Vector Approximation (VA) file

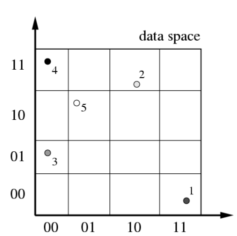
- Tile d-dimensional data-space uniformly into 2^b rectangular cells.
- b bits for each approximation
- Dimension i is partitioned into 2^{b_i} partitions; this requires b_i bits:

  $$b = \sum_{i=1}^{d} b_i.$$

Vector Approximation (VA) file

- A fixed number of bits in each dimensions (8)
- 256 partitions along each dimension
- 256^d tiles
- Approximate each point by corresponding tile
- Size of approximation = 8d bits = d bytes
- Size of each point = 4d bytes (assuming 4 bytes per dimension)

Example

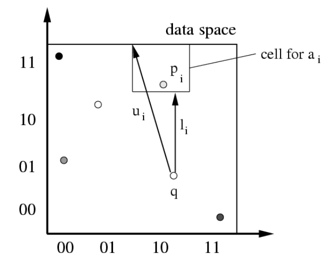


	vector data	VA-File
1	0.9 0.1	11 00
2	0.6 0.8	10 11
3	0.1 0.4	00 01
4	0.1 0.9	00 11
5	0.3 0.7	01 10

Search using VA-file

- VA file is an array of compact, geometric approximations.
- Approximations are scanned, and upper and lower bounds for points are found.
- After pruning on these bounds, remaining points are read.

Computing bounds



Simple k-NN searching

- δ = distance to kth NN so far
- For each approximation ai
  - If lb(q,ai) < δ then
    - Compute r = distance(q,vi)
    - If r < δ then
      - Add point i to the set of NNs
      - Update δ


```
VAR ans:  ARRAY OF INT; dst: ARRAY of REAL;
    a: ARRAY of Approx;  p⃗: ARRAY of Vector;

FUNC InitCandidate(): REAL;
VAR j: INT;
   FOR j := 1 TO k DO
      dst[j] := MAXREAL;
   RETURN MAXREAL;
END-FUNC InitCandidate;

FUNC Candidate(δ: REAL; i: INT): REAL;
   IF δ < dst[k] THEN
      dst[k] := δ; ans[k] := i;
      SortOnDst(ans, dst, k);
   RETURN dst[k];
END-FUNC Candidate;

PROC VA-SSA(q⃗: Vector);
VAR i: INT; l_i, δ: REAL;
   δ := InitCandidate();
   FOR i := 1 TO N DO
      l_i := GetBounds(a_i, v⃗_q);
      IF l_i < δ THEN
         δ := Candidate(L_p(p⃗_i, q⃗), i);
   END-FOR;
END-PROC VA-SSA;
```


Evaluation

- Advantages:
  - Simple
  - Low memory overhead
  - Everything processed sequentially: no random seeks.
- Disadvantages:
  - Performance depends on the ordering of vectors

Near-optimal NN searching

- δ = distance to kth closest ub(q,a) so far
- For each approximation ai
  - Compute lb(q,ai) and ub(q,ai)
  - If lb(q,ai) ≤ δ then
    - If ub(q,ai) < δ then update δ
    - InsertHeap(lb(q,ai),i)

/* First phase */

Near-optimal NN searching (2)

/* Second phase */

- δ = distance to kth NN so far
- Repeat
  - Examine the next entry (li,i) from the heap
  - If δ < li then break
  - else
    - Compute r = distance(q,vi)
    - If r < δ then
      - Add point i to the set of NNs
      - Update δ

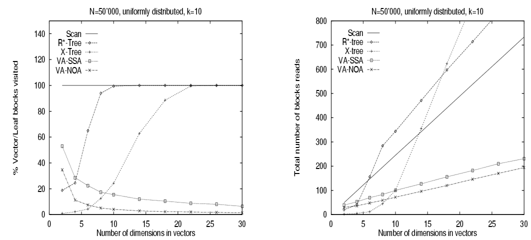

```
PROC VA-NOA(q⃗: Vector);
VAR i: INT; δ, l_i, u_i: REAL; Heap: HEAP; Init(Heap);

(* PHASE - ONE *)
  δ := InitCandidate();
  FOR i := 1 TO N DO
    l_i, u_i := GetBounds(a_i, q⃗);
    IF l_i <= δ THEN
      δ := Candidate(u_i, i);
      InsertHeap(Heap, l_i, i);
  END-FOR;

(* PHASE - TWO *)
  δ := InitCandidate();
  l_i, i := PopHeap(Heap);
  WHILE l_i < δ DO
    δ := Candidate(L_p(p⃗_i, q⃗), i);
    l_i, i := PopHeap(Heap);
  END-WHILE;
END-PROC VA-NOA;
```


Complexity

- After first phase, 95-99% of candidates are pruned
  - Remaining number of candidates are empirically estimated to be sub-linear (log n)

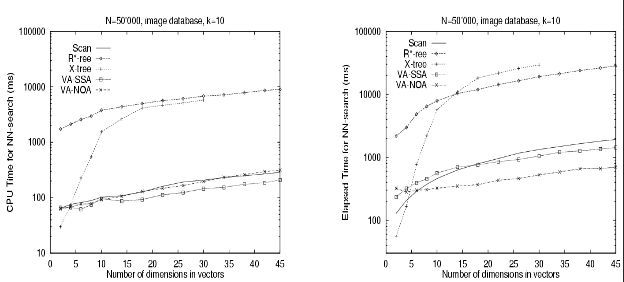
Experimental setup

- Data sets:
  - Synthetic: uniformly distributed data points
  - Image data: 45-dimensional 50,000 vectors
- Machine:
  - Sun SPARCstation 4
  - 85 Mhz CPU
  - 64 MB RAM



(a) (b)

Figure 10: (a) Block selectivity and, (b) the total number of blocks accessed as a function of number of dimensions. Synthetic data set (uniformly distributed).



(a) (b)

Figure 12: CPU time (a) and wall-clock time (b) for the image database

References

- When Is "Nearest Neighbor" Meaningful? K. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft, ICDT 1999, pp. 217-235
- A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces, R. Weber, H-J. Schek, S. Blott, VLDB 1998, pp. 194-205
- Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases, C. Böhm, S. Berchtold, D. Keim, ACM Computing Surveys, Volume 33, Issue 3, September 2001, pp. 322 - 373