K-Nearest Neighbor Finding
Using the MaxNearestDist Estimator

Hanan Samet

hjs@cs.umd.edu   www.cs.umd.edu/~hjs

Department of Computer Science

Center for Automation Research

Institute for Advanced Computer Studies

University of Maryland

College Park, MD 20742, USA

Similarity Searching

1. Important task when trying to find patterns in applications involving mining different types of data such as images, video, time series, text documents, DNA sequences, etc.
2. Often reduces to finding k nearest neighbors of query object
3. Organize data by use of hierarchical clustering
   - partition data in clusters which are aggregated to form other clusters
   - total aggregation is represented as a tree
4. Search hierarchies used by algorithms are partly specific to vector data but can be adapted to non-vector data as well
5. Algorithms are applicable to any index based on hierarchical clustering

Best-First Method

1. Explores elements of the search hierarchy in increasing order of their distance from the query object q
2. Achieved by storing nonobject elements of the search hierarchy in a priority queue in this order
3. Some algorithms also store the objects in the priority queue enabling the algorithms to be incremental
   - implies both objects and nonobjects are visited in increasing order of distance
   - no need to know k in advance and can obtain neighbors one by one
4. May need as much storage as total number of nonobjects (and hence objects) if their distance from q is approximately the same

Depth-First (Branch-and-Bound) Method

1. Order of exploring elements of search hierarchy is result of a depth-first traversal of hierarchy using distance D_k from the query object to the current $k^{\mathtt{th}}$-nearest object to prune the search
   - most commonly used method
2. Advantage over best-first method is that amount of storage is bounded by k instead of by the number of objects
3. Advantage of best-first is avoiding visiting nonobject elements that will eventually be determined to be too far from q due to poor initial estimates of D_k

Overview

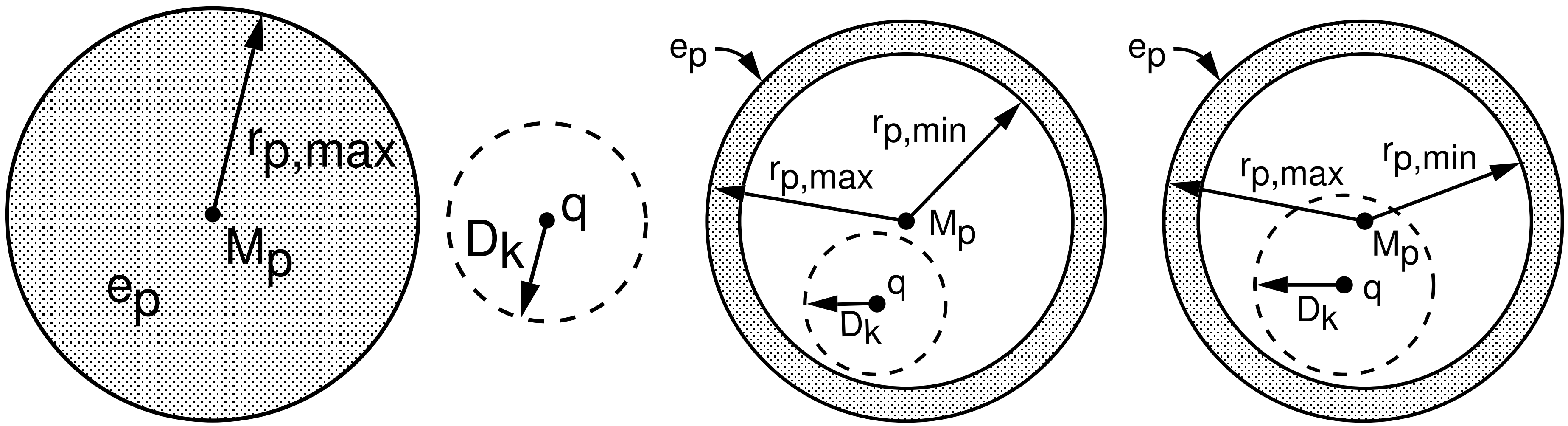
1. Implementations of both depth-first and best-first have traditionally used the estimate of the minimum distance at which a nearest neighbor can be found to prune the search
2. Describe use of an estimate of the maximum possible distance at which a neighbor must be found to prune the search for finding the k nearest neighbors
3. New estimate helps each algorithm overcome its disadvantages vis-a-vis each other
   - prunes number of nonobject elements that must be examined in depth-first algorithm
   - reduces number of nonobject elements that must be retained in priority queue for best-first algorithm
4. Main focus is on how new estimate is incorporated in depth-first algorithm
   - incorporated similarly in best-first algorithm
   - comparison of two algorithms is beyond scope of this work

Depth-First Algorithm

recursive procedure DFTRAV(e)
if ISLEAF(e) **then** /* e is a leaf with objects */
  foreach object child element o of e **do**
    Compute $d(q,o)$
    if $d(q,o) < D_k$ **then** INSERTL$(o, d(q,o))$
    endif
  enddo
else
  Generate active list A containing child elements of e
  foreach element e_p of A **do** DFTRAV(e_p)
  enddo
endif

Speeding Up Depth-First Algorithm – 1

1. DFTRAV visits every element in search hierarchy
2. No point in visiting an element and its objects if it is impossible for it to contain any of k nearest neighbors of q
   - e.g., when $d(q,e) \leq d(q,e_0)$ for every nonobject element e in search hierarchy and for every object e_0 in e and that $d(q,e) > D_k$
   - always true if define $d(q,e)$ as minimum distance from q to any object e_0 in nonobject e (MINDIST)



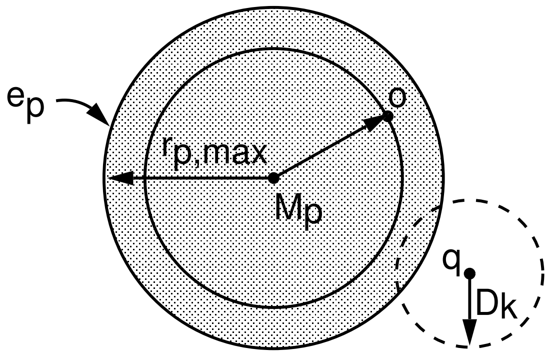
q outside cluster

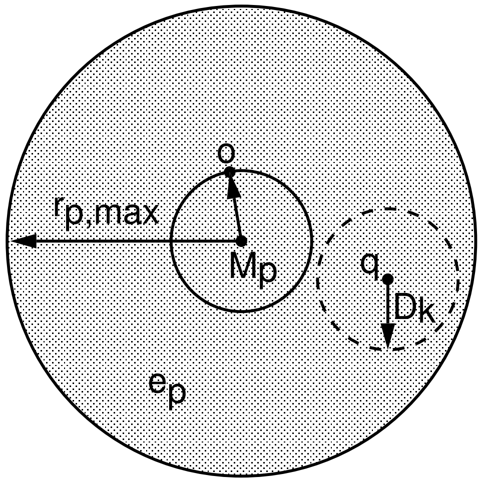
q inside inner sphere of cluster

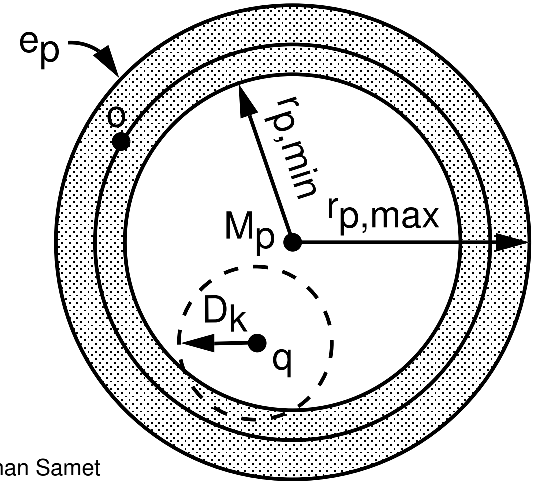
Speeding Up Depth-First Algorithm – 2

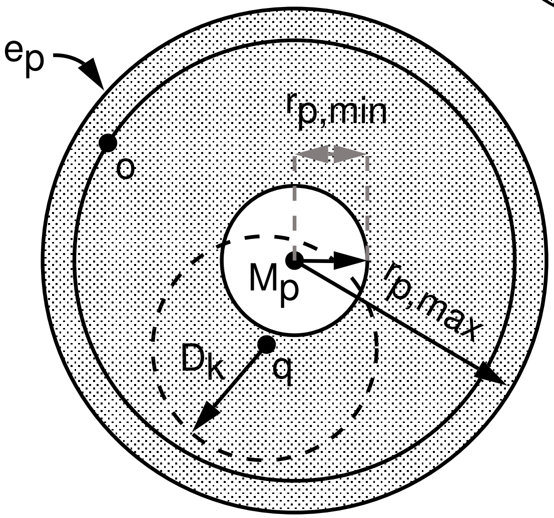
3. If process elements of active list $A(e)$ in MinDist order, then as soon as find one element e_i in $A(e)$ such that $d(q, e_i) > D_k$, then no need to process remaining elements e_j of $A(e)$ as $d(q, e_j) > D_k$
   - exit loop and backtrack to parent of e, OR
   - terminate if e is root of search hierarchy

4. Can also avoid visiting some objects in a nonobject element



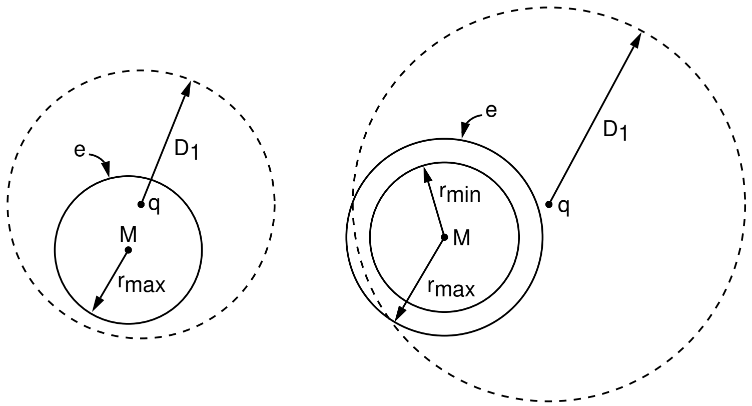




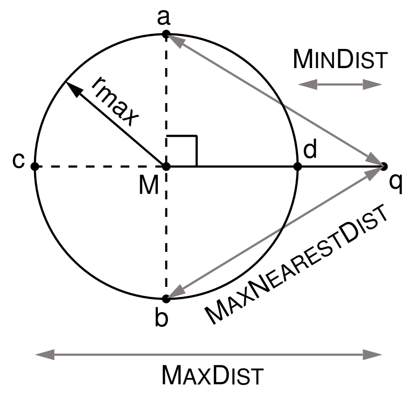


MAXNEARESTDIST Estimator

- Tighten value of estimate of distance to nearest neighbor D_1
  1. MAXDIST: maximum distance from q to an object in e (Fukunaga and Narendra, 1975)
  2. MAXNEARESTDIST: maximum possible distance from q to nearest neighbor in e (Larsen and Kanal, 1986)



- Ex: assume search hierarchy consists of minimum bounding hyperspheres

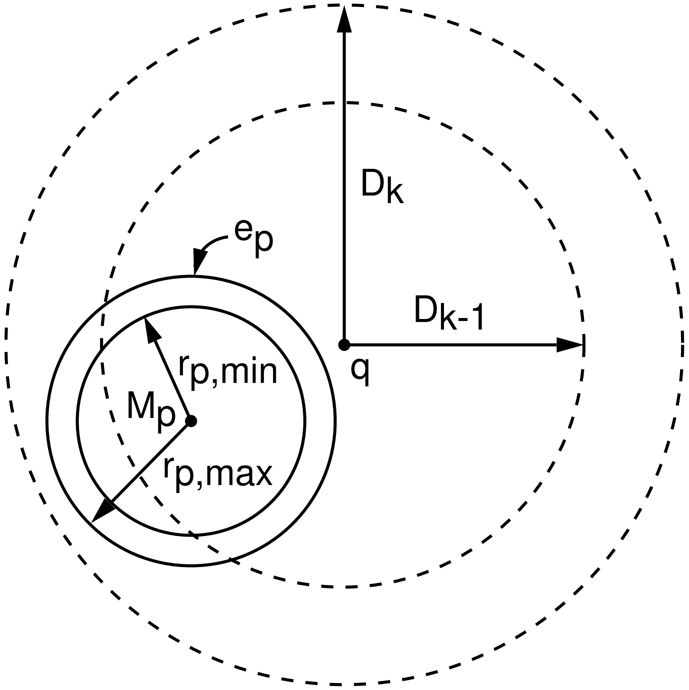


Extending MaxNearestDist for Arbitrary k

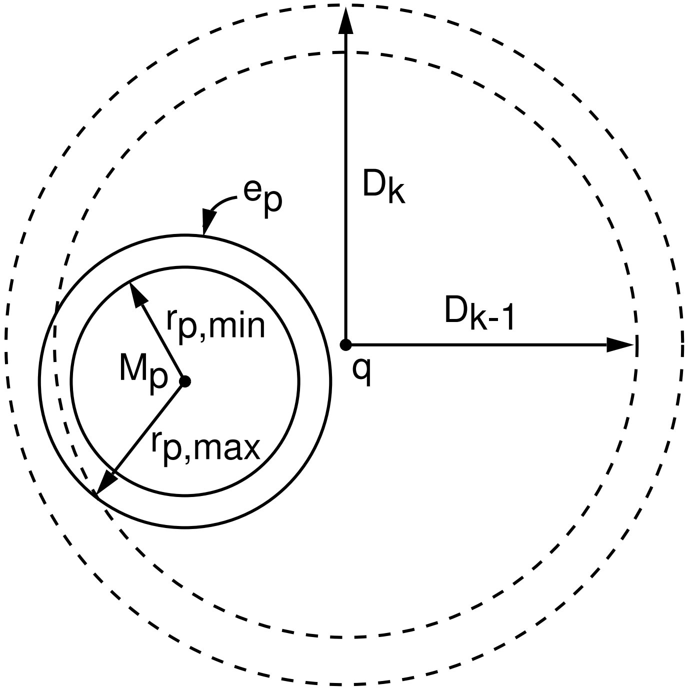
- Cannot simply reset D_k to MaxNearestDist(q, e) whenever MaxNearestDist$(q, e) < D_k$
- Problem: distance s from q to some of its k nearest neighbors may lie within MaxNearestDist$(q, e) < s \leq D_k$, and thus resetting D_k to MaxNearestDist(q, e) may cause them to be missed, especially if child element e contains just one object
- Need to examine the values of D_i $(1 \leq i < k)$

Alternative Solution

- Whenever find that $\text{MAXNEARESTDIST}(q,e) < D_k$, reset D_k to $\text{MAXNEARESTDIST}(q,e)$ if $D_{k-1} \leq \text{MAXNEARESTDIST}(q,e)$ (a); otherwise, reset D_k to D_{k-1} (b)



(a)



(b)

- Problem: if $D_{k-1} > \text{MAXNEARESTDIST}(q,e)$, then now both D_k and D_{k-1} are equal, and from now on we will never be able to obtain a lower bound on D_k than D_{k-1}

Ultimate Solution

- Overcome by adding additional explicit check to determine if $D_{k-2} \leq$ MaxNearestDist(q, e_p), in which case reset D_{k-1} to MaxNearestDist(q, e_p); otherwise,reset D_{k-1} to D_{k-2}
- Only temporary remedy as break down again if $D_{k-2} >$ MaxNearestDist(q, e_p)
- Only solution is to repeatedly apply the method until finding smallest $i \geq 1$ such that $D_i >$ MaxNearestDist(q, e_p)
- Once locate this value of i, set D_i to MaxNearestDist(q, e_p) after resetting D_j to D_{j-1} ($k \geq j > i$).

Additional Problems

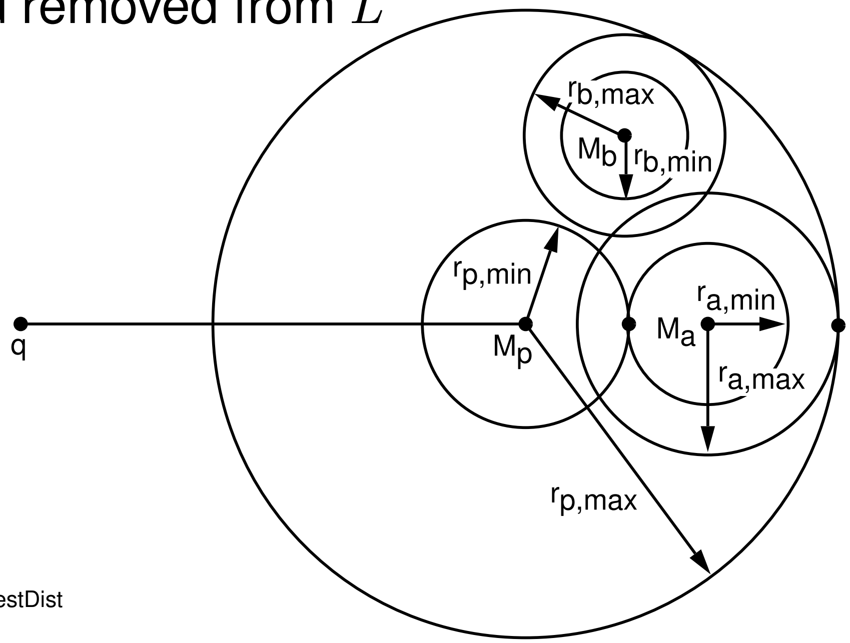
1. No guarantee that objects associated with the different $D_j (1 \leq j \leq k)$ values are unique
   - problem: same object o may be responsible for the MAXNEARESTDIST value associated with both elements e_p and e_a of the search hierarchy that caused MAXNEARESTDIST$(q, e_p) < D_k$ and MAXNEARESTDIST$(q, e_a) < D_k$, respectively, at different instances of time
   - of course, this situation can only occur when e_p is an ancestor of e_a, but must be taken into account as otherwise results of the algorithm are wrong
2. Primary role of MAXNEARESTDIST estimator is to set an upper bound on distance from q to nearest neighbor in a particular nonobject element
   - NOT same as saying that it is the minimum of maximum possible distances to k nearest neighbor of q, which is not true!
   - instead, MAXNEARESTDIST should be used to provide bounds for different clusters (nonobject elements)
   - only once we have k distinct such bounds do we have an estimate on the distance to the k^{th} nearest neighbor

Use of MaxNearestDist in Depth-First Algorithm

1. Expand role of list L of k nearest neighbors
   - object elements and distance from q
   - also nonobject elements corresponding to elements in active list and their corresponding MaxNearestDist values
2. Each time process a nonobject element e, insert in L all of e's child elements that comprise e's active list with their corresponding MaxNearestDist values
   - before inserting the child elements of e in L, remove e from L
   - ensures no ancestor-descendant relationship for any pair of items in L
   - implies object o associated with nonobject element u of L at distance MaxNearestDist is unique
3. Each entry u in L with distance d_u ensures that there is at least one object in the data set whose maximum possible distance from q is d_u
4. Implement L using a priority queue so can access farthest of k nearest neighbors as well as update (i.e., insert and delete k nearest neighbor) without the needless exchange operations if L was an array

D_k Is Not the Same as $D(L_k)$

1. $D(L_k)$: distance associated with the entry in L corresponding to q's k^{th}-nearest neighbor
2. D_k: keeps track of the minimum of $D(L_k)$ as algorithm progresses
3. D_k is not necessarily equal to $D(L_k)$
   - cannot guarantee that the MAXNEARESTDIST values of all of e's immediate descendents (i.e., the elements of the active list of e) are smaller than e's MAXNEARESTDIST value
   - only know that distance from q to the nearest object in e and its descendents is bounded from above by MAXNEARESTDIST value of e
   - in other words, D_k is nonincreasing, while $D(L_k)$ can increase and decrease as items are added and removed from L
   - Ex: $D(L_k)$ must increase when element $E(L_k)$ has just two sons e_a and e_b both of whose MAXNEARESTDIST values are $> D(L_k)$



Need to Insert All Nonobject Elements in L?

1. D_k is reset to $D(L_k)$ whenever upon insertion of a nonobject element e_p into L, with its corresponding MAXNEARESTDIST value, we find that L has at least k entries and that $D(L_k)$ is less than D_k as this corresponds to the situation that MAXNEARESTDIST$(q, e_p) < D_k$
2. Do not reset D_k upon explicitly removing a nonobject element from L as D_k is already a minimum and thus it cannot decrease further as a result of the removal of a nonobject element although it may decrease upon the subsequent insertion of an object or nonobject
3. Nonobjects can only be pruned on basis of their MINDIST values, in which case they should also be removed from L as their MAXNEARESTDIST value is always greater than their MINDIST value which is greater than D_k
   - no harm in not removing any of the pruned nonobjects from L as neither the pruned nonobjects nor their descendents will ever be examined again as all of their MINDIST (and MAXNEARESTDIST) values are already greater than D_k which is nonincreasing
   - drawback of not removing from L is that L can get much larger than k
   - can get as large as $O(k + m \cdot \log N)$ when no nonobject elements in active list have been pruned and at deepest level of search hierarchy

Limiting the Size of L

1. Only reason for L to keep track of the MAXNEARESTDIST values of nonobject elements is to enable lowering the known value of D_k so that more pruning will be possible in the future
2. D_k being nonincreasing means that should not insert into L any nonobject element e such that MAXNEARESTDIST$(q,e) \geq D_k$
   - no problem when trying to remove e where MINDIST$(q,e) < D_k$ while MAXNEARESTDIST$(q,e) \geq D_k$ in order to ensure that same object o is not responsible for the presence of both e and a descendant nonobject element of e being in the k closest elements of L to q at the same time
3. When insert into L and try to update D_k, only examine first k elements of L
   - implies no need for L to even contain more than k elements
   - however, when try to explicitly remove nonobject element e from L just before inserting in L all of e's child elements, e might no longer be in L
   - e could have been implicitly removed as a byproduct of the insertion of closer objects or nonobject elements with lower MAXNEARESTDIST values than that of e thereby resulting in resetting D_k

Removing Nonobjects from L

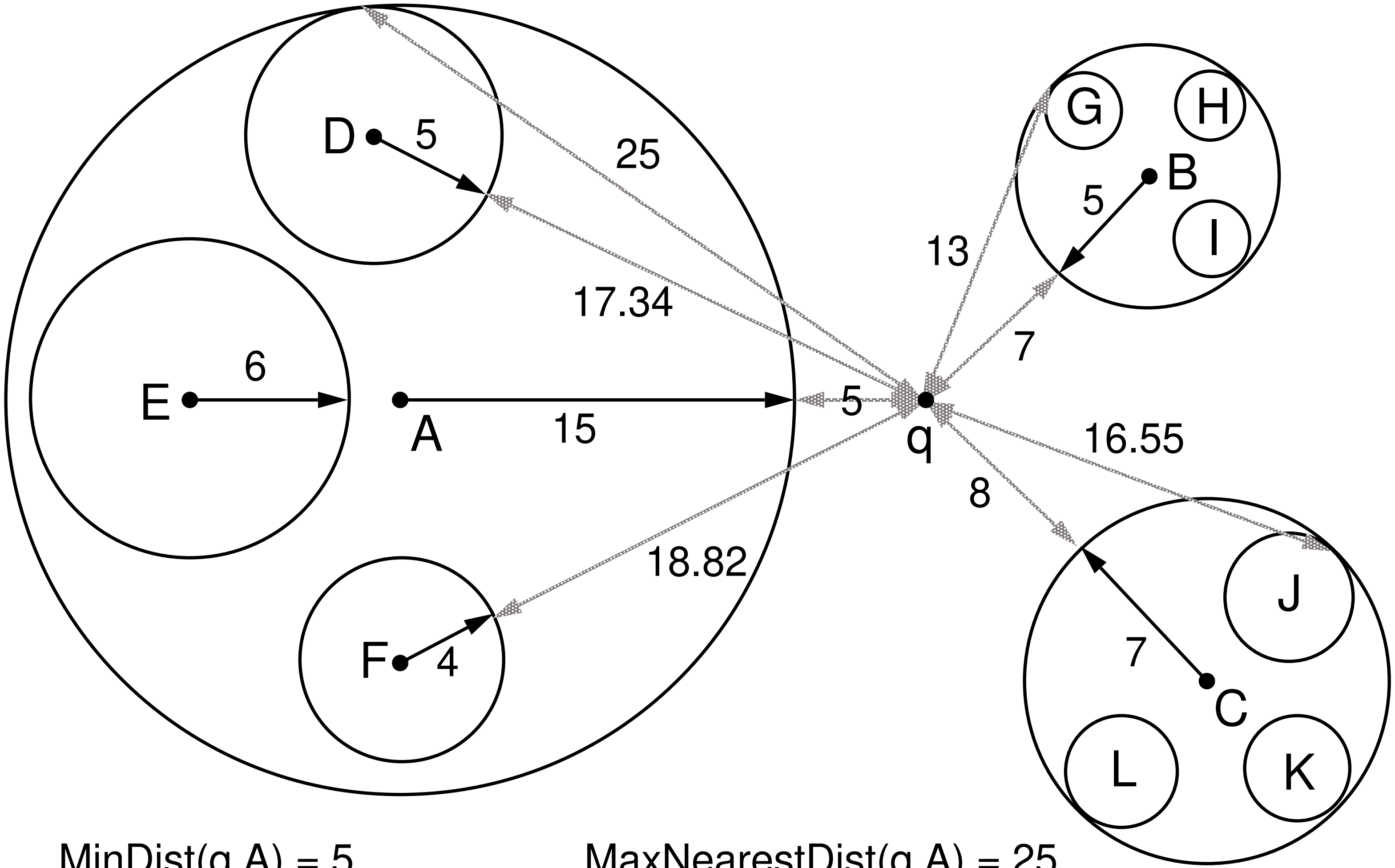
1. If MAXNEARESTDIST$(q, e) > D_k$, do nothing as impossible for e to be in L, and thus guaranteed that e was implicitly removed from L
2. Otherwise, if several elements in L with distance D_k, don't want to needlessly search for e as may be the case if e had already been implicitly removed from L by virtue of the insertion of a closer object or a nonobject with a smaller MAXNEARESTDIST value
   - avoid search by adopting convention that objects have precedence in terms of nearness over nonobjects
   - implies that if insertion into full priority queue L and dequeue one nonobject at a given distance d, then dequeue all nonobjects at the same distance
   - D_k is reset only if exactly one entry has been dequeued and the distance of the new MAXPRIORITYQUEUE(L) entry is less than D_k

Advantage of Expanding L to Also Contain Nonobjects

1. Otherwise, when L contains h ($h < k$) objects, then all remaining entries in L (i.e., L_i ($h < i \leq k$) are ∞
2. Therefore, as long as the remaining $k - h$ entries in L correspond to some nonobjects, we have a lower bound D_k than ∞
3. Nonobjects in L often enable us to provide a lower bound D_k than if all entries in L were objects
   - this is the case when have nonobjects with smaller MAXNEARESTDIST values than the k objects with the k smallest distance values encountered so far
4. Can use estimator value at a deeper level than the one at which it is calculated
5. Enables using MAXNEARESTDIST value of an unexplored nonobject at depth i to aid in pruning objects and nonobjects at depth $j > i$
6. better than conventional depth-first algorithm where MAXNEARESTDIST value of a nonobject element at depth i could only be used to tighten the distance to the nearest neighbor (i.e., for $k = 1$), and to prune nonobject elements at larger MINDIST values at the same depth i

Example

1. Use of MAXNEARESTDIST results in not needing to explore clusters D, E, and F for $k = 2$



MinDist(q,A) = 5
MinDist(q,B) = 7
MinDist(q,C) = 8

MaxNearestDist(q,A) = 25
MaxNearestDist(q,B) = 13
MaxNearestDist(q,C) = 16.55

Conclusions

1. Using MaxNearestDist estimator in depth-first k-nearest neighbor algorithm provides a middle ground between a pure depth-first and a best-first k-nearest neighbor algorithm
2. For N data items, the priority queue implementation of L in the MaxNearestDist depth-first k-nearest neighbor algorithm behaves similarly to the priority queue Q in the best-first k-nearest neighbor algorithm
   - except that the upper bound on L's size is k, while the upper bound on Qś size is $O(N)$
3. Worst-case storage requirements are independent of use of MaxNearestDist estimator:
   - depth-first: maximum height of search hierarchy $O(\log N)$)
   - best-first: size of data set $O(N)$
4. Can adapt best-first k-nearest neighbor algorithm to use MaxNearestDist estimator
   - does not lead to more pruning or faster algorithm, BUT
   - reduces number of nonobject elements that need to be retained in the priority queue