

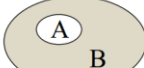



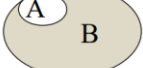



The 8 Spatial Relationships (I)

$$\begin{pmatrix} A_b \cap B_b & A_b \cap B_l & A_b \cap B_e \\ A_l \cap B_b & A_l \cap B_l & A_l \cap B_e \\ A_e \cap B_b & A_e \cap B_l & A_e \cap B_e \end{pmatrix}$$

 disjoint	 contain	 inside	 equal	$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ meet	$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ cover	$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ covered_by	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ overlap
$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ disjoint	$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ contain	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ inside	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ equal	 meet	 cover	 covered_by	 overlap

R-tree and R+-tree

Spatiotemporal data - HR-tree, TPR-Tree

Lock-step, LCSS

Vertical line query

Segment tree

Quadtree

B/B+ trees

IDistance

Z-value

Space-filling curve

kNN query/ skyline

Hashing

Dimensionality curse

VA-file

Filter-and-refine strategy

Data abstraction

Precision and recall

R-tree and R+-tree

To use **R+-tree to index** a set of polygons, which statement below is correct?

- A polygon may appear in different sub-trees.
- When an **R+-tree** is used to process a point query, it not possible to search more than one subtree at the same level.

Compare and contrast **R-Tree** and **R+-Tree** when they are used to index polygons.

R-Tree allows node overlap of the internal MBRs while R+-Tree does not. R-Tree only stores an object once, while R+-Tree will store that object in multiple leaf nodes.

The **R+-Tree** “clips” an object so MBRs do not have to overlap. This occurs during the split algorithm. The MBRs in R+-Tree are not physically “clipped”.

Adv of R-Tree: Single search path for the point query.

Dis of R-Tree: May exams all the MBRs at all levels. Because the MBRs may overlap, and the space is not disjointly decomposed.

Adv of R+-Tree: A point query has a single unique path down the tree. It has less overlap so the queries may access less nodes.

Dis of R+-Tree: It stores an object more than once. For a window query, it may return the same object more than once. It has more complex split algorithm, and sometimes leads to deadlock.

Briefly explain the reason underlying the “cell overlapping” feature of **R-trees**.

R-trees form a tree of minimum bounding rectangles for the points/polygons they index. It is impossible to form these minimum bounding rectangles without some overlap between them (unless the R-tree is trivial, which makes it useless). There are many algorithms to decide the best way to make the MBRs.

Briefly discuss the impact of cell overlapping on Search and Insert operations in **R-trees**.

For the search operation cell overlapping causes multiple cells (and thus branches) to be searched to find the spatial object. This is because it is contained in only in one cell, but many cells may overlap with each other.

Inserting may require many parent cells to be expanded, which slows down the operation significantly. An object can also only be inserted into one cell, so the DBMS must further decide which cell to put it in and this requires even more computation time.

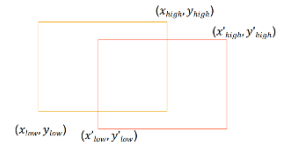
R-tree index can be used to index polygon data. Explain how an R-tree index on a polygon dataset can be used to speed up the processing of a *window query* (i.e. to find all polygons with a given rectangular window).

In an R tree, each leaf is an MBR approximation of the polygon it points to. Each internal node points to several children (which could be the leaves) and has an MBR approximation of these children. Thus, a tree is formed where the root node is a large rectangle encompassing all children and each level down grows becomes a better approximation.

So, R trees can speed up the processing of a window query as the filter step in the query process can disregard any MBRs (and their children) that do not intersect with the window.

Rectangle intersection is one of the most essential operation in the **R-Tree** window search. How can we do it efficiently?

One straightforward way to test if two rectangles intersect is reducing this problem to the point-in-rectangle test. However, it is hard to know which point to choose for the testing. If one rectangle is inside another one, then no point along the outside rectangle is inside the inner one. Or if two rectangles intersect like a crossing, none of the eight endpoints of the two rectangles are inside the others.



Therefore, to solve this problem more efficiently, we must first check its dual problem: when are the two rectangles disjoint? The first case is when one rectangle is above another one. Therefore, one rectangle's ylow should be larger than the other's yhigh. The below case is similar.

The second case is when one rectangle is on the right hand of the other one. Therefore, one rectangle's xlow is larger than the other's xhigh. The left case is the also similar.

In this way, we have the conditions when two rectangles are disjoint:

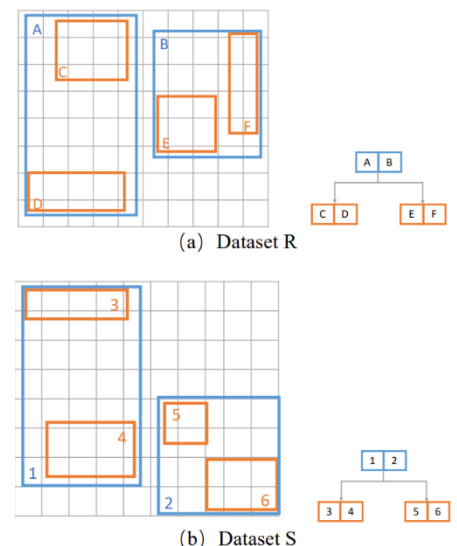
- $y'_{low} > y_{high}$
- or $y_{low} > y'_{high}$
- or $x_{low} > x'_{high}$
- or $x'_{low} > x_{high}$

So, when any of the above conditions is true, it is guaranteed that two rectangles do not intersect. When none of these four conditions is satisfied, we say the two rectangles intersect. So, it takes at most four comparisons.

Nested Loop Spatial Join with R-Tree. The following is an example of two datasets R and S indexed by two R-Trees. Suppose we are doing spatial join on the polygon intersection. Please describe how to use the synchronize traversal to do the nested loop join algorithm.

During the synchronized traversal, we visit the two R-Trees level by level and keep a task list. Suppose the task list is T, and the candidate list is C. Below are the detailed steps:

1. We visit the first level of the two trees. The root of R has two MBRs A and B, and the root of S has two MBRs 1 and 2. We add the cartesian product of these two sets of MBRs into the task list:
 - a. $T = \{(A,1), (A,2), (B,1), (B,2)\}$
 - b. $C = F$
2. We test the first pair (A,1) in T. Because A and 1 intersect, we need further check their children. A has two children C and D, and 1 has two children 3 and 4. We add the cartesian product of them into the task list:
 - a. $T = \{(A,2), (B,1), (B,2), (C,3), (C,4), (D,3), (D,4)\}$
 - b. $C = F$
3. We test next pair (A,2) in T. Because they do not intersect, we skip them directly.
 - a. $T = \{(B,1), (B,2), (C,3), (C,4), (D,3), (D,4)\}$



- b. $C=F$
4. We test next pair (B,1) in T. Because they do not intersect, we skip them directly.
 - a. $T=\{(B,2), (C,3), (C,4), (D,3), (D,4)\}$
 - b. $C=F$
5. We test next pair (B,2) in T. Because B and 2 intersect, we need further check their children. B has two children E and F, and 2 has two children 5 and 6. We add the cartesian product of them into the task list:
 - a. $T=\{(C,3), (C,4), (D,3), (D,4), (E,5), (E,6), (F,5), (F,6)\}$
 - b. $C=F$
6. Now all the tasks in the list are in the leaf level. We test them one by one and add the intersected ones in to the candidate list:
 - a. $C=\{(C,3), (D,4), (E,5)\}$

After obtaining the candidate list, we visit these MBRs, retrieve the actual objects, and test if they intersect or not.

When we do the **hash based spatial join**, we have two choices of hash functions: *single assignment* and *multiple assignment*, and two join procedures: *single join* and *multiple join*. Different index would require different join procedure. Please discuss which kind of join procedure should be used by **R-Tree**, **Z-order**, and **R+-Tree**, and explain why.

For R-Tree, each object is enclosed by only one MBR, so it belongs to the single assignment. When we use the single assignment hash function, we must run the multiple join procedure to guarantee the correctness. Therefore, to get the result, we must join all the intersected objects together.

For Z-order and R+-Tree, one polygon can exist in several cells or MBRs, so it belongs to the multiple assignment. When we use the multiple assignment hash function, we only need to join once for each object to obtain the result. Otherwise, we will produce duplicate results.

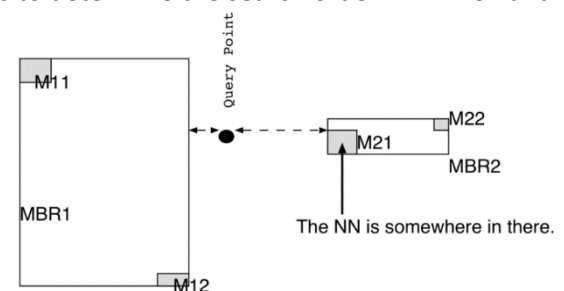
In an **R-tree**, briefly describe the “**MinDist**” and “**MinMaxDist**” parameters.

MinDist is the absolute minimum distance from a point/rectangle to the corresponding MBR. MinMaxDist is the minimum of the maximum distances of each closest face in each dimension (it is the minimum upper bound).

When using the **R-tree** to search for the **nearest neighbour**, we have two metrics to determine the search order: **MINDIST** and **MINMAXDIST**. The MINDIST is the optimistic choice, while the MINMAXDIST is the pessimistic one. Is the MINDIST always a better ordering than using the MINMAXDIST?

MINDIST ordering is not always better than MINMAXDIST ordering.

As shown in the above example, the nearest neighbour is somewhere inside M21. If we use the MINDIST ordering, because MBR1 is nearer than MBR2, we will visit MBR1 first and visit M11 and M12 in it. If we use the MINMAXDIST ordering, because MBR2 has smaller MINMAXDIST than MBR1, we will visit MBR2 first and then M21. When we visit MBR1, we can prune M11 and M12.



When is this condition necessary and when can we use a **tighter pruning condition**?

When there are many dead spaces in the nodes of the R-tree, we must use **MINMAXDIST** as the pruning threshold because we cannot afford losing the correct results.

When there is no or very little dead space in the nodes of R-tree, using **MINDIST** is a better choice as it will prune more candidate MBRs and we can find the result earlier.

Explain the role played by the above parameters in optimising the processing of **Nearest Neighbour queries**.

It can clearly be seen that given a minimum distance from a point and a minimum maximum distance from a point, the nearest neighbour must be somewhere between these. That is: $\text{MinDist}(p, R) \leq \text{NN}(p) \leq \text{MinMaxDist}(p, R)$

Given this property, we can do pruning based on the following -

Downward pruning. An MBR R is discarded if there is an MBR R' such that $\text{MinDist}(p, R) > \text{MinMaxDist}(p, R')$. The same applies for objects (but actual distance instead of minimum). That is, R' is closer than R.

Upward pruning. An MBR R is discarded if $\text{MinDist}(p, R) > d(p, o)$ (o is closer than R)

Spatiotemporal data - HR-tree, TPR-Tree

A set of **spatiotemporal data** in a geographical space can be viewed as 3D data, where the three dimensions are (x, y, t) with (x, y) for geographical locations and t for time.

- (1) Give an example to show that this approach can be highly **inefficient**. (Why 3D-Tree is not efficient for **trajectory data**)
3D R-tree has three problems: 1. The t-dimension is unbonded. 2. It's not effective to use boxes to approximate lines. 3. It is only efficient for coordinate-based queries, but not efficient for trajectory-based queries.
- (2) **Historical R-tree (HR-tree)** is proposed to index trajectory more naturally. Please briefly describe the key ideas and benefits of HR-Tree.
HR-tree allows that trees at consecutive timestamps to share branches if nothing changes to save space. HR-tree is an R-tree that is maintained for each timestamp in history. Two HR-tree indexing different snapshots may share branches if the objects are static for two snapshots.
- (3) Give an example to show that **HR-tree** can also be highly inefficient for some datasets or queries.
HR-tree can be highly inefficient such that little can be shared. We may need many HR-trees to index many snapshots if the objects are moving very fast. Because the position of the objects of two close time snapshots can be very different for fast moving objects. Then few branches may be shared, and this leads to the index being particularly large.
- (4) What is the basic idea of **TPR-Tree** and what is its benefit?
TPR tree is a suitable way for queries with a time-interval and has smaller overlapping ratio by considering the current location and future location of objects. Because an MBR can grow with time and we can estimate its future location to minimize overlapping. If we consider both the locations and the prediction of the future location, we can have a better choice by redesigning the MBRs to minimize the overlap.

Lock-step, LCSS

It is a fundamental operation to measure **trajectory similarity**. Explain and discuss the following two methods.

- (1) What is the **lock-step** window-based Euclidean distance?
Lock-step window-based Euclidean distance: the kth point of trajectory is aligned to the kth point of the other trajectory.
 - Use a sliding window based on the shorter trajectory.
 - The distance is the best of window-based total Euclidean distance.(we select the mapping of points with the smallest Euclidean distance)
- (2) What is **LCSS**?
LCSS is the **Longest Common Sub-Sequence**.
We first define a threshold. If the Euclidean distance of two points is lower than that threshold, we say that points are similar or close enough to be the same. Then we count the number of points from two trajectories that are the same to measure the trajectory similarity. (similar points do not need to be consecutive)
- (3) Which method above is more computationally efficient? Why?
Lock-step is faster but at the cost of affecting by noise.
- (4) Which one is more robust to data noise? Why?
LCSS. In LCSS, two points are similar when they are within a certain threshold. The effect of noise can be mitigated by adjusting the threshold. However, in lock-step, the similarity is defined as Euclidean distance between two points, which can be impacted by noise.

Vertical line query

What is the query time of answering the **vertical line query** using an **interval tree**? What is the query time of answering vertical segment query?

The vertical line query takes $O(\log n)$ time to visit at most $\log n$ crossed nodes. Because all the line segments in the crossed node are retrieved in the pre-sorted order, so only k (the number of the results) will be visited. The total query time is $O(\log n)$.

The vertical segment query must further check the y dimension, so it uses the 2D Range tree instead of the sorted list to index the end points. If the 2D Range Tree uses a binary tree as the auxiliary structure same as the lecture note, the search of 2D Range tree takes $O(\log^2 n + k_u)$ time, and the total time complexity is $O(\log^3 n + k)$. If the 2D Range tree uses the fractional cascading on the auxiliary structure, it takes a $O(\log n + k_u)$ time, so the total time complexity is $O(\log^2 n + k)$.

Segment tree

The **segment tree** organizes the segments based on their end points. The line segments are stored at any nodes u that it covers

it entirely while it doesn't cover u's parent's region. Why it cannot appear in the same level for more than two times?

The line segment is continuous, so it will cover the nodes continuously.

Firstly, for a level of nodes, the segment left end could only appear in its parents' right child, and the right end could only appear in its parent's left child. Otherwise, the segment will store in its parent instead.

Secondly, for any nodes between the left end and the right end, they and their siblings are all covered by the segment, so the segment is stored in their ancestors.

Therefore, for each level, only the two ends are possible to store the segment.

Compare how to answer the **line segment range query** with **interval tree**, **priority search tree**, and **segment tree**.

The result line segments have two types: the ones with at least one endpoint in the rectangle, and the ones have no endpoint in the rectangle. The first kind can be answered with 2D range tree, while the second kind needs further computation.

Interval tree and priority tree are used to find the second kind of segments that are parallel to the axis. We need to build one tree for each axis. The interval tree itself takes $O(n)$ space. To answer the vertical segment search instead of vertical line search, the interval tree uses another set of 2D range trees within each node to achieve $O(\log^3 n + k)$ complexity (or $O(\log^2 n + k)$ with fractional cascading). If we use the priority search tree to organize the segments within each node, it takes $O(\log n + k)$ to answer, so the overall time complexity is also $O(\log^2 n + k)$. However, the 2D range tree takes $O(n \log n)$ space, while the priority search tree version takes $O(n)$ space.

The segment tree can deal with the case when the segments are not required to be parallel to the axis. It takes $O(\log^2 n + k)$ to report results and takes $O(n \log n)$ space.

Quadtree

Quadtree is a tree structure in which each internal node has exactly four children. In a quadtree, each node represents a bounding box covering some part of the space being indexed, with the root node covering the entire area. *Is quadtree a balanced tree? What is the worst case of the two quadtrees? What's the main difference between Point Quadtree and Region Quadtree?*

Quad tree is not a balanced tree. The point quadtree's shape depends on the order in which points were inserted. The region quadtree's shape is affected by the point distribution.

The worst case of building a quadtree is linear. It takes $N(N-1)/2$ times of comparison because the i^{th} point requires $i-1$ comparison operations.

The main difference:

Point Quadtree

- The tree is constructed based on the points, its shape and size are dependent on the insertion order
- Data points can be stored in non-leaf nodes
- The space spanned by the point quadtree is rectangular and can be of infinite width and height
- Deletion is hard

Region Quadtree

- The tree is constructed based on regular decomposition of the space, and its shape and size are independent on the insertion order
- Data points are only stored in the leaf nodes
- The space spanned by the region quadtree is constraint to a maximum width and height
- Deletion is simple

How should we delete a non-leaf node (like root) in **point Quadtree**?

Choice one: re-insert all points of the sub-tree rooted at the deleted point; choice two: replace the deleted node A and re-insert the affected areas: - hatched; - determine four candidates for the replacement (select best)

Briefly explain the principle underlying the design of a **Region Quadtree** for indexing spatial data points.

A quadtree is a hierarchical data structure where each node corresponds to the NW, NE, SW, and SE cells. A region quadtree is where regular decomposition for the cells is used.

The idea is that points are added to the leaves of the tree and when it gets too full that node is split into a further 4 cells (and the points are then placed appropriately).

Is a **Region Quadtree** always balanced? Please briefly explain your answer.

No, it often wouldn't be. As data is not usually uniformly distributed, you would have some branches that are very decomposed

and others that aren't at all (i.e. the NE section of a dataspace has many points, but the SW has none).

B/B+ trees

An attractive choice for indexing multi-dimensional data is an index that uses **B or B+ trees**. Briefly explain:

(a) The general principle underlying the idea of utilising **B/B+ trees** to index multi-dimensional data.

The general principle is to convert the multi-dimensional data space into one dimension (usually using some form of a space filling curve) and then indexing this in a B/B+ tree.

(b) Name an efficient method for indexing *spatial data points* that used a **B/B+ tree**.

A **z-value index**. It converts a two-dimensional data space to a single dimension by using a space filling curve which follows the pattern SW, NW, SE, NE. A B/B+ tree indexes these cells which each contains a bucket of points (i.e. the leaves will point to a bucket of points).

(c) Name an efficient method for indexing *high dimensional data points* that uses a **B/B+ tree**.

iDistance is an efficient method for indexing high dimensional data points. It assumes that most data are clustered and so it partitions all the points into clusters which have a reference point. Each point is indexed based on their distance to this reference point (using a B/B+ tree).

iDistance

In **iDistance**, the number of partitions affects its indexing performance. What are the possible problems when the number of partitions is too small or too large?

A small number means more points will have the similar distances. This means that the refine step in spatial processing will still have many points to deal with after the filter step.

A large number means more overlap and thus incurs additional traversal and searching.

In **iDistance** method, explain the problem called "over search"?

Either:

- Many points that aren't even close to the given point are returned due to only a distance from a reference point being used (so it gives a huge area instead of the small circle you care about)
- Or, having to look in multiple clusters as the nearest point may be in a difference cluster

Z-value

Many spatial indexing methods used in a 2D space can also be used in a higher dimensional space.

(1) How can we obtain a **Z-value** for a point in a 3D space? Please provide one method with detailed description.

We use line (or polyline) to do recursive decomposition on 2D space to obtain Z-value. We can use polygons to do recursive decomposition on 3D space to obtain Z-value.

We start with recursively decomposing the 3D space to cubes of equal volume. The number each partition. The space is decomposed into finer parts where there are more points. If the point is within a partition, then the number of this partition is appended to the z-value of the point. This is done recursively until the number of the number of the smaller partition the point lies in is appended to the z-value of the point.

(2) For two **Z-values** x and y obtained in a 3D space, what is the spatial relationship between the cells represented by them in the following two cases respectively? [A spatial relationship can be identical, inside, overlap, disjoint etc.]

- When $x = y$
x identical y or (x equal y)
- When x is a prefix of y
x covers y

Let R and S be two sets of polygons. A polygon is approximated using several **Z-values**. So, we have the following two sets of (Z-value, polygonID) pairs where a polygon is associated with the **Z-values** that are used to approximate the polygon:

$X = \{(z, \text{pid}) \mid z \text{ is a Z-value and pid is a pointer to a polygon in R}\}$

$Y = \{(z, \text{pid}) \mid z \text{ is a Z-value and pid is a pointer to a polygon in S}\}$

- (1) Please give a step-by-step execution plan following the **sort-merge join paradigm** to identify candidates in the form of $\{(\text{pid1}, \text{pid2}) \mid \text{pid1 in R, pid2 in S, and pid1 and pid2 have at least one pair of their Z-values overlap}\}$.

1. We sort the z-values of R&S in increasing order; 2. We visit both these two sorted lists from the smallest values at the same time; 3. After the procedure of the sort-merge, we have obtained a candidate list. We might do some housekeeping, because some pairs are duplicated; 4. We need to retrieve and test the actual intersection ones.
- (2) Once the candidates are generated by the algorithm above, how to get the final results $\{(pid1, pid2) \mid pid1 \text{ in } R, pid2 \text{ in } S, \text{ and } pid1 \text{ and } pid2 \text{ do intersect}\}$?
We need to do housekeeping check of the candidate list. Because some pairs may be duplicated. Finally, we need to retrieve and actually test each pair if they intersect. **Polygon intersection**: step1: MBR intersection: filter the impossible; step2: point in polygon testing; step3: polyline intersection.
- (3) Typically, it can be more efficient to perform a join operation on X and Y as a filter step first as compared to perform **polygon intersection join** directly between R and S. Why?
Yes. Because the first filter step use approximation and index to quickly find a candidate list. After removing duplicates in this candidate list, we only need to perform a fill test to this candidate list, which can save CPU cost and I/O cost. Step 2 & 3 mentioned above are computational costly. However, obtaining the candidate set and filtering out unnecessary candidates using sort-merge algorithm is computational cheap.

Space-filling curve

What is the purpose of **space-filling curve**? Map the partitioned cells of a space in a one-dimensional list.

Explain briefly how **space-filling curves** can be used for spatial indexing of points and polygons.

Space-filling curves are an attempt to convert data into a single dimension while maintaining “closeness”. There are many different types but z-order is the most used (SW, NW, SE, NE)

A space filling curve can be used to convert 2D cells into a 1D line. This then be indexed by a B+-tree. Points would correspond to a single cell, whereas a polygon may correspond to many.

kNN query/ skyline

What is a **kNN query**? What is a **skyline query**? Compare and contrast these two types of queries.

A **kNN (k nearest neighbours)** query returns the closest k neighbours to a given point. A **skyline query**, on the other hand, returns a set of points that are not dominated by any other points. kNN queries can be utilised in performing a skyline query.

kNN is useful in that it returns the points that are most like the given point. There are also a few variations of kNN such as reverse NN, FN, and reverse FN.

Skyline queries are useful because they provide answers to minimisation queries such as “find hotels that are cheap and closest to the beach”.

Why the **skyline** result will appear in all monotonically increasing function?

The property of a k -dimensional monotonically function f is that for any two values $f(x_1, x_2, \dots, x_k)$ and $f(x'_1, x'_2, \dots, x'_k)$

- If $\forall 1 \leq i \leq k, x_i \leq x'_i$
- And $\exists 1 \leq i \leq k, x_i < x'_i$
- $\Rightarrow f(x_1, x_2, \dots, x_k) \leq f(x'_1, x'_2, \dots, x'_k)$

No matter which dimension, if its value increases, the function value increases.

The requirement of skyline result is for any two k -dimensional points

$p(x_1, x_2, \dots, x_k)$ and $p'(x'_1, x'_2, \dots, x'_k)$

- $\forall 1 \leq i \leq k, x_i \leq x'_i$
- $\exists 1 \leq i \leq k, x_i < x'_i$
- $\Leftrightarrow p$ dominates p'

The two conditions are the same as the increasing condition of the monotonically function. Therefore, we can get the following equation:

- $p \text{ dominates } p' \Rightarrow f(p) \leq f(p')$

Because the points in the skyline result set P dominate all the other points P' , no matter how we change the actual function f' , a point $p' \in P'$ still can find some skyline points $p \in P$ that has smaller function value $f'(p) < f'(p')$. Otherwise, the skyline result is not complete.

Skyline computation is very powerful to identify a set of results that satisfy all the monotonically increasing preference functions for multi-criteria optimization. Its power does not limit to the kNN search but can also extend to other applications.

For example, in the **route planning in road network**, we mostly consider the distance d as an optimization goal. However, other factors like toll charge, and travel time are also very important. Therefore, it would be beneficial if we could provide users with a set of **skyline paths** instead of only one path.

Suppose we have a graph $G(V, E)$, where V is the vertex set denoting the intersections, and E is the edge set denoting the roads. For each edge (u, v) of E , it has two weights: length $d(u, v)$ and cost $c(u, v)$. Suppose the starting vertex is s and the destination vertex are t .

- (1) Define the **skyline shortest path problem** using the above notations.

Skyline: informally, a point dominates another point if it is as good or better in all dimensions and better in at least one dimension.

Skyline for this question: a vertex dominates another vertex if the length and cost of this vertex is as good or better than another vertex, and at least length is shorter, or cost is smaller.

$V1$ dominates $V2$ iff: $V1.d(u,v) \leq V2.d(u,v)$ and $V1.c(u,v) \leq V2.c(u,v)$ and $(V1.d(u,v) < V2.d(u,v) \text{ or } V1.c(u,v) < V2.c(u,v))$

- (2) Please give a sketch of how to solve this problem. Either in plain English or pseudo-code.

Function Dijkstra (G , source):

For each v in V :

Length[v]: infinity

Cost[v]: infinity

Previous[v]: undefined

length[source]: 0

cost[source]: 0

Q : the set of all nodes in G

while Q is not empty:

u: node in Q with smallest length & smallest cost

remove u from Q

for each neighbour v of u :

alt.len = length[u] + length_between(u, v)

alt.cost = cost[u] + cost_between(u, v)

if as least alt.len or alt.cost \leq length[u] or cost[u]

length[v] = alt.len

cost[v] = alt.cost

previous[v] = u

return previous []

Consider a k -dimensional dataset D with each record in the form of (d_1, d_2, \dots, d_k) , where each d_i is a numerical value. Define a preference as a set of weights (w_1, w_2, \dots, w_k) , where each $0 \leq w_i \leq 1$ and $w_1 + w_2 + \dots + w_k = 1$ (it is called a preference as these weights define the preferred relative importance for these dimensions). Once a preference is given, the best record in D can be defined as the record in D with the smallest $d_1 * w_1 + d_2 * w_2 + \dots + d_k * w_k$ value. However, such preferences may not be known by a user to choose the best result. Therefore, the skyline operator is introduced.

- (1) Define the **skyline operator** using the notations above.

Dominates: suppose p, q are two k -dimensional points in D , p dominates q iff: (1) $\forall 1 \leq i \leq k, p_i \leq q_i$ and (2) $\exists 1 \leq i \leq k, p_i < q_i$. In other words, a point dominates another point if it is as good or better in all dimensions and better in at least one dimension.

- (2) No matter what preferences a user may have, the best record to be selected based on any given preference will always be in the skyline results. Is this statement correct? Why?

Correct. **Skyline** query is to find all points from a dataset which are not dominated by any other points. If the best record is not in the skyline, which means it can be dominated by another point. It is a contradict with the "best" record

Hashing

Static hashing and linear hashing are two different ways to access one dimensional data. Explain the way how these two methods deal with 'collision' when inserting a new record.

With static hashing, we push the record into an overflow bucket and add a pointer from the original bucket (i.e. the bucket it should have gone in to without collision) to the record in the overflow bucket. Linear hashing also uses chaining and overflow buckets, but a bit more complex. For linear hashing, we need to keep track of m = the number of buckets, n = the index of the bucket to split next (0 at start). On collision, we push the record to an overflow bucket. We add a new bucket to the end (increasing m). We then split bucket n and rehash all its contents using $h(k) = k \bmod (2m)$. We then increase n .

This doesn't give any guarantee that the overloaded bucket is split.

Dimensionality curse

A common multimedia search approach is to represent a multimedia object such as an image as a point in a high dimensional feature space, such that a query to find similar objects to a given query object is formulated as to find the **nearest neighbours** of the query object in the feature space. Discuss possible issues related to the problem of “**dimensionality curse**” for this type of approach.

The **dimensionality curse** is where performance degrades rapidly as dimensionality increases due to almost all points in high dimensional space being “far away” from the centre. Also, as the dimensionality increases, the distance to the nearest neighbour approaches the distance to the furthest neighbour. (another two reasons for causing curse: *All tree-based indexing structures examine a large fraction of leaf nodes due to the heavy overlap; As dimensionality increases, the number of results in a skyline query grow exponentially*)

Because of this last point, a nearest neighbour query will:

1. Be quite slow
2. Likely not provide the answer that is wanted (this is because as the dimensionality increases the “difference” in points becomes much smaller)

Thus, a **nearest neighbour query** is not all that meaningful in high dimensional space (assuming the data is uniformly distributed).

If the data is not uniformly distributed and exhibits clusters, trends, or skewness then methods such as **iDistance** can be employed to speed up the result.

“**Dimensionality curse**” is a well-known problem in high dimensional space.

(1) Please explain the problem of “**dimensionality curse**”, and briefly discuss its impact on the **space-centric index**, the **object centric index**, and the **nearest neighbour search** in a high dimensional space.

The performance of an index degrades rapidly as dimensionality increases, and eventually underperformance linear scan. There are three reason:

- Too many partitions: the number of partitions grows exponentially as the dimensionality number increase, if the dimensionality number is very large, we will have more partitions than points.
- Too few points: when data is uniformly distributed, most of them are in the boundary regions, leaving the central space as a big hollow.
- The nearest is not near enough: it’s hard to distinguish the distance of objects. In high-d space, the distance between the nearest neighbour and the farthest neighbour becomes nearly the same.

(2) If the dimensionality curse is so severe, why we still can organize the data in practice efficiently?

Special method like X-tree, Pyramid technique, iDistance, VA-File can help us.

VA-file

(3) **VA-File** is a very simple index structure to index the uniformly distributed high-dimensional data. Queries on the VA-file, such as finding the nearest neighbours, make use of a two-stage procedure such as a **filter-and-refine** algorithm. Why is it not necessary to sort the elements obtained in the first stage prior to starting to process them in the second stage?

Phase 1: Filtering

- The VA-file is sequentially scanned.
- The lower and upper bounds on the distance for each object’s approximation (i.e. VA) is then computed.
- Assume δ is the smallest upper bound so far, eliminate approximations with a lower bound that exceeds δ

Phase 2: Refine

- After the filtering step, a small set of candidates remain
- Candidates are sorted by lower bound
- If a lower bound is encountered that exceeds the nearest distance seen so far, the VA-file method stops

Filter-and-refine strategy

Why is the **filter-and-refine strategy** necessary in spatial database query processing?

Spatial operations are expensive in terms of both CPU and I/O costs.

Describe key objectives in the filter phase and the refinement phase of the **filter-and-refine** approach to spatial query processing, and how these objectives can be achieved.

The **filter-and-refine strategy** is a strategy that splits the processing of a query into two phases, the filter phase and the refine phase. The total **cost** to process a spatial operation is reduced using this strategy as the filter phase uses only very simple operations and reduces the result set as much as it can. This means that only very few complex operations must be completed.

Filter: Filter out as many results as possible, leaving only probably results. That is, reduce the resultant set as much as possible. Can be achieved by applying simple operations on approximations of the spatial objects.

Refine: Take the result set from the filter phase and get rid of any false positives. This is achieved through using spatial operations on the actual geometries of the spatial objects (far more expensive than the simple operations in the filter phase).

Approximations

Conservative approximation: containing the object (min)

Approximations don't overlap -> impossible for objects to overlap

Approximations do overlap -> objects may overlap (need to refine)

Progressive approximation: contained by the object (max)

Approximations do overlap -> objects overlap too

Approximations don't overlap -> objects may overlap (need to refine)

Data abstraction

What is **data abstraction**?

Data abstraction is a way to make data more easily searchable. It takes features out of the data which are more compact and easier to search.

Why is **data abstraction** important for multimedia databases?

Without data abstraction all the multimedia data would have to be processed every time. Given the enormous size of such data, this is completely infeasible.

What are the possible image **data abstraction** methods?

Colour (e.g. colour histogram), texture (e.g. grey level co-occurrence matrix), and shape (e.g. blob extraction)

Consider an image database that supports **colour-based similarity search**.

- (1) Describe a method to represent an image using 256 colours.

Use colour histogram: distribution of colour is useful feature for image representation. Each pixel can be described by three colour components. R: 2 bits - 4bins B: 1 bit - 2 bins G: 1 bit - 2 bins

- (2) Define a **similarity function** that can be used to compare the colour features extracted in the above step. Explain the steps of the query with an image.

Database: the image from question 1

Query: the d-dimensional colour feature extracted by colour histogram

Measure: similarity function: normalized distance + similarity = 1. So, all the distance can be used to describe the similarity. The larger distance, the smaller similarity.

- (3) **Precision** is the ratio of the correctly returned ones in the returned results, while the **recall** is the ratio of the correctly returned ones in all the correct ones. How can we achieve high recall? At what cost?

High recall: to retrieve more correct ones, we must get more results, so we must suffer from lower precision.

- (4) How can we increase the precision and recall at the same time?

(consider the relationship between precision and recall) use f-measure. $F = 2 \times \text{precision} \times \text{recall} / (\text{precision} + \text{recall})$

- (5) It is possible, at least theoretically, to index the colour features using a **R-tree**. What problems could we encounter if a R-tree is used here?

Index the colour features using R-tree depends on the data distribution and dimension number. Dimensionality curse could be the problem here: the performance of an index degrades rapidly as dimensionality increase, and eventually underperforms linear scan.

In high dimension, most of the data lies close to the boundary of the dataspace, and as the dimensionality increase, the distance to the nearest neighbour approaches the distance to farthest neighbour.

- (6) In what way can the **VA-file** approach be more efficient?

The total cost of VA-file also includes the random access to the candidate. If the candidate set is large, linear scan will be cost large. Therefore, VA-file performs best in uniform distribution data.

Precision and recall

Precision and recall are two frequently used evaluation metrics in multimedia information retrieval.

$\text{Precision} = |\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}| / |\{\text{retrieved documents}\}|$

(Fraction of retrieved instances that are relevant to the search)

$\text{Recall} = |\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}| / |\{\text{relevant documents}\}|$

(Fraction of relevant instances that are retrieved)

Vector approximation (VA file) is a frequently used data compression based high-dimensional indexing method.

(1) Explain the reason why VA-file can speed up linear scan.

Since we use approximation to compress vector data, we reduce the amount of data that must be read during a search, thus speeding it up.

(2) Compute the VA-files for the following 4 vectors and map them to a data space with 16 cells.

⊙1	0.2	0.3
⊙2	0.6	0.2
⊙3	0.2	0.9
⊙4	0.5	0.4

11	3			
10				
01		4	4	
00			2	
	00	01	10	11

GRID and EXCELL are two different methods to access point data. List the steps of inserting a new point of the two approaches, respectively.

GRID: 1. Use a point query to find the cell to insert into a. This can be done by calculating its position, getting the scales from memory, and thus the relevant cell.

2. Insert into the cell a. If collision, split the cell either horizontally or vertically, and resort that cell

EXCELL: like GRID, except we don't need to keep the scale in memory and when we have a collision, we split ALL the cells, not just the one in which the collision occurred.

Videos can be represented either by **semantic concepts** or **low-level BOW features**. Please compare the two representations in **content-based video retrieval**. What are the advantages and disadvantages of them?

Here we discuss the issue of the semantic gap. By using semantic concepts to represent videos, the results of content-based video retrieval will more closely capture human perception, thus giving better results. The drawback to semantic concepts is that human perception varies person to person, so results are highly subjective.

Using BOW features fixes this by providing consistent, objective results. Additionally, they are more feasible to compute. The disadvantage is the semantic gap, where the retrieved video may be similar in feature space, but not in semantic space.

List three major challenges of **integrating content-based multimedia search** into a typical web search engine in terms of scalability and efficiency. Briefly explain each challenge you listed.

Indexing: need a new technique to index multimedia objects which is efficient.

Searching: need a new algorithm to optimize the search process to take less time and less processing resources.

Size: Support for larger size database due to the nature of multimedia objects' sizes.

User intent: The key words entered by a user aren't necessarily what they want to search for

Data Scope: Factors such as the diversity of user-base and expected user traffic for a search system also largely influence the design."

The complexity of queries supported by the system could represent a challenge. "The processing becomes more complex when visual queries and/or user interactions are involved."

Visualisation: how to order search results

List three major **research issues in online subsequence detection for video streams**, and briefly explain each issue you listed.

1. **Similarity Measure:** efficient similarity measures are needed which takes less time to process and give accurate results as much as possible.

2. **Data Representation:** a signature(signature=feature) that is compact is required to represent a video.

3. **Indexing:** Must find a way to index video representations for very highly dimensional sequences

4. **Query Processing:** Need efficient query processing (which will obviously require indexes). This may involve pruning records that are not in the result set.

a) What is **metric distance function**?

A metric distance function is a function that maps between two pairs from a set of elements (generally vectors or matrices) to a non-negative real number. Properties of a metric function d acting on two elements x and y are

- $d(x, y) \geq 0$ (non-negativity, or separation axiom)
- $d(x, y) = 0$ if and only if $x = y$ (identity of indiscernible, or coincidence axiom)
- $d(x, y) = d(y, x)$ (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$ (subadditivity / triangle inequality).

b) Is **L1 distance function** a metric distance function?

Yes. The L1 (or Manhattan distance) is a metric function. For $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$, $d(x, y) = |y_1 - x_1| + \dots + |y_n - x_n|$. The norm is positive, or equal if $x=y$. The norm is also symmetrical via the definition of the $|\cdot|$ absolute function. Finally, since $|x_i - z_i| \leq |x_i - y_i| + |y_i - z_i|$, the triangle inequality holds for the whole vector.

c) Is **colour histogram intersection distance** a metric distance function? Prove your answer.

No, because the function returns 1 if two images are identical, which is not the behaviour of a metric function.

What is **relevance feedback**? Discuss its advantages and disadvantages.

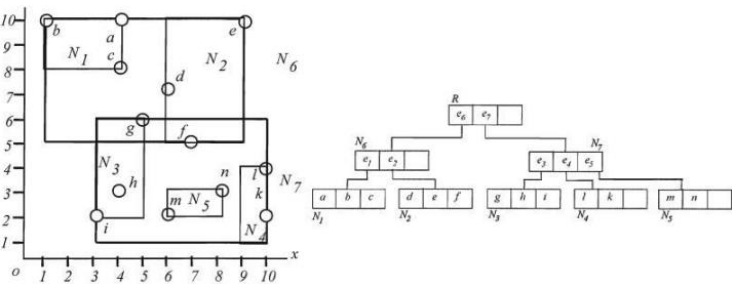
The idea behind relevance feedback is to take the results that are initially returned from a given query and to use information about whether those results are relevant to perform a new query.

It is needed in multimedia searching as we humans tend to give more relevance to certain features.

The advantages of such a feedback system is that better results will be found. A disadvantage is that relevance varies from person to person, so what one person may find relevant another may not.

In the below figures, assume that we have a set of hotels (a, b, ..., n) and for each hotel we store its distance from the beach (x-axis) and its price (y-axis). The hotels are organised in the shown **R-tree** with node capacity=3. AN intermediate entry e_i corresponds to the MBR of a node N_i , while a leaf entry corresponds to a data point (i.e., hotel). Distances are computed according to the L1 norm. The **Branch and Bound Skyline (BBS)** algorithm is used to find the skyline points.

Each row in the following table represents one iteration of the BBS algorithm. For each iteration, the table shows the entry expanded, the heap content, and the list of skyline points. Please complete all rows until the algorithm terminates.

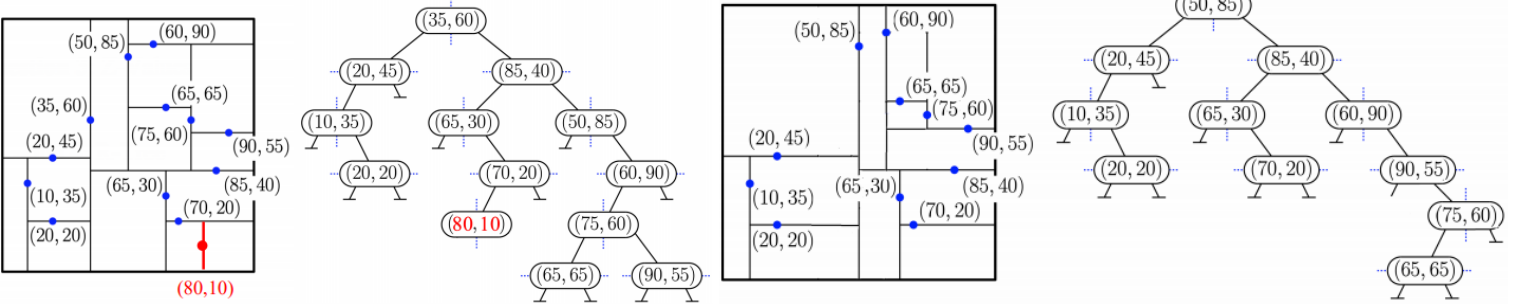


Action	Heap	Skyline
Expand root R	<e7, 4><e6, 6>	Null
Expand e7	<e3, 5><e6, 6><e5, 8><e4, 10>	Null
Expand e3	<i, 5><e6, 6><h, 7><e5, 8><e4, 10><g, 11>	Null
Insert i	<e6, 6><h, 7><e5, 8><e4, 10><g, 11>	i
Expand e6	<h, 7><e5, 8><e1, 9><e4, 10><g, 11>	i
Expand e1	<b, 11><e4, 10><g, 11><c, 12><a, 14>	i
Insert b	<e4, 10><g, 11><c, 12><a, 14>	i, b
Expand e4	<g, 11><c, 12><i, 13><f, 14><e, 14>	i, b

Consider the **Kd-Tree** shown below. Assume that the cutting dimensions alternate between x and y.

(1) Show the result of inserting (80,10) into this tree.

We find the point (80,10) and fall out of the tree on the right child of (70,20). We insert the new node here. Since the parent y-splitter, the new node is an x-splitter.



(2) Show the Kd-Tree that results after deleting the point (35,60) from the original tree. Show both the tree structure and the subdivision of space of the two operations.

The deleted node is at the root. Since it is an x-splitter, its replacement is the node with the smallest x-coordinate in the root's right child, which is (50,85). Since (50,85) is an x-splitter, its replacement is the node with the smallest x-coordinate in its right subtree, which is (60,90).

Since (60,90) is a y-splitter and its right subtree is empty, we find the point with smallest y-coordinate in its left subtree, which is (90,55), and we move this subtree to become the right child of (60,90). Finally, we delete (90,55) from this subtree. Since it is a leaf, it can simply be unlinked from the tree.

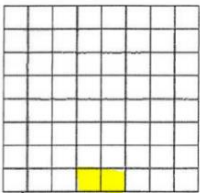
Let the 8x8 grid in the following figures represent the maximum level of space decomposition.

(a) Please find the binary **z-values** to best approximate the two objects in the following figure. You can use up to 4 **z-values** for one object.

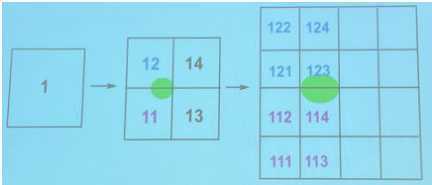
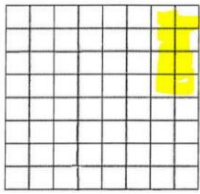
A: 001100 B: 1100**, 011010, 110100

(b) Identify (by shading) the regions represented by the following **z-values**.

The base 10 z-values C: {10, 32}



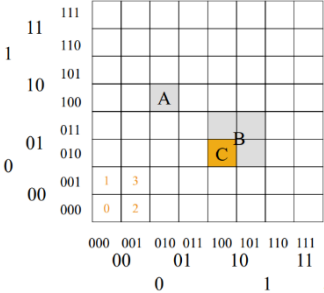
The base 2 z-value D: {111***}



Consider the following table:

Country (name: String, pop: number,

Base-5



boundary: POLYGON)

Where for each country, we record its name, population, and boundary. Also assume that country name is a primary key. Write an **SQL-like query language** for (a) and (b).

In your SQL query, you will need to use some non-standard SQL operations. You need to define these non-standard operations used in your query (what function it performs, what parameters it takes and what it returns).

(a) List the name, population, and area of each country listed in the country table.

```
SELECT name, population, AREA (boundary)
FROM country;
AREA - Returns the area of the polygon passed in as an argument
```

(b) Find all the names of countries that are neighbours of the United Kingdom (UK).

```
SELECT c1.name
FROM country c1
JOIN country c2
  ON TOUCHES (c1.boundary, c2.boundary) and c2.name = "United Kingdom";
TOUCHES - Takes two polygons as arguments and returns true if these have at least one point in common, but their interiors do not intersect
```

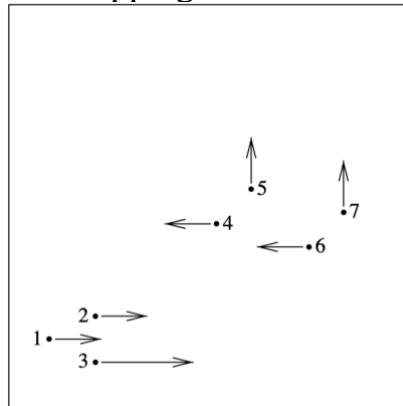
Tutorial 7: Spatiotemporal Data Management

Semester 1, 2020

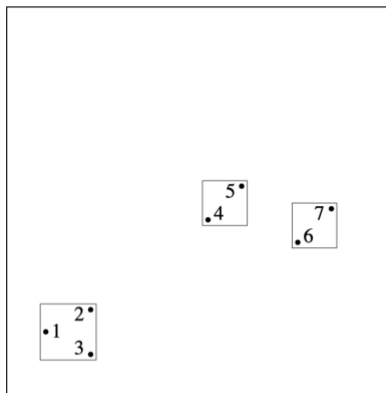
Question 1: What is the benefit of using the TPR-Tree than the other R-Tree-based structures to index the moving objects?

Sample Solution:

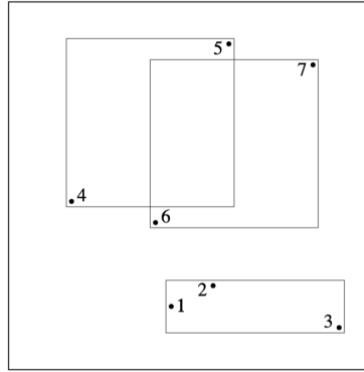
The TPR-Tree is more suitable for queries with a time-interval and has smaller overlapping ratio by considering the current location and the future location. An example of the lower overlapping is shown below:



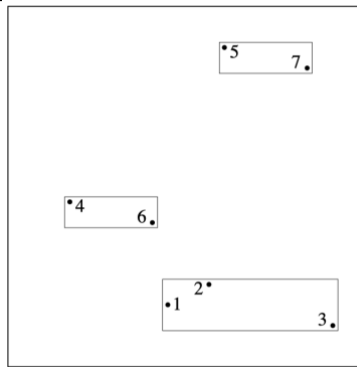
Suppose we have seven points with their current locations at this timestamp as shown in the above figure, and their moving directions are labeled as the arrows. If only considering the locations of the current timestamp, the optimal MBR should be like this:



However, if we use these three MBRs to approximate the points in the next timestamp, they will have a very large overlapping as shown below:



Therefore, if we consider both the locations and the prediction of the future locations (with moving direction and velocity), we could have better choices, like put 5 and 7 together, put 4 and 6 together.



Question 2: LCSS (Longest Common Sub-Sequence) is a way to compare the similarity between two sequences. Given two sequences, LCSS finds the length of the longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous (if we require the contiguous, it is the longest common sub-string). For example, “abc”, “agb”, “bdf”, “aeg”, “acefg” are all sub-sequences of “abcdefg”. Such a meter can also be used in to evaluate how similar two trajectories are. Given two points p and q , a distance threshold ϵ , we regard p and q as the same point if $dist(p, q) \leq \epsilon$. Please discuss the dynamic programming technique that can be used to compute LCSS.

Sample Solution:

We use two sequence to illustrate how LCSS can be computed. Suppose one string is $S=AGGCAB$, another one is $T=GXCXAYB$. We first divide this problem into a set of subproblem: we will look at the LCSS of a prefix of S and a prefix of T , running over all pairs of prefixes. For simplicity, let's

worry first about finding the length of the LCSS and then we can modify the algorithm to produce the full result.

Suppose we use $LCSS[i, j]$ to denote the result of LCSS of sub-sequence $S[1, \dots, i]$ and $T[1, \dots, j]$ (If $i = 2$, and $j = 3$, we are comparing AG and GXC). And suppose we already know all the results of shorter prefix pairs ($LCSS[i-1, j]$, $LCSS[i, j-1]$, $LCSS[i-1, j-1]$). How can we solve the $LCSS[i, j]$ with those smaller and already solved problems' results? Here are two cases:

1. When $S[i] \neq T[j]$, then the result has to ignore one of $S[i]$ or $T[j]$, so we have
 - $LCSS[i, j] = \max (LCSS[i-1, j], LCSS[i, j-1])$
 - Maximum value of the upper and left neighbor cells
2. When $S[i] = T[j]$, then we increase the already matched value of $LCSS[i-1, j-1]$ by 1:
 - $LCSS[i, j] = LCSS[i-1, j-1] + 1$
 - Top left neighbor value plus one

Therefore, we need only two loops to compute these LCSS. Initially, the first row and the first column are set to 0.

	ϕ	G	X	C	X	A	Y	B
ϕ	0	0	0	0	0	0	0	0
A	0							
G	0							
G	0							
C	0							
A	0							
B	0							

Then we fill this table row by row with the previous two rules:

	ϕ	G	X	C	X	A	Y	B
ϕ	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
C	0	1	1	2	2	2	2	2
A	0	1	1	2	2	3	3	3
B	0	1	1	2	2	2	2	4

Therefore, the LCSS is $|GCAB|=4$.

Question 3: Edit Distance is another widely used string similarity measurement that can be used to compare two trajectories. To convert from one string to another, for each character, we have four choices: let it be, insert another one, delete it, and replace it. The insertion, deletion, and substitution are called edit operations, and the minimum number of edit operations it takes to transform one string to another is called the edit distance between them. Similarly, in the trajectory distance, we can regard two points within a distance threshold as the same one. In this way, the trajectory similarity is converted to the string similarity. Please use the example in Question 1 to show what is the edit distance between S and T .

Sample Solution:

All the notations are the same as Question 1. For each cell $[i, j]$ in the table, it means the minimum number of edit operations it takes to convert string $S[1, i]$ to $T[1, j]$. The edit operations have the following actual effects on the value computation. Suppose we are now at $[i, j]$, we have three choices to compute its edit distance:

1. We can use the edit distance between $S[1, i - 1]$ and $T[1, j]$ (the left neighbor), so we only need insert the value of $T[j]$ at $S[i]$ to make $S[1, i]$ and $T[1, j]$ the same. Therefore, the edit distance at $ED[i, j] = ED[i, j - 1] + 1$.
 1. For example, in the first line, from the empty string ϕ to G , we have to insert G , so the edit distance is 1; From G to GX , we have insert X , so the edit distance is $1+1$... And eventually, from ϕ to $GXCXAYB$, we have to insert 7 characters in total.
 2. Whenever we use the edit distance from the left neighbor, it means we do the insertion, so the edit distance plus 1.
2. We can also use the edit distance between $S[1, i]$ and $T[1, j - 1]$ (the upper neighbor), so we only need delete the value of $S[i]$ to make $S[1, i]$ and $T[1, j]$ the same. Therefore, the edit distance at $ED[i, j] = ED[i, j - 1] + 1$.

$ED[i-1, j]$

1. For example, in the first column, it means the edit distance from A to ϕ , so we delete A , and its edit distance is 1; Then if we convert AG to ϕ , the edit distance is 2... Finally, the edit distance from $AGGCAB$ to ϕ is 6, because we have to delete all six characters.

2. Whenever we use the edit distance from the upper neighbor, it means we do the deletion, so the distance plus 1.
3. We can also use the edit distance between $S[1, i - 1]$ and $T[1, j - 1]$ (the upper-left neighbor). This time we have two cases to consider
 1. If $S[i] = T[j]$, we can do not need any editing, so $ED[i, j] = ED[i - 1, j - 1]$
 2. If $S[i] \neq T[j]$, we can replace $S[i]$ with $T[j]$, so $ED[i, j] = ED[i - 1, j - 1] + 1$
4. Now for each $ED[i, j]$, we have the following three choices. We use the minimum of these three as the edit distance:
 1. From the left neighbor, $ED[i, j] = ED[i, j - 1] + 1$
 2. From the upper neighbor, $ED[i, j] = ED[i - 1, j] + 1$.
 3. From the upper-left neighbor:
 - i. $ED[i, j] = ED[i - 1, j - 1]$, if $S[i] = T[j]$
 - ii. $ED[i, j] = ED[i - 1, j - 1] + 1$, if $S[i] \neq T[j]$

In summary, $ED[i, j] = \min(ED[i, j - 1] + 1, ED[i - 1, j] + 1, ED[i - 1, j - 1] + 0/1)$. So only three computations and taking the minimum is enough.

	ϕ	G	X	C	X	A	Y	B
ϕ	0	1	2	3	4	5	6	7
A	1	1	2	3	4	4	5	6
G	2	1	2	3	4	5	5	6
G	3	2	2	3	4	5	6	6
C	4	3	3	2	3	4	5	6
A	5	4	4	3	3	3	4	5
B	6	5	5	4	4	4	4	4

The edit distance between AGGCAB and GXCXAYB is 4. The actual edit steps are: Delete the first A, replace the second G with X, insert X between C and A, insert Y between A and B.

Question 4: How to compute the DTW with the dynamic programming?

Sample Solution:

Unlike LCSS and ED, the DTW is not about how many same characters or how many operations, but the actual distance between two trajectories. Still, using the points are not handy to illustrate the example, we use two one dimensional numbers here: $S = \{1, 2, 3, 5, 5, 5, 6\}$, $T = \{1, 1, 2, 2, 3, 5\}$. Therefore,

the distance between two numbers are just their difference. When working in the 2D point, we can simply replace the distance with Euclidean distance or Network distance.

Like the dynamic programming in LCSS and ED, for each cell $[i, j]$, we also have three sources: the left neighbor $DTW[i - 1, j]$, the upper neighbor $DTW[i, j - 1]$ and the upper-left neighbor $DTW[i - 1, j - 1]$. All we need to do is take the minimum of them and add the distance between $S[i]$ and $T[j]$.

	0	1	1	2	2	3	5
0	0	INF	INF	INF	INF	INF	INF
1	INF	0	0	1	2	4	8
2	INF	1	1	0	0	1	4
3	INF	3	3	1	1	0	2
5	INF	7	7	4	4	2	0
5	INF	11	11	7	7	4	0
5	INF	15	15	10	10	6	0
6	INF	20	20	14	14	9	1

Tutorial 8: High Dimensional Indexing and Search

Semester 1, 2020

Question 1: What is the dimensionality curse? Why we still can do the similarity search in high dimensional space?

Sample Solution:

The dimensionality curse are the following observations:

1. The number of partitions 2^d grows exponentially as the dimensional number grows. When d becomes large enough, we have more partitions than data points.
2. When data is uniformly distributed, most of the data are in the boundary regions, leaving the central space a very hollow space. Suppose we take the range of $[0.05, 0.95]$ of each dimension, then the interior region's volume is 0.9^d . When $d = 50$, that takes only 0.005 of the entire volume.
3. It would be hard to distinguish the distance between the objects. The distance between the nearest neighbor and the farthest neighbor becomes nearly the same.

Because the real data distribution is nearly never uniformly distributed, there are always some clusters and skewed distribution. Therefore, we still can find the nearest neighbor within each cluster.

Question 2: X-Tree contains three kinds of nodes: *Data Node*, *Normal Directory Node*, and *Supernode*. Describe what causes the following two special cases of the X-Tree and analyze their performance: (1) None of the directory nodes is a Supernode; (2) The directory consists of only one large Supernode (root).

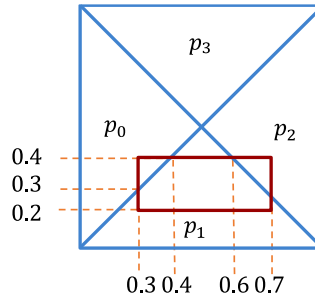
Sample Solution:

- (1) The X-Tree has a completely hierarchical organization of the directory and is therefore similar to an R-Tree. It could be caused by the low

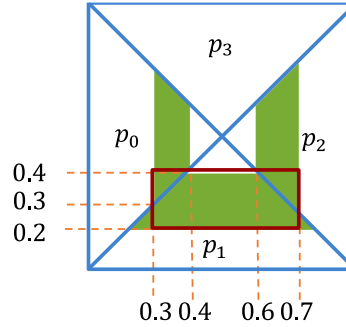
dimensional and non-overlapping data. The performance corresponds to the tree height. It has the extreme height of the X-Tree.

- (2) The directory of the X-Tree is basically one root: one Supernode that contains the lowest directory level of the corresponding R-Tree. It could be caused by high-dimensional and highly overlapping data. The performance corresponds to the performance of a linear directory scan. It has the extreme size of a Supernode.

Question 3: Given a 2D pyramid as shown below, what are the query ranges to search in the B-Tree?



Sample Solution:



The query region covers three pyramids: p_0 , p_1 and p_2 . The actual query regions are labeled in the green. The height ranges in these three pyramids are computed as follow:

1. For the height in p_0 , its range is $[0.5-0.4, 0.5-0.3] = [0.1, 0.2]$.
2. For the height in p_1 , its range is $[0.5-0.4, 0.5-0.2] = [0.1, 0.3]$.
3. For the height in p_2 , its range is $[0.6-0.5, 0.7-0.5] = [0.1, 0.2]$.

Then we add each of these values with their corresponding pyramid ID to generate the final search range in the B-Tree:

1. For Pyramid p_0 : $[0.1, 0.2]$
2. For Pyramid p_1 : $[1.1, 1.3]$
3. For Pyramid p_2 : $[2.1, 2.2]$

Question 4: VA-File bears a close resemblance to the grid file as we discussed in the spatial index module. They both impose a grid on the underlying dataspace, and queries are processed by inspecting data that falls into relevant grid cells. What are the differences between them?

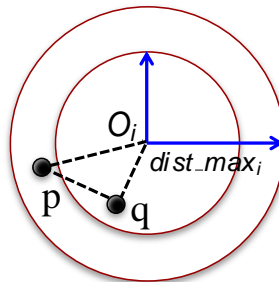
Sample Solution:

1. In VA-File, the relevant grid cells are obtained by a sequential scan of all the grid cells, while in grid file, the relevant grid cells are obtained by random access via the aid of the d linear scales.
2. In the grid file, the boundaries of the slices are modified as the underlying data changes. Thus, the dynamic behaviour of the VA-File may be bad, which means that, at times, the VA-File may have to be rebuilt.
3. Moving points in a grid file is random access, while the same action in VA-File needs a scan of the entire VA-File. Therefore, the grid file has a good dynamic behavior.

Question 5: How is the triangle inequality is used to determine the search space in the iDistance?

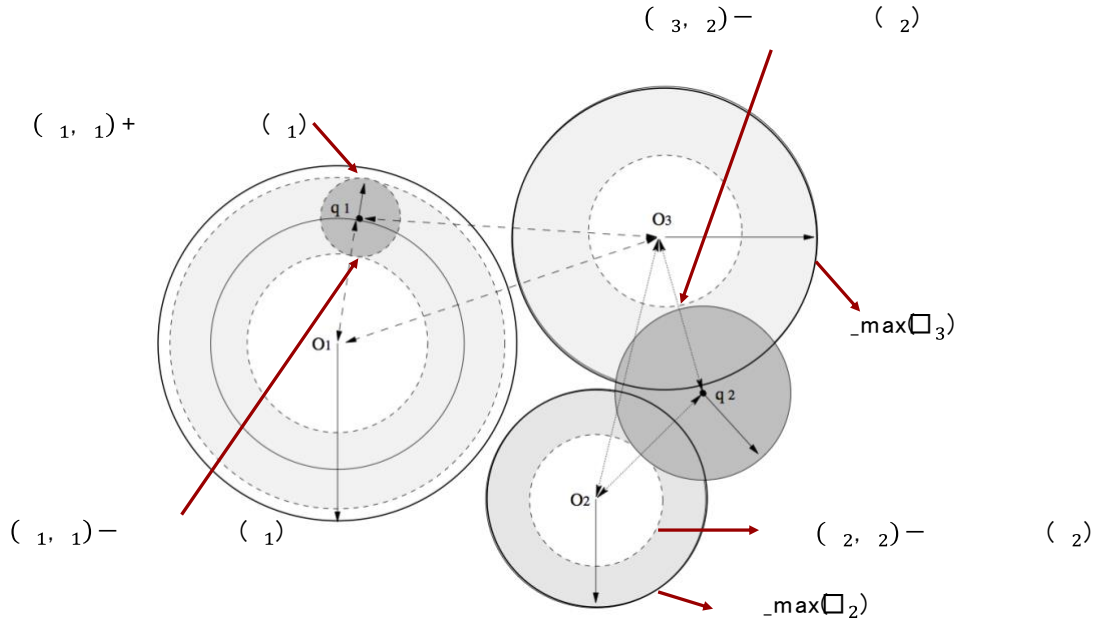
Sample Solution:

As we can see from the following example, the cluster reference point O , the query point q , and the result p form a triangle. Because the data are organized with their distance to O , so we have to know range of $dist(O_i, p)$ to determine the search space. The lower bound of the search space (the inner radius) is $dist(O_i, q) - querydist(q)$, and the upper bound of the search space (the outer radius) is the smaller one of $dist_max(O_i)$ and $dist(O_i, q) + querydist(q)$.



As we can see in this example, for query point q_1 with query range $querydist(q_1)$, the lower bound the query result is $dist(O_1, q_1) - querydist(q_1)$, and the upper bound the query result is $dist(O_1, q_1) + querydist(q_1)$. The result search space is a ring around O_1 .

q_1 does not need to test visit cluster 2 and cluster 3 because its lower bound to O_2 (similar to O_3) is $dist(O_2, q_1) - querydist(q_1) > dist_max(q_2)$. Therefore, O_2 and O_3 are pruned.



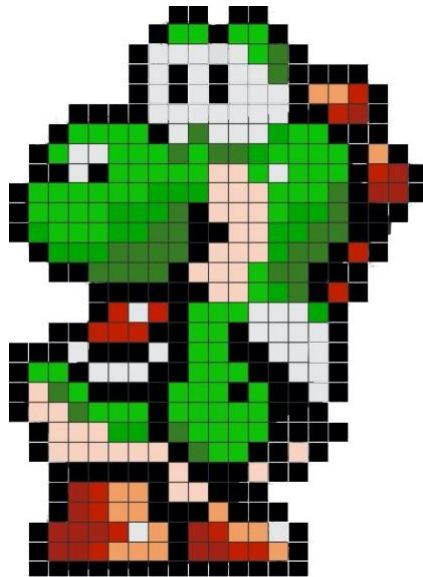
For query point q_2 , its lower bounds to O_2 and O_3 are $dist(O_2, q_2) - querydist(q_2)$ and $dist(O_3, q_2) - querydist(q_2)$. Their upper bounds are all their maximum distance because their maximum distances are all smaller than $dist(O_i, q_1) + querydist(q_1)$.

q_1 does not need to visit cluster 1 because its distance lower bound to O_1 is $dist(O_1, q_2) - querydist(q_2) > dist_max(O_1)$.

Tutorial 9: An Introduction to Multimedia Databases











Semester 1, 2020

Question 1: Suppose we are using 24-bit RGB model for image representation, and extract color features using the 4-bit 3D color histogram: 2 bits for red, and 1 bit for green and blue. Given a picture of the Yoshi (21 × 29 bit), what is the color distribution of it?



Sample Solution:

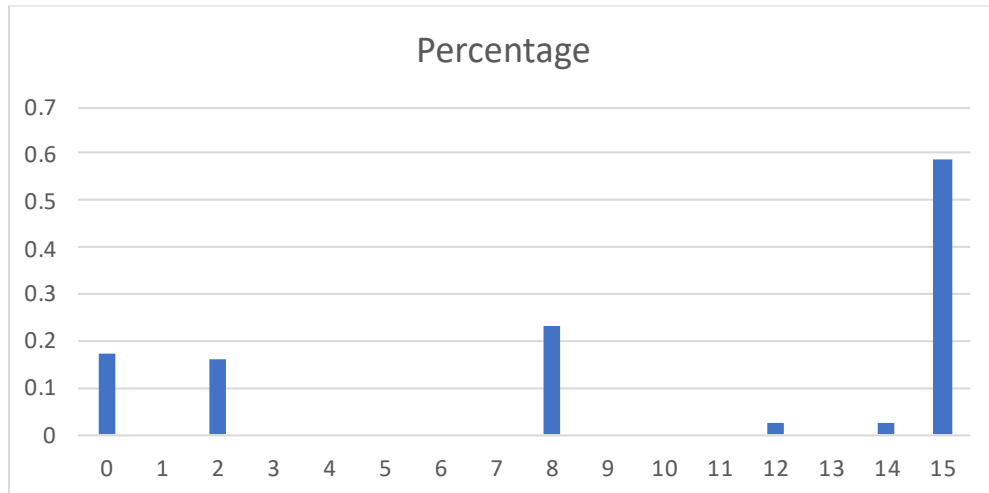
Firstly, this figure has following pixels (suppose I counted correctly...)

1.  (22, 191, 9), 78 pixels, (0,1,0)
2.  (0, 169, 0), 22 pixels (0,1,0)
3.  (57, 125, 39), 30 pixels (0,0,0)
4.  (246, 211, 193), 35 pixels (3,1,1)
5.  (239, 160, 101), 15 pixels (3,1,0)
6.  (193, 32, 1), 17 pixels (3,0,0)
7.  (163, 35, 23), 14 pixels (2,0,0)
8.  (0, 0, 0), 75 pixels (0,0,0)
9.  (229, 230, 231), 46 pixels (3,1,1)
10.  (255,255,255), 277 pixels (3,1,1)

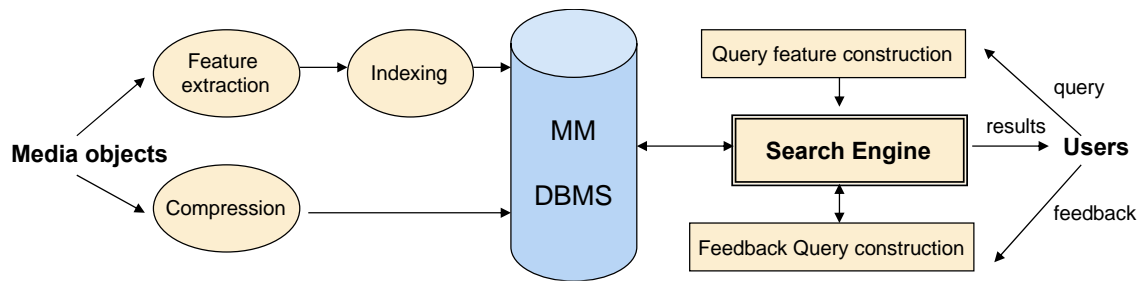
Next we fill the 3D histogram table to do the counting:

Bin	R,G,B	# Pixels	Percentage
0	0,0,0	105	0.172
1	0,0,1	0	0
2	0,1,0	100	0.164
3	0,1,1	0	0
4	1,0,0	0	0
5	1,0,1	0	0
6	1,1,0	0	0
7	1,1,1	0	0
8	2,0,0	14	0.23
9	2,0,1	0	0
10	2,1,0	0	0
11	2,1,1	0	0
12	3,0,0	17	0.028
13	3,0,1	0	0
14	3,1,0	15	0.025
15	3,1,1	358	0.588

Finally, we can get the histogram below:



Question 2: Discuss the architecture of the multimedia database system.



Example Solution:

On the left-hand side is the storage / index component, and on the right-hand side is the query component.

For each multimedia object to be stored in the MMDBMS, we store its compressed version in disk because most of the MM data are very big. This compressed data is only used to return as the final result, and it will not be used during the search. Therefore, in order to support query/search, we need to extract features from the MM object and organize these features. This process is called feature extraction. For most of the cases, the extracted features have a large dimensional number, so we treat them as high dimensional data and organize them with the high dimensional indexes. Once again, the index is built on the features, not on the original data. Each original MM object is represented by a point in the feature space. How the features are abstracted is also part of the MMDBMS design, so all the data stored in it share the same feature space.

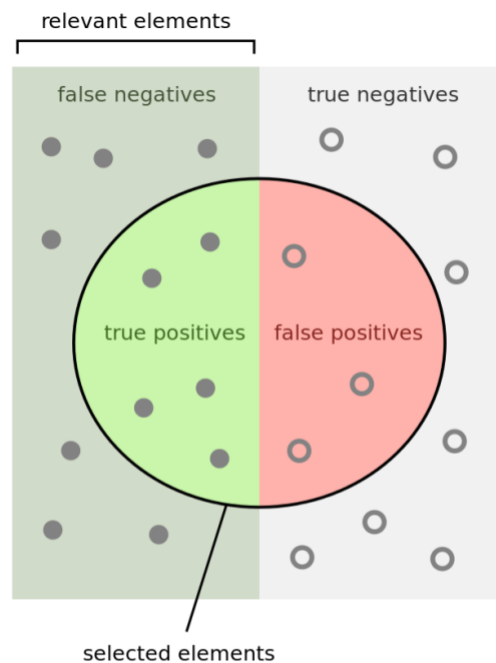
During the search, user provides a query object. MMDBMS does not use this query object to search directly. Instead, it first abstracts the features of the query object with exactly the same process when it abstracts the data it stores. Then it uses the query features to search in the feature space with the help of high dimensional index. With the algorithm of kNN, it finds a list of points that are similar to the user's query object, and retrieves the actual data stored in the database, decompresses them, and returns them to the user.

There is a feedback loop for user to evaluate the query result and let the database refine the result. Because all the results are feature similarity-based, there is no guarantee the results can satisfy the user's need. Therefore, this feedback loop can help identify which feature the user cares more and assigns higher weights to them during the search. For example, if the user cares more about

the color, then in the next iteration, the search gives higher weight on the color feature and lower weights on the other features like shape and texture.

Question 3: Suppose we have 100 data in the database, 60 of them are positive, 40 of them are negative, and we want to retrieve all the positive ones. However, the system returns 50 results, and 40 of them are positive. What are the precision and recall of this result?

Sample Solution:



As shown in the above plot, the overall rectangle contains the entire data set, with the dots (left half) representing the positive data, and circles (right half) representing the negative data. The big circle is the result of query finding all the positive data. The dots in the green half are the correct ones, so they are called *true positive* (*true* because of correct, *positive* because the query asks for positive). In our example, the true positive number is 40.

The circles in the red half appear in the positive results, but they are actually negative, so they are not correct. Therefore, we call them *false positive* (*false* because they are incorrect, *positive* because the query asks for positive). In our example, the false positive number is 10.

For all the data outside the result circle, the query result regards them as negative. However, on the left-hand side dots, they are positive, so this

negative is not correct, and we call them *false negative*. In our example, it is 20.

Similarly, the right circles are actually negative, so we call them *true negative*. In our example, it is 30.

The precision asks for the ratio of the correctly returned ones in the returned results, so it is $Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{40}{50} = 0.8$.

The recall asks for the ratio of the correctly returned ones in all the correct ones, so it is $Recall = \frac{True\ Positive}{True\ Positive + False\ negative} = \frac{40}{60} = 0.667$.

Question 4: We always want to achieve high precision and high recall in a system, but it is not always achievable. Please discuss the relation between the precision and recall.

Sample Solution:

Often, there is an inverse relationship between precision and recall, where it is possible to increase one at the cost of reducing the other. Brain surgery provides an illustrative example of the trade-off. Consider a brain surgeon tasked with removing a cancerous tumour from a patient's brain. The surgeon needs to remove all of the tumour cells since any remaining cancer cells will regenerate the tumour. Conversely, the surgeon must not remove healthy brain cells since that would leave the patient with impaired brain function. The surgeon may be more liberal in the area of the brain she removes to ensure she has extracted all the cancer cells. This decision increases recall but reduces precision. On the other hand, the surgeon may be more conservative in the brain she removes to ensure she extracts only cancer cells. This decision increases precision but reduces recall. That is to say, greater recall increases the chances of removing healthy cells (negative outcome) and increases the chances of removing all cancer cells (positive outcome). Greater precision decreases the chances of removing healthy cells (positive outcome) but also decreases the chances of removing all cancer cells (negative outcome). Therefore, another performance measure, *F-measure*, is proposed to combine precision and recall by their harmonic mean, as shown below:

$$F = 2 \times \frac{precision \times recall}{precision + recall}$$

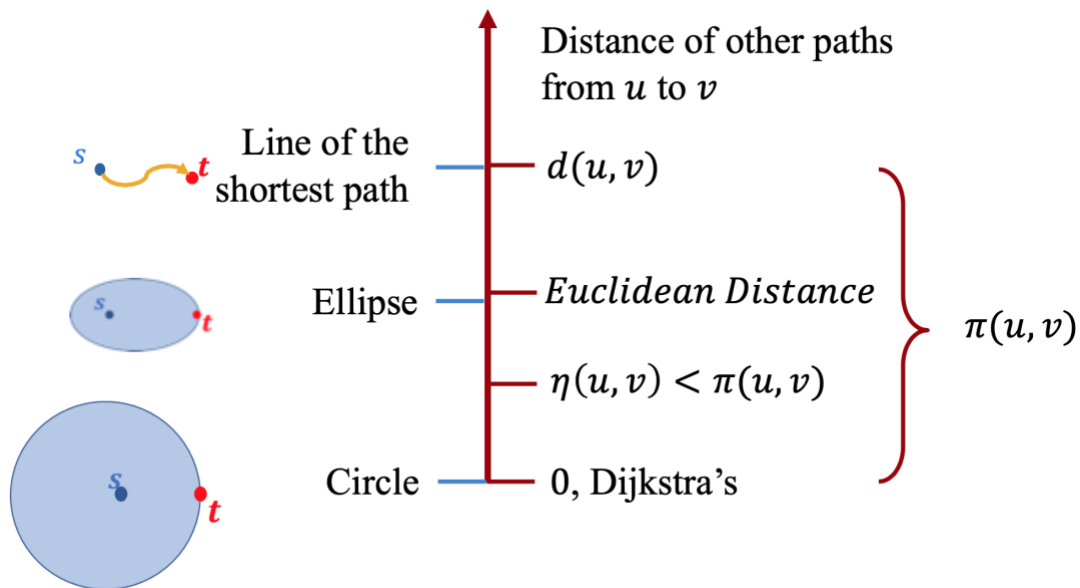
Tutorial 10: Route Planning in Road Network

Semester 1, 2020

Question 1: The performance of the distance-estimation based heuristic search algorithms (A^* and Landmark) are highly affected by the quality of the estimation. Please discuss how the estimated distance influences the algorithm performance.

Sample Solution:

In the distance-estimation based heuristic search, the key in the priority queue is $d(s, u) + \pi(u, t)$ instead of $d(s, u)$. As long as $\pi(u, t) \leq d(u, t)$, the correctness can be guaranteed.



As for the influence of $\pi(u, t)$, when it is 0, the search space reduces to the same as the Dijkstra's (a circle). When $\pi(u, t) = d(u, t)$, the search space is just the shortest path itself (a line). When $0 \leq \pi(u, t) \leq d(u, t)$, the search space is in between, like an ellipse. When $\pi(u, t)$ is closer to 0, the ellipse is larger; when (u, t) is closer to $d(u, t)$, the ellipse is smaller.

Question 2: The Dijkstra's algorithm is essentially a single-source shortest path algorithm that can be used to answer the point-to-point path query. On the contrary, the A^* algorithm is essentially a point-to-point shortest path algorithm because its searching heuristic is towards a destination. Can A^* also be used as a single-source algorithm? If so, how does it work?

Sample Solution:

Yes, A^* can be used to find the distances from the source to all the other vertices in the graph.

The procedure is the same as the Dijkstra's: When a point pops out of the priority queue with $d(s, u) + \pi(u, t)$, this $d(s, u)$ is the shortest distance from s to u . We can run this procedure on and on, even passes t . The algorithm terminates when the priority queue becomes empty. In this way, we can find the distance from s to all the other vertices.

The proof is not required by this course.

Question 3: During the construction of CH, the shortcuts we add have the actual shortest distances between the vertices. However, this requires a large amount of Dijkstra's search to guarantee this property. If for any neighbour pair (v, w) of u , where v and w are contracted later than u , we add a shortcut (v, w) with distance $\min(d(v, u) + d(u, w), d(v, w))$ ($d(v, w) = \infty$ if there is no edge or shortcut between v and w), then the contraction process will become much faster. Can this method also answer the query correctly? What are the advantages and the disadvantages of this method compared with the classic CH?

Sample Solution:

Yes, the correctness is guaranteed. Because we still need to run the Bi-directional upwardly search, the Dijkstra's search is in a way postponed to the query answering stage. When we contract a vertex v , the standard CH adds the shortcut only when $u \rightarrow v \rightarrow w$ is the shortest path from u to w , while the new method is much looser and creates more shortcuts even when $u \rightarrow v \rightarrow w$ is not the shortest path. This redundant information guarantees the correctness of the latter contractions.

This approach has the advantage of fast construction because no Dijkstra is needed. But it takes longer time to answer a query than the standard CH because it has a much larger index size. In fact, its performance is near to A^* .

Question 4: Pruned Landmark Labeling is an easy way to construct the 2 Hop labels. However, it is more suitable for the *small world* graph than the road network. The small world graph is a kind of graph that is composed of several densely connected communities, and the connections between the communities are very limited. Please discuss how the pruned landmark labeling can benefit from this property.

Sample Solution:

The small world graph has a property that small number of nodes have a very large number of degrees such that most of the remaining nodes connect to them. Therefore, these nodes are the natural hub of the graph. The shortest paths among other vertices have a very high chance to pass through them. Then if we run the pruned landmark labeling in the degree decreasing order, these important nodes would become labels in other nodes earlier, and they can help prune the search space a lot.

