.

We present and analyze two algorithms that solve the ordered set problem and then combine them to obtain an algorithm that is faster than both.

The first algorithm is an example of *binary search*. Let $\bar{B} = \{B_1, B_2, \cdots, B_n\}$ and let $B_i = b_{i1} \cdots b_{ik}$. Proceed comparing the components of $A$ with those of the central vector, i.e. compare $a_h$ with $b_{jh}$ for $h = 1, 2, \cdots$ where $j = \lfloor (n+1)/2 \rfloor$. If all comparisons result in "equal" then $A = B_j$. Otherwise, if at some point we get a "less than" result then $A \neq B_j$ and we can restrict our attention to $\bar{B}' = \{B_1, \cdots, B_{j-1}\}$. Similarly, if we get a "greater than" result, then we can restrict our attention to $\bar{B}' = \{B_{j+1}, \cdots, B_n\}$. In the worst case, we will require $k$ comparisons in each of $1 + \lfloor \log n \rfloor$ iterations for a total of $k + k\lfloor \log n \rfloor$ comparisons. This is equivalent to the result in [2].

The second algorithm uses *linear search* and is as follows:

```
        $j \leftarrow 1$
        $h \leftarrow 1$
        **while** $j \leqq n$ AND $h \leqq k$
        **do compare** $a_h : b_{jh}$
            **if** $=$ **then** $h \leftarrow h+1$
            **else if** $>$ **then** $j \leftarrow j+1$
            **else** [**print** 'NO SOLUTION'; **stop**]
        **od**
        **if** $j > n$ **then** [**print** 'NO SOLUTION'; **stop**]
final:  **if** $a_i = b_{ji}$ for all $i \in \{1, 2, \cdots, k-1\}$
        **then print** $j$; **comment** $A = B_j$
        **else print** 'NO SOLUTION'
        **stop**
```

The algorithm finds the first (lowest indexed) vector, $B_j$, that matches $A$ in the $h$th component. All lower indexed vectors are not considered further. All other vectors are *assumed* to match in this component. The algorithm then iterates on the $(h+1)$st component. This part of the algorithm will make at most $n + k - 1$ comparisons (each iteration increments either $j$ with upper limit $n$, or $h$ with upper limit $k$). If we succeed in matching all $k$ components in this manner then the vector, $B_j$, that is found will be equal to $A$ if all assumptions made earlier apply to $B_j$. However, if $B_j$ disagrees with $A$ in any component $h'$ then $A$ does not appear in $\bar{B}$ since all $j' < j$ have been eliminated, $B_j \neq A$ (assumed here) and for all $j'' > j$, $B_{j''}$ will disagree with $A$ among the first $h'$ components since $\bar{B}$ is in lexicographic order. The final phase of the algorithm, in which the components of $B_j$ (which were assumed to agree with $A$) are compared with $A$, requires at most $k - 1$ comparisons for a total of at most $n + 2k - 2$ comparisons.

We note that the linear search algorithm's mirror image also works. That is, we can start with $j = n$ and decrement $j$, being careful to interchange the $<$'s and $>$'s. We can, as an initial improvement, compare $A$ with the central vector in $\bar{B}$ rather than with $B_1$ or $B_n$ and, at the first "less than" or "greater than" result, continue with the linear search algorithm applied to only half of the original set $\bar{B}$. This leads to an algorithm that requires, in the worst case, only $\lfloor n/2 \rfloor + 2k - 1$ comparisons. We call this improved algorithm the *modified linear search* algorithm.

We can make further improvements by deciding, at the time that a "less than" or "greater than" result is obtained, whether to continue in the style of the binary or the modified linear search algorithm depending upon which will lead to fewer comparisons in the worst case. If, after making $h$ comparisons (resulting in "equal") along vector $B_j$ within feasible set $\bar{B}$ of cardinality $n$, we make a comparison resulting in "less than" or "greater than" then continuing with the linear search algorithm requires, in the worst case, at most $\lfloor n/2 \rfloor + 2k - 1 - h$ additional comparisons. If, however, we decide to proceed with the comparisons in a new vector within $\bar{B}$ and thus follow the binary search algorithm or follow the modified linear search algorithm on a feasible set of half the size, then we will have upper bounds of $k \lfloor \log n \rfloor$ and $\lfloor n/4 \rfloor + 2k - 1$ additional comparisons respectively. We should continue with linear search only if

$$\lfloor n/2 \rfloor + 2k - 1 - h < \min\{\lfloor n/4 \rfloor + 2k - 1, k \lfloor \log n \rfloor\}$$

which holds only if $n < 4h$. For particular values of $k$, we can solve this inequality to gain further restrictions. For example, if $k = 3$ then we should continue with linear search only if $n = 4$ and $h = 2$ or $n = 5$ and $h = 2$.

Let $T(n, k)$ be the minimum number of comparisons required for the ordered vector search problem when $\bar{B}$ consists of $n$ $k$-vectors. Then, for $n \leqq 4k$, $T(n, k) \leqq \lfloor n/2 \rfloor + 2k - 1$.

For $n = k2^r$, $T(n, k) \leqq k + T(n/2, k) \leqq (r-2)k + T(4k, k) \leqq (r+2)k - 1$.

For $k2^{r-1} < n < k2^r$, $T(n, k) \leqq (r-2)k + T(4k-1, k) \leqq (r+2)k - 2$. Note that in both cases, $r = \lceil \log (n/k) \rceil$.

Algorithm VECTOR_SEARCH incorporates the modifications mentioned above.

```
         VECTOR_SEARCH ($A$, $\bar{B}$, $n$, $k$)
         low ← 1
         high ← $n$
binary:  binsearch ← TRUE
         **while** binsearch AND low ≦ high
         **do** $j$ ← (low + high)/2
             **compare** $a_1 : b_{j1}$
             **if** > **then** high ← $j - 1$
             **else if** < **then** low ← $j + 1$
             **else** binsearch ← FALSE
         **od**
         **if** low > high **then** [**print** 'NO SOLUTION'; **stop**]
         $n$ ← high − low + 1
modlin:  $h$ ← 2
         **while** $h \leqq k$
         **do compare** $a_h : b_{jh}$
             **if** = **then** $h$ ← $h + 1$
             **else if** > **then if** $n \geqq 4*h$
                            **then** [high ← $j - 1$; **goto** binary]
                            **else goto** linear
             **else if** < **then** **if** $n \geqq 4*h$
                            **then** [low ← $j + 1$; **goto** binary]
                            **else goto** linear2
```

```
          od
          print $j$; comment $A = B_j$
          stop
linear:   while $j \leqq$ high AND $h \leqq k$
          do compare $a_h : b_{jh}$
                if = then $h \leftarrow h + 1$
                else if > then $j \leftarrow j + 1$
                else [print 'NO SOLUTION'; stop]
          od
          if $j >$ high then [print 'NO SOLUTION'; stop]
final:    if $a_i = b_{ji}$ for all $i \in \{1, 2, \cdots, k-1\}$
          then print $j$; comment $A = B_j$
          else print 'NO SOLUTION'
          stop
linear 2: while $j \geqq$ low AND $h \leqq k$
          do compare $a_h : b_{jh}$
                if = then $h \leftarrow h + 1$
                else if < then $j \leftarrow j - 1$
                else [print 'NO SOLUTION'; stop]
          od
          if $j <$ low then [print 'NO SOLUTION'; stop]
          goto final
```

## REFERENCES

[1] A. V. Aho, D. S. Hirschberg and J. D. Ullman, *Bounds on the complexity of the longest common subsequence problem*, J. Assoc. Computer Mach., 23 (1976), pp. 1–12.

[2] D. Dobkin and R. J. Lipton, *Multidimensional searching problems*, this Journal, 5 (1976), pp. 181–186.

[3] D. S. Hirschberg, *On the complexity of vector searching*, Rice Univ. Tech. Rept. #7807, Houston, TX, June 1978.

[4] D. E. Knuth, *The Art of Computer Programming*, vol. 3. Addison-Wesley, New York, 1973.

[5] V. V. Raghavan and C. T. Yu, *A note on a multidimensional searching problem*, IPL 6 (1977), pp. 133–135.

[6] E. M. Reingold, *On the optimality of some set algorithms*, J. Assoc. Comput. Mach., 19 (1972), pp. 649–659.

[7] L. J. Stockmeyer and C. K. Wong, *On the number of comparisons to find the intersection of two relations*, IBM Watson Research Center Tech. Rept., 1978.