On the Generation of Spatiotemporal Datasets

Yannis Theodoridis, Jefferson R. O. Silva and Mario A. Nascimento

January 22, 1999

TR-40

A TIMECENTER Technical Report

Title	On the Generation of Spatiotemporal Datasets
	Copyright © 1999 Yannis Theodoridis, Jefferson R. O. Silva and Mario A. Nascimento . All rights reserved.
Author(s)	Yannis Theodoridis, Jefferson R. O. Silva and Mario A. Nascimento
Publication History	January 1999. A TIMECENTER Technical Report.
	January 1999. A ChroroChronos Technical Report.

TIMECENTER Participants

Aalborg University, Denmark
Christian S. Jensen (codirector), Michael H. Böhlen, Renato Busatto, Curtis E. Dyreson,
Heidi Gregersen, Dieter Pfoser, Simonas Šaltenis, Janne Skyt, Giedrius Slivinskas,
Kristian Torp

University of Arizona, USA
Richard T. Snodgrass (codirector), Bongki Moon, Sudha Ram

Individual participants
Anindya Datta, Georgia Institute of Technology, USA
Kwang W. Nam, Chungbuk National University, Korea
Mario A. Nascimento, State University of Campinas and EMBRAPA, Brazil
Keun H. Ryu, Chungbuk National University, Korea
Michael D. Soo, University of South Florida, USA
Andreas Steiner, TimeConsult, Switzerland
Vassilis Tsotras, University of California, Riverside, USA
Jef Wijsen, Vrije Universiteit Brussel, Belgium

For additional information, see The TIMECENTER Homepage:
    URL: <http://www.cs.auc.dk/TimeCenter>

The TIMECENTER icon on the cover combines two "arrows." These "arrows" are letters in the so-called *Rune* alphabet used one millennium ago by the Vikings, as well as by their precedessors and successors. The Rune alphabet (second phase) has 16 letters, all of which have angular shapes and lack horizontal lines because the primary storage medium was wood. Runes may also be found on jewelry, tools, and weapons and were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote "T" and "C," respectively.

Abstract

An efficient benchmarking environment for spatiotemporal access methods should at least include modules for: generating synthetic datasets, storing datasets (real datasets included), collecting and running access structures, and visualizing experimental results. Focusing on the dataset repository module, a collection of synthetic data that would simulate a variety of real life scenarios is required. Several algorithms have been implemented in the past to generate static spatial (point or rectangular) data, for instance, following a predefined distribution in the workspace. However, by introducing motion, and thus temporal evolution in spatial object definition, generating synthetic data tends to be a complex problem. In this paper, we discuss the parameters to be considered by a generator for such type of data, propose an algorithm, called "Generate_Spatio_Temporal_Data" (GSTD), which generates sets of moving point or rectangular data that follow an extended set of distributions. Some actual generated datasets are also presented. The GSTD source code and several illustrative examples are currently available in the Internet[1].

Keywords: spatiotemporal databases, benchmarking, data generators, indexing, access structures, query performance

1. Introduction

A field of ongoing research in the area of spatial databases and Geographical Information Systems (GIS) involves the accurate modeling of real geographical applications, i.e., applications that involve objects whose position, shape and size change over time. Real world examples include storage and manipulation of trajectories, fire or hurricane front monitor, simulators (e.g. flight simulators), weather forecast, etc.

Database Management Systems (DBMS) should be extended towards the efficient modeling and support of such applications. Towards this goal, recent research efforts have aimed at:

- modeling and querying time-evolving spatial objects (e.g. [SWCD97, EGSV98, T98]),
- designing index structures and access methods (e.g. [NS98, TVM98]),
- implementing appropriate architectures and systems (e.g. [WXCJ98]).

In the recent literature, one can find work on formalization and modeling of spatiotemporal databases and a wide set of definitions about spatiotemporal objects. In the rest of the paper, we adopt the *discrete* definition for spatiotemporal objects that appears in [TSPM98]:

Definition: A *spatiotemporal object*, identified by its *o_id,* is a time-evolving spatial object, i.e., its evolution (or 'history') is represented by a set of instances (o_id, s_i, t_i), where s_i is the location of object o at instant t_i (s_i and t_i are called *spacestamp* and *timestamp*, respectively).

According to the above definition, a two-dimensional time-evolving point (region) is represented by a line (solid) in three-dimensional space. Figure 1 illustrates two examples: (a) a *moving point* and (b) a

---

[1] URLs: `http://www.dblab.ece.ntua.gr/~theodor/GSTD` and `http://www.dcc.unicamp.br/~mario/GSTD`.

moving region, according to the terminology proposed in [EGSV98]. Although in the rest of the paper, we consider objects of dimensionality $d = 2$, extension to higher dimensions is straightforward[2].
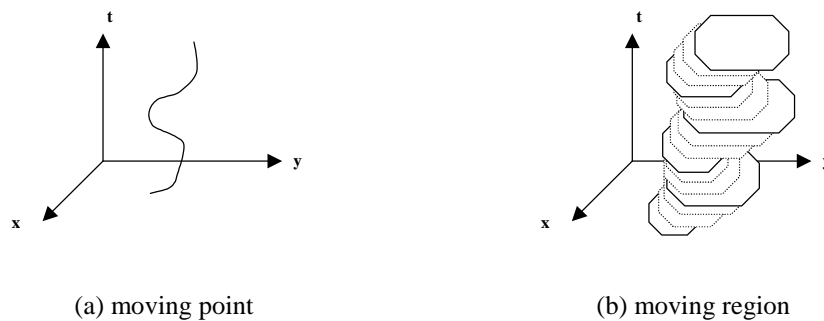

(a) moving point   (b) moving region

Figure 1: Two-dimensional time-evolving spatial objects

One of the tasks that a SpatioTemporal Database Management System (STDBMS) should definitely support includes the efficient indexing and retrieval of spatiotemporal objects. This task demands robust indexing techniques and fast access methods for a wide set of possible queries on spatiotemporal data. Either extensions of existing spatial access methods [XHL90, TVS96, NS98, TVM98] or new 'from-the-scratch' methods could be reasonable candidates. All proposals, however, should be evaluated under extensive experimentation on real and synthetic data. For instance, query processing and/or index building time (either real wall-clock time, or number of disk I/Os), space requirements and combinations thereof are all possible parameters against which one may want to evaluate a given index proposal.

Overall, there is a lack of consistent performance comparison among the proposed approaches, with respect to the space occupied, the construction time, and the response time in order to answer a variety of spatial, temporal, and spatiotemporal queries. Moreover, [ZMR96] suggests that "*experiments of indexing techniques should be based on benchmarks such as standard sets of data and queries*".

Following that, the general architecture of a benchmarking environment for spatiotemporal access methods (STAMs) that is currently under design includes the following:

(a) *a module that generates synthetic data and query sets*, which would cover a variety of real life examples,

(b) *a repository of real datasets* (such as TIGER files for - static - spatial data),

(c) *a collection of access structures* for experimentation purposes,

(d) *a database of experimental results*, and

(e) *a visualization tool* that could be able to visualize datasets and structures, for illustrative purposes.

Our study continues an attempt towards a *specification and classification scheme* for STAMs initiated in [TSPM98]. Within the above framework, in this paper we concentrate on module (a) and, in particular:

• discuss parameters that have to be taken into consideration for generating spatiotemporal datasets, and

• propose an algorithm that generates datasets simulating a variety of scenarios with respect to user requirements.

---

[2] Popular examples of spatial datasets with dimensionality $d > 2$ include, among others, virtual reality worlds ($d = 3$) and feature-based image databases (usually $d \leq 256$).

The rest of the paper is organized as follows: In Section 2 we discuss the motivation for this study. Section 3 discusses the parameters that need to be taken into consideration. An appropriate algorithm is presented in Section 4 together with example results and applications. Section 5 discusses several issues that arise and surveys related work. Finally, Section 6 concludes by also giving directions for future work.

2. Motivation

In the literature, several access methods have been proposed for spatial data without, however, taking the time aspect into consideration. Those methods are capable of manipulating geometric objects, such as points, rectangles, or even arbitrary shaped objects (e.g. polygons). An exhaustive survey is found in [GG98]. On the other hand, temporal access methods have been proposed to index valid and/or transaction time, where space is not considered at all. A large family of access methods has been proposed to support multiversion / temporal data, by keeping track of data evolution over time (e.g. assume a database consisting of medical records, or employees' salaries, or bank transactions, etc.). For a survey on temporal access methods see [ST98].

To the best of our knowledge, there is a very limited number of proposals that consider both spatial and temporal attributes of objects. In particular, *MR-trees* and *RT-trees* [XHL90], *3D R-trees* [TVS96], and *HR-trees* [NS98] are based on the R-tree family [Gut84, BKSS90, KF94] while *Overlapping Linear Quadtrees* [TVM98] are based on the Quadtree structure [Sam84]. These approaches have the following characteristics:

- 3D R-trees treat time as another dimension using a 'state-of-the-art' spatial indexing method, namely the R-tree,
- MR-trees and HR-trees (Overlapping Linear Quadtrees) embed the concept of overlapping trees [MK90] into R-trees (Quadtrees) in order to represent successive states of the database, and
- RT-trees couple time intervals with spatial ranges in each node of the tree structure by adopting ideas from TSB trees [LS89].

The majority of proposed spatiotemporal access structures are based on the R-tree (one exception is [TVM98]), as such we focus on such structures and a short survey of the R-tree based approaches follows.

Assuming *time* to be another dimension is a simple idea, since several tools for handling multidimensional data are already available [GG98]. The 3D R-tree implemented in [TVS96] considers time as an extra dimension in the original two-dimensional space and transforms two-dimensional rectangles in three-dimensional boxes. Since the particular application considered in [TVS96] (i.e., multimedia objects in an authoring environment) involves Minimum Bounding Rectangles (MBRs) that do not change their location through time, no dead space is introduced by their three-dimensional representation. However, if the above approach were used for moving objects, a lot of empty space would be introduced (Figure 2).
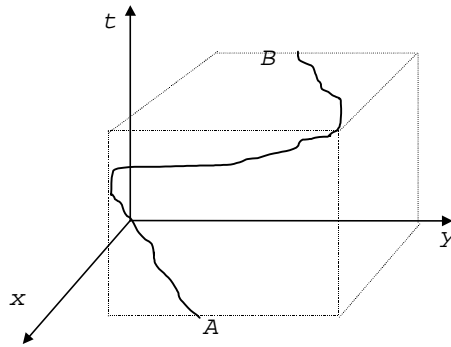
Figure 2: The MBR of a moving object occupies a large portion of the data space

The approach followed by the RT-tree [XHL90] only partially solves that problem. Time information is incorporated, by means of time intervals, inside the (two-dimensional) R-tree structure. Each entry, either in a leaf or a non-leaf RT-tree node, contains entries of the form (S, T, P), where S is the spatial information (MBR), T is the temporal information (interval), and P is a pointer to a subtree or the detailed description of the object. Let $T = (t_i, t_j)$, $i \leq j$, t_j be the current timestamp and t_{j+1} be the consecutive one. If an object does not change its spatial location from t_j to t_{j+1}, then its spatial information S remains the same, whereas the temporal information T is updated to T', by increasing the interval upper bound, i.e., $T' = (t_i, t_{j+1})$. However, as soon as an object changes its spatial location, a new entry with temporal information $T = (t_{j+1}, t_{j+1})$ is created and inserted into the RT-tree. This insertion strategy makes the structure mostly efficient for databases of low mobility; evidently, if we assume that the number of objects that change is large, then many entries are created and the RT-tree grows considerably. An additional criticism is based on the fact that R-tree node construction depends on spatial information S while T plays a complementary role. Hence the RT-tree is not able to support temporal queries (e.g. "*find all objects that exist in the database within a given time interval*").

On the other hand, MR-trees and HR-trees are influenced by the work on overlapping B-trees [MK90]. Both methods support the following approach: different index instances are created for different transaction timestamps. However, in order to save disk space, common paths are maintained only once, since they are shared among the structures. The collection of structures can be viewed as an *acyclic graph*, rather than a collection of independent tree structures. The concept of overlapping tree structures is simple to understand and implement. Moreover, when the objects that have changed their location in space are relatively few, then this approach is very space efficient. However, if the number of moving objects from one time instant to another is large, this approach degenerates to independent tree structures, since no common paths are likely to be found. Figure 3 illustrates an example of overlapping trees for two different time instants t_0 and t_1. The dotted lines represent links to common paths / subpaths.
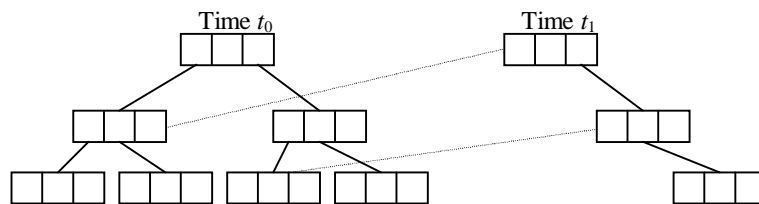

Figure 3: Overlapping trees for two different time instants t_0 and t_1.

Among the aforementioned proposals, the 3D R-tree has been implemented and experimentally

tested [TVS96] using synthetic (uniform) datasets. The retrieval cost for several pure temporal, pure spatial and spatiotemporal operators was measured and appropriate guidelines were extracted. Recently, [NST98] compares the HR-tree with the 3D R-tree and another structure, called 2+3 R-tree, using two R-trees and a rationale similar to the 2R approach presented in [KTF98]. The basic conclusion is that the HR-tree is far more efficient in terms of query processing for *time point* queries while that is not true for *time interval* queries. Also, the HR-tree may result in a rather large structure.

3. A set of operations and parameters

[TSPM98] discusses a list of specifications to be considered when designing and evaluating efficient STAMs with respect to: (i) data types and datasets supported, (ii) issues on index construction, and (iii) issues on query processing. While the second and third ones mainly address the internal structure of a method and hence should be considered by STAM designers, the first group of specifications highly affect the design of an efficient benchmarking environment since they focus on database characteristics for evaluation purposes. In particular, the specifications that are addressed in [TSPM98] with respect to type (i) are the following:

- *Spec 1: on the data type(s) supported.* Appropriate STAMs could support either point or non-point spatial objects. In some cases, point objects could be considered as special cases of non-point objects but this depends on the underlying modeling.
- *Spec 2: on the time dimension(s) supported.* A second classification concerns the time dimension(s) supported, i.e., valid and/or transaction time. Since at least one time dimension should be supported, spatiotemporal databases are classified in *valid-time*, *transaction-time*, and *bitemporal* ones.
- *Spec 3: on the dataset mobility.* Three cases are addressed, with respect to the motion of objects and the cardinality of the dataset through time, namely *evolving* (i.e., moving objects of a fixed cardinality through time), *growing* (i.e., static objects of varying cardinality through time), and *full-dynamic* (i.e., moving objects of varying cardinality through time) databases.
- *Spec 4: on the timestamp features.* Whether future instances could refer to past timestamps or not leads to a distinction between *chronological* and *dynamic* databases, i.e., collections of objects' instances (o_id, s_i, t_i) that either have or not to obey the rule of consecutive timestamps: $t_{i+1} > t_i$.

In the rest of the paper we study the case of *temporally degenerate* databases that obey the rule of consecutive timestamps, i.e., for each object in the database, the following inequality exists between the timestamp of the current instance t_i and that of the next instance t_{i+1} to be inserted into the database: $t_{i+1} > t_i$. The term *degenerate* refers to the characteristic that the valid time of object instances is identical to their transaction time. That is, an object is valid as long as it exists in the database. The problem that arises when no such rule exists[3] is clarified through the following example: Consider that two instances (o_id, s_i, t_i) and (o_id, s_j, t_j) of an object o have been inserted into the database (without a loss of generality, we assume that $t_i < t_j$) and no instance (o_id, s_k, t_k) exists, such that $t_i < t_k < t_j$. Hence [t_i , t_j) is the valid (and transaction) time of instance i. Let now assume that a new instance (o_id, s_l, t_l) is inserted into the database, such that $t_i < t_l < t_j$. Due to that action, (a) the valid time of instance i has to be changed from [t_i , t_j) to [t_i , t_l) and (b) the validity interval of the new instance l has to be set to [t_l , t_j). No straightforward support for those operations exists in current STAMs and, therefore, we currently leave

---

[3] Applicable to valid-time only since transaction-time always obeys that rule.

that case out of study. Note however, that this assumption is not made in the area of bitemporal databases [ST98]. Indeed in bitemporal access structures the rule is that, by definition, only transaction is monotonically increasing as discussed above. However, adding spatial features to bitemporal data is still an open area for research.

3.1. User requirements

Three (*Spec1* to *Spec3*) out of the above four specifications are orthogonal to each other. On the other hand, only the *chronological* case of *Spec4* is supported in this study, as declared earlier, and, as a result of that, we currently treat *transaction-* and *valid-* time under a uniform platform. Hence, we distinguish among 12 different database families (e.g. a *point* plus *transaction-time* plus *evolving* plus *chronological* database) according to the following options:

- *Spec1*: *point* vs. *region* database,
- *Spec2*: *transaction-* (or *valid-*) vs. *bitemporal* database,
- *Spec3*: *evolving* vs. *growing* vs. *full-dynamic* database,
- *Spec4*: *chronological* database.

    In order for the user of a benchmarking environment to generate a synthetic dataset, he/she should be able to (a) select one among the above database options and, then (b) tune the cardinality of the dataset and an appropriate set of parameters and distributions.

    A fundamental issue on generating synthetic spatiotemporal datasets is the definition of a *complete set of parameters* that control the evolution of spatial objects. Towards this goal, we first address the following three operations:

- *duration of an object instance*, which involves change of timestamps between consecutive instances,
- *shift of an object*, which involves change of spatial location (in terms of center point shift), and
- *resizing of an object*, involves change of an object's size (only applicable to non-point objects).

    In a more general case, the latter one could be regarded as *reshaping of an object*, as not only size but also shape could change. However, as the MBR is the most common approximation used by indices, we only consider that case, and thus shape changes are not an issue.

    A description of each operation follows. In particular, the goal to be reached is the calculation of the consecutive instances (o_id, s_i, t_i) of an object o (recall the definition in Section 1) starting from an initial instance (o_id, s_1, t_1). We also assume that the spatial workspace of interest is the unit square $[0,1)^2$ and time varies from 0 to 1 (i.e., the unit interval). For illustration reasons, in Figure 4 we visualize four instances of a time-evolving two-dimensional region object o and the corresponding projections on spatial plane and temporal axis, respectively.
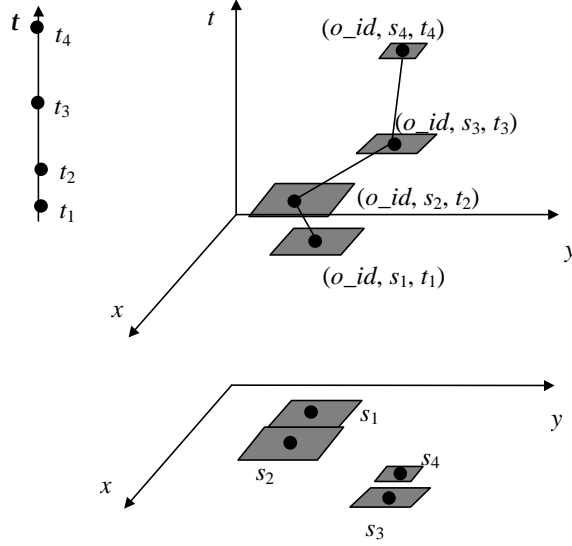
Figure 4: Consecutive instances of a time-evolving object *o* and the corresponding projections

3.2. Parameters involved

The *shift*, the *duration*, and the *resizing* of an object's instance are represented by the functions:

`duration(o_id,interval,current_timestamp,new_timestamp)`

`shift(o_id,Δcenter[],current_spacestamp_center,new_spacestamp_center)`

`resizing(o_id,Δextent[],current_spacestamp_extent[],new_spacestamp_extent[])`

which calculate `new_timestamp` (a numeric value), `new_spacestamp_center` (a 2-dimensional point), and `new_spacestamp_extent[]` (an array of 2 intervals), respectively, of an object `o_id`, as functions of the respective current values and three parameters, namely `interval`, `Δcenter[]`, and `Δextent[]`, respectively.

As an example, consider the object illustrated in Figure 4 and its initial position (o_id, s_1, t_1). Each consecutive spacestamp s_i and timestamp t_i (i = 2, 3, 4) depends on the previous one, s_{i-1} and t_{i-1} respectively, with respect to the following formulae: $t_i = t_{i-1} +$ `interval`$_i$, $s_{i.center.x} = s_{i-1.center.x} +$ Δ`center`$_i$`.x`, and $s_{i.extent.x} = s_{i-1.extent.x} + \Delta$`extent`$_i$`.x`

In summary, Table 1 lists the parameters of interest and their corresponding domains. All parameters should follow a (user-defined) distribution, such as the ones we discuss in the following subsection.

Parameter	Type	Domain
`interval`	number	$(0 \ldots 1)$
`Δcenter[]`	2-dimensional vector	$(-1 \ldots 1)^2$
`Δextent[]`	2-dimensional vector	$(-1 \ldots 1)^2$

Table 1: Parameters for generating time-evolving objects

7

3.3. Distributions

A benchmarking environment should support a wide set of well-established initial data distributions. Figure 5 illustrates three two-dimensional initial distributions, namely the uniform, the gaussian, and the skewed one.
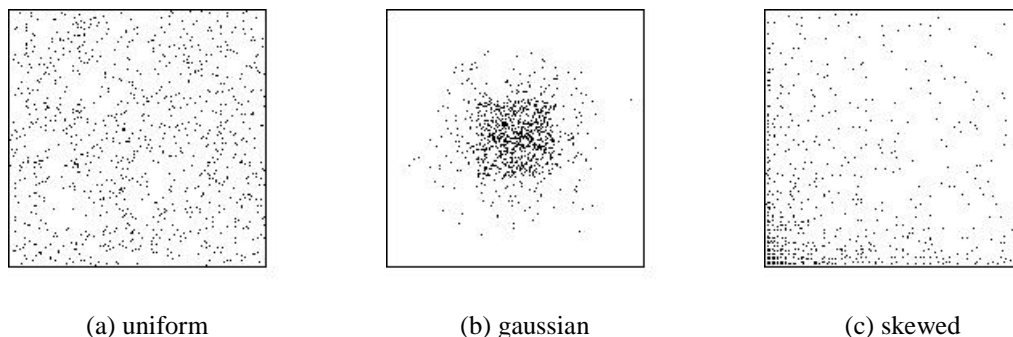

(a) uniform	(b) gaussian	(c) skewed

Figure 5: Basic statistical distributions in two-dimensional space

In addition to the initial spatial distributions, there are several other parameters that require some kind of statistical distribution, especially those mentioned above (Δcenter[], interval, and Δextent[]). Through careful use of, possibly different, distributions for the above parameters one may simulate several interesting scenarios, for instance, using a random distribution for the Δcenter[i] as well as for the interval, all objects would move equally fast (or slow) and uniformly on the map; whereas using a skewed distribution for the interval one would obtain a relatively large number of slow objects moving randomly, and so on. Also, by properly adjusting the distributions for each Δcenter[i], one may control the direction of the objects movement. For instance, by setting Δcenter[i] = Uniform(0,1) $\forall$i, one would obtain a scenario where the set of objects eventually converge to the upper-right corner of the unit workspace, irrespectively from the initial distributions, but using the "*adjustment*" approach (see subsection 4.1). Similarly, if one wants the objects moving towards some specific direction (e.g. East), he/she can adjust Δcenter and put lower and upper bounds for the center's generated value, as will be discussed in detail in the following section.

Among the distributions supported and illustrated in Figure 5, the uniform distribution only requires the minimum / maximum values while the other ones require extra parameters to be tuned by the user. In particular, the gaussian distribution needs *mean* and *variance* parameters as input and the skewed distribution needs a parameter to be declared, which controls the "skewedness" of the distribution.

In the following section, we adopt the issues discussed earlier in order to present an algorithm that generates synthetic spatiotemporal datasets for benchmarking purposes.

4. The GSTD algorithm

We propose an algorithm, called *Generate_Spatio_Temporal_Data* (GSTD), for generating time-evolving (i.e., moving) point or rectangular objects. For each object *o_id*, GSTD generates tuples of the format: (*o_id*, *t*, p_l, p_u), where *t* is the timestamp and p_l (p_u) is the lower (upper) coordinate point of the spacestamp. The GSTD algorithm is illustrated in Figure 6.

4.1. Description of the algorithm

GSTD gets several user-defined parameters as input:

- `N` and `D` correspond to the initial cardinality and density (i.e., the ratio of the sum of the areas of data rectangles over the workspace area) of the dataset,
- `starting_id` corresponds to the initial id number of the objects,
- `numsnapshots` corresponds to the time resolution of the workspace,
- `min_t` and `max_t` correspond to the domain of the `interval` parameter,
- `min_c[]` and `max_c[]` correspond to the domain of the Δ`center[]` parameter,
- `min_ext[]` and `max_ext[]` correspond to the domain of the Δ`extent[]` parameter,

and generates several tuples for each object, according to the following procedure:

> *"Each object is initially active and, for each one, new instances are generated as long as their timestamp t < 1; when all objects become inactive, the algorithm ends".*

During the initialization phase (lines 01-04), all objects' instances are initialized, such that their center points are randomly distributed in the workspace, based on the `distr_init()` distribution, and their extensions are either set to zero (in case of point datasets) or calculated according to `extent(N,D)` routine with respect to the input `N` and `D` parameters (in case of non-point datasets)[4].

During the main loop phase (lines 06-27), each new instance of an object is generated as a function of the existing one and the three parameters (`interval`, Δ`center[]`, Δ`extent[]`). Then, invalid instances (i.e., those with coordinates located outside the predefined workspace) can be manipulated in three alternative ways as described below. In order for a new instance to be generated, the `interval`, Δ`center[]` and Δ`extent[]` values are calculated by calling an `RNG(distr(),min,max)` routine, i.e., a random number generator that generates random numbers between `min` and `max` following a predefined `distr`, which is a statistical distribution, such as the ones discussed in subsection 3.3.

The `print_instance` function checks whether the current instance of an object has a timestamp value greater than or equal to the value in `next_snapshot`. If so, the coordinates of the instance (given by the `old_instance` variable) before the current instance are printed, using the apropriated timestamp (which depends on the `next_snapshot` variable). In addition, the value of the `next_snapshot` variable is properly adjusted. Otherwise, the current instance is not output.

---

[4] In other words, an appropriate `k=extent(N,D)` value is set to achieve an initial density `D` of the dataset with respect to initial cardinality `N`.

```
Generate_Spatio_Temporal_Data algorithm
Input:      values N, starting_id, numsnapshots, D, min_t, max_t
            arrays min_c[], max_c[], min_ext[], max_ext[]
            distributions distr_init(), distr_t(), distr_c(), distr_ext()
Output:     instance (id, t, lower_left_point, upper_right_point), validity_flag

begin
01  for each id in range [starting_id .. N+starting_id] do //initialization phase
02      Set t = 0, center[] = RNG(distr_init(), 0, 1), extent[] = extent(N, D)
03      Set active = TRUE
04  end-for
05  Set step = 1 / numsnapshots
06  for each id in range [starting_id .. N+starting_id] do  /* loop phase */
07      Set next_snapshot = step
08      while active do
            /* calculate delta-values and new instances */
09          Set interval = RNG(distr_t(), min_t, max_t)
10          Set Δcenter[] = RNG(distr_c(), min_c[], max_c[])
11          Set Δextent[] = RNG(distr_ext(), min_ext[], max_ext[])
12          Set old_instance = instance
13          update_instance(instance)
            /* check instances and output */
14          if t > 1 then
15              active = FALSE
16              print_instance(old_instance,current[i],next_snapshot)
17          else //check instance validity and output
18              Set validity_flag = valid(instance)
19              if validity_flag = FALSE and approach ≠ 'radar' then
20                  adjust_coords(instance, approach)
21              end-if
22              if t > next_snapshot then
23                  print_instance(old_instance,current[i],next_snapshot)
24              end-if
25          end-if
26      end-while
27  end-for
end.
```

Figure 6: The GSTD algorithm

Obviously, it is possible that a coordinate may fall outside the workspace; GSTD manipulates *invalid* instances according to one among three alternative approaches:

- the *'radar'* approach, where coordinates remain unchanged, although falling beyond the workspace,
- the '*adjustment*' approach, where coordinates are adjusted (according to linear interpolation) to fit the workspace, and
- the '*toroid*' approach, where the workspace is assumed to be toroidal, as such once an object traverses one edge of the workspace, it enters back in the "opposite" edge.

In the first case, the output instance is appropriately flagged to denote that invalidity but the next generated instance is based on that. On the other hand, in the other two cases, it is the modified instance that is stored in the resulting data file and used for the generation of the next one. Notice that in the

'*radar*' approach, the number of objects present at each time instance may vary.

The three alternative approaches are illustrated in Figure 7 for the example of Figure 4. For simplicity, only the centers are illustrated; black (grey) locations represent valid (invalid) instances. In the example of Figure 7a, the '*radar*' fails to detect s_3, hence it is not stored but the next location s_4 is based on that. Unlike '*radar*', the other two approaches calculate a valid instance s_3' to be stored in the data file which, in turn, is used by GSTD for the generation of s_4. It is interesting to watch the behavior of s_4 in Figure 7c, where the calculated location finally stored (s_4') is actually identical to that in Figure 7a, as the effect of two consecutive calculations for s_3' and s_4'.

(a) '*radar*'   (b) '*adjustment*'

(c) '*toroid*'

Figure 7: GSTD manipulation of invalid instances

4.2. Examples of generated datasets

As mentioned earlier, real world examples of (point or region) spatiotemporal datasets include trajectories of humans, animals, or vehicles, e.g. detected by a global positioning system (GPS), digital simulations of flights or battles, weather forecast and monitoring of fire or hurricane fronts. For instance, detecting vehicle motion by GPS and storing the whole trajectory in a database is a typical every day life example. However, different motion scenarios correspond to different datasets which an efficient structure should be evaluated on. Random versus biased direction, fast versus slow motion are some of the parameters that result to totally different applications.

In this subsection, in order to simulate some of those scenarios, we present six example datasets consisting of point or rectangle objects generated by GSTD. For all files the following parameters were set: `N` = 1000, `D` = 0 or 0.5 (for points or rectangles, respectively), `numsnapshots` = 100. Illustrated snapshots correspond to t = 0, 0.25, 0.50, 0.75, and 1. Table 3 presents the non-fixed input parameters and the generated snapshots for each file. Scenarios 1 and 2 follow the '*toroid*' and '*radar*' approach, respectively, to manipulate invalid instances, while scenarios 3 through 6 follow the '*adjustment*' approach.
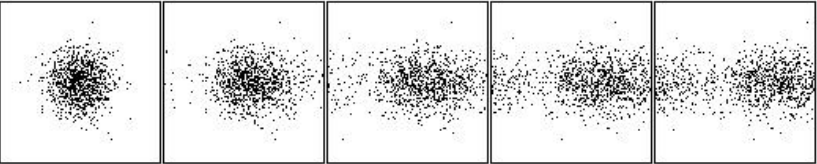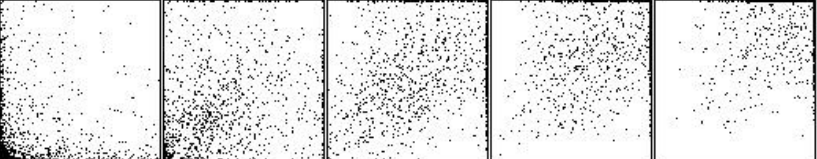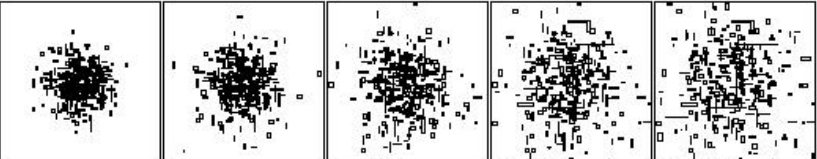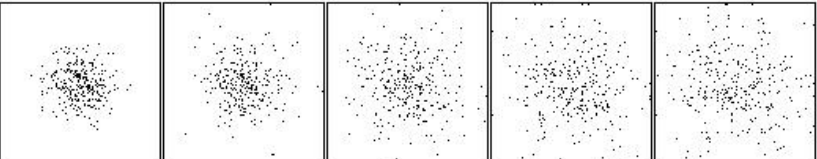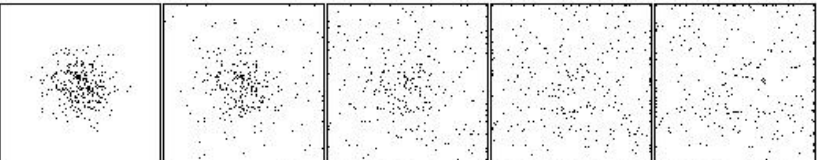
Distributions of Parameters	Snapshots
| ```
distr_init = Gaussian(0.5,0.1)
  interval = Gaussian(0,0.5)
  Δcenter[x] = Uniform(0,0.3)
   Δcenter[y] = Uniform(0,0)
    Δextent[x] = Uniform(0,0)
    Δextent[y] = Uniform(0,0)
``` | |
| **scenario 1**: points moving from center to East (*'toroid'* approach) | |
| ```
distr_init = Gaussian(0.5,0.1)
  interval = Gaussian(0,0.5)
 Δcenter[x] = Uniform(0,0.4)
 Δcenter[y] = Uniform(0,0.4)
  Δextent[x] = Uniform(0,0)
  Δextent[y] = Uniform(0,0)
``` | |
| **scenario 2**: points moving from center to NorthEast (*'radar'* approach) | |
| ```
   distr_init = Skewed(1)
 interval = Gaussian(0,0.2)
Δcenter[x] = Uniform(0,0.3)
Δcenter[y] = Uniform(0,0.3)
 Δextent[x] = Uniform(0,0)
 Δextent[y] = Uniform(0,0)
``` | |
| **scenario 3**: points moving from SouthWest to NorthEast | |
| ```
distr_init = Gaussian(0.5,0.1)
  interval = Gaussian(0,0.5)
Δcenter[x] = Uniform(-0.2,0.2)
Δcenter[y] = Uniform(-0.2,0.2)
Δextent[x] =Uniform(-0.01,0.01)
Δextent[y] =Uniform(-0.01,0.01)
``` | |
| **scenario 4**: rectangles moving (and resizing) randomly | |
| ```
distr_init = Gaussian(0.5,0.1)
  interval = Gaussian(0,0.5)
Δcenter[x] = Uniform(-0.2,0.2)
Δcenter[y] = Uniform(-0.2,0.2)
  Δextent[x] = Uniform(0,0)
  Δextent[y] = Uniform(0,0)
``` | |
| **scenario 5**: points moving randomly (low speed) | |
| ```
distr_init = Gaussian(0.5,0.1)
  interval = Gaussian(0,0.5)
Δcenter[x] = Uniform(-0.4,0.4)
Δcenter[y] = Uniform(-0.4,0.4)
  Δextent[x] = Uniform(0,0)
  Δextent[y] = Uniform(0,0)
``` | |
| **scenario 6**: points moving randomly (high speed) | |

Table 3: Example files generated by GSTD

Scenarios 1 and 2 illustrate points with initial gaussian distribution moving towards East and NorthEast, respectively. In the former case, where the *toroidal* world model was used, when the points traverse the right edge, they enter back in the left side of the map. Notice that to force the points moving to the East, $\Delta\texttt{center[y]} = 0$ and $\Delta\texttt{center[x]} > 0$. In the latter case, where the 'radar' approach is simulated, the points move towards NorthEast and some of them fall beyond the upper-right corner (some quite early due to their speed), in fact some points move beyond the map. Notice that since $\Delta\texttt{center[]}$ is always > 0, those points will never reappear in the map.

Scenario 3 illustrates the initially skewed distribution of points and the movement towards NorthEast. As the 'adjustment' approach was used, the points concentrate around the upper-right corner. Scenario 4 includes rectangles initially located around the middle point of the workspace, which are moving and resizing randomly. The randomness of *shift* and *resizing* is guaranteed by the `Uniform(min,max)` distribution used for $\Delta\texttt{center[]}$ and $\Delta\texttt{extent[]}$, where `abs(min)=abs(max)` > 0.

Finally, scenarios 5 and 6 exploit the *speed* parameter of a moving dataset as a function of the GSTD input parameters. By increasing (in absolute values) the `min` and `max` values of $\Delta\texttt{center[]}$, a user can achieve 'faster' objects while the same could happen by decreasing the `max_t` value that affects `interval`. Thus, the speed of the dataset is considered to be a *meta-information* since it could be derived by the knowledge of the primitive parameters. Similarly, the direction of the dataset can be controlled, as presented in scenarios 1 through 3.

Alternatively, if the user's application makes necessary the conjunction of two (or more) scenarios, as for instance, a population of MBRs with only a small percentage of them moving towards some direction and the rest ones being static, two individual scenarios can be generated according to the above by properly setting the two `starting_id` input parameters and then merged, which is a straightforward task. Bottomline, by properly adjusting the parameters of Table 1, one can yield a scenario that fits his/her needs.

5. Discussion and Related Work

An alternative straightforward algorithm for generating `N` time-evolving objects would include the calculation of the spacestamp of each object at each snapshot, thus leading to an output consisting of $T = $ `N · numsnapshots` tuples. Our approach outperforms that since it outputs a limited number T' of tuples $(T' \ll T)$, i.e., the necessary ones in order to reproduce the dataset motion.

However, a fundamental question arises: based on the knowledge of two instances (o_id, s_i, t_i) and (o_id, s_{i+1}, t_{i+1}) that correspond to consecutive timestamps, what is the location of an object at a time t_j, such that $t_i < t_j < t_{i+1}$? As an example, recall the instances of the object o illustrated in Figure 4. The status of its spacestamp between e.g. t_i and t_{i+1} is a 'fuzzy' issue. Two alternatives may be followed:

- *projection*: the spacestamp is considered to be static and equal to the one at time t_i,
- *linear interpolation*: the spacestamp is considered to be moving with respect to a start- (at time t_i) and an end- (at time t_{i+1}) position.

Both alternatives find applications in real world; cadastral systems, on the one hand, versus

navigational systems, on the other hand, are popular examples[5]. Figure 8 illustrates the two alternative scenarios for the example of Figure 4.
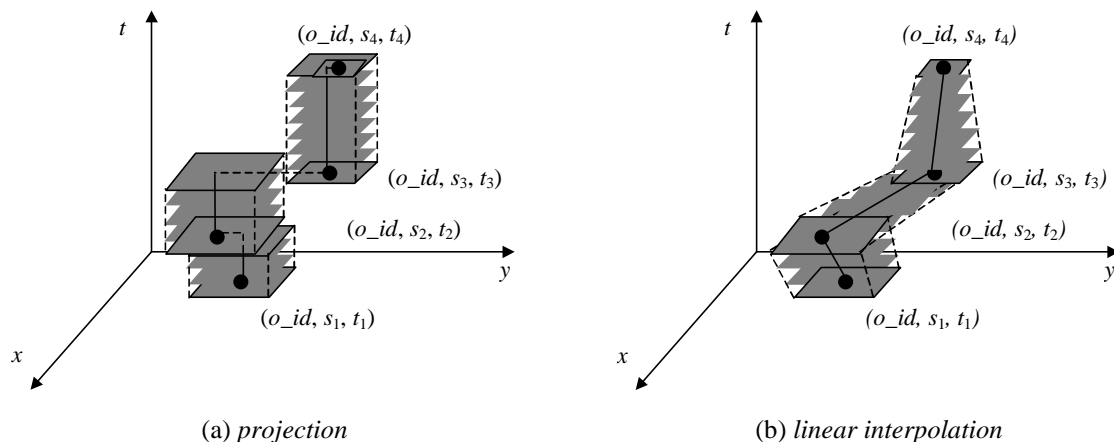

(a) *projection*   (b) *linear interpolation*

Figure 8: Alternative scenarios for the location of an object between two timestamps

In any case, detecting the status of object o at a time instance during (t_1, t_2) is an open issue. We argue that the GSTD algorithm proposed earlier is independent of that issue. Actually, it generates a series of instances regardless of such an issue. On the other hand, it is a visualization tool or a STAM construction algorithm that needs to support one or both alternative scenarios. Since in this study we are interested in spatiotemporal databases that follow the rule of consecutive timestamps, the knowledge of both the current and the new instances of an object, as supported by GSTD (line 13), are sufficient to deal with both alternatives.

The need for independent platforms for benchmarking purposes or, in general, experiment management has been already addressed in the past [SFGM93, ILGP96]. Such a need arises when a researcher aims to make a 'fair' performance study or experimentation without the dilemma of building his/her own datasets for this purpose. Although extended related work is found in traditional database benchmarks and data generators (e.g. [BDT83, GSE+94]), in the field of spatial databases it is very limited [SFD93, Pat97, GOP+98]. Moreover, when motion is introduced to support spatiotemporal databases, to our knowledge, no related work exists.

The most relevant to our work is the '*A La Carte*' benchmark [GOP+98]. It is a WWW-based tool consisting of a rectangle generator that builds datasets based on user defined parameters (cardinality, coverage, coordinates' distributions) and an experimentation module that runs experiments on either user built or stored sample datasets (including parts of the Sequoia 2000 storage benchmark [SFGM93]). The module is actually a *spatial join* performance evaluator that supports several spatial join strategies.

6. Conclusion and Future Work

STDBMS require appropriate indexing techniques on spatiotemporal data. Although conceptually the problem seems to be easy to solve, several issues arise when one attempts to adopt a spatial indexing method to organize time-evolving objects by just adding an extra dimension for time. Therefore, a

---

[5] Linear interpolation assumes that a linear function represents boundary points' motion, i.e., intermediate locations are linear to the start- and end- points. Higher-order polynomial problems are hardly modeled.

limited number of STAMs have been proposed in the literature as briefly surveyed in Section 2.

The effort towards the design and implementation of a benchmarking environment in order to provide performance comparison of STAMs leads to the need of collecting a variety of appropriate synthetic and real spatiotemporal datasets. However, in accordance to the design of efficient methods, generating efficient synthetic datasets is not a straightforward extension of generating spatial data, such as the ones that have been thoroughly used for experimental purposes in the spatial database literature. At a first step, several specifications that identify the type of the dataset have to be addressed and, at a second step, a set of parameters and corresponding distributions have to be tuned by the user. More specifically, we have discussed three operations, namely *duration of an object instance*, *shift* and *resizing of an object* (the latter one applicable to non-point objects) and derived a set of three parameters, namely `interval`, Δ`center`, and Δ`extent`, which control the evolution of a spatial object through time in satisfactory terms.

Based on those parameters, we have designed and implemented the GSTD algorithm that generates sets of moving points or rectangles according to users' requirements, thus providing a tool that simulates a variety of possible scenarios. Some of those scenarios have been illustrated and discussed in Section 4. GSTD also includes alternative methodologies to support *invalid* instances, i.e., those with coordinates falling outside the workspace.

This study continues the work initiated in [TSPM98] towards a full and interactive support tool for designing, implementing, and evaluating access methods for the purposes of STDBMS. We are currently working on a WWW environment to make GSTD available to all researchers through the Internet (mirror sites: `http://www.dblab.ece.ntua.gr/~theodor/GSTD/` and `http://www.dcc.unicamp.br/~mario/GSTD/`). We are also investigating some additional functionality on GSTD. For example, users may want to specify a movement flow to a specific point *p* in the workspace. Although, given the target *p*, it is not a complicate task, it is a specific implementation to a specific scenario. We currently study the parameterization of such specific scenarios by permitting GSTD input parameters to be (user-defined) functions rather than fixed values. Such an extension will enhance GSTD flexibility to simulate a variety of real applications.

Acknowledgments

References

[BDT83]   D. Bitton, D. J. DeWitt, C. Turbyfill, "Benchmarking Database Systems: A Systematic Approach", *Proceedings of the 9th Conference on Very Large Data Bases* (VLDB), 1983.

[BKSS90]   N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", *Proceedings of ACM SIGMOD Conference*, 1990.

[EGSV98]   M. Erwig, R. H. Güting, M. Schneider, M. Vazirgiannis, " Abstract and Discrete Modeling of Spatio-Temporal Data Types", *Proceedings of the 6th ACM International Workshop on Geographical Information Systems* (ACM-GIS), 1998.

[GG98]   V. Gaede, O. Günther, "Multidimensional Access Methods", ACM Computing Surveys, 30(2): 170-231, June 1998.

[GOP+98]   O. Günther, V. Oria, P. Picouet, J.-M. Saglio, M. Scholl, "Benchmarking Spatial Joins *A La Carte*", *Proceedings of the 10th International Conference on Scientific and Statistical Database Management* (SSDBM), 1998.

[GSE+94]   J. Gray, P. Sundaresan, S. Englert, K. Backlawski, P. J. Weinberger, "Quickly Generating Billion-Record Synthetic Databases", *Proceedings of ACM SIGMOD Conference*, 1994.

[Gut84]   A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching", *Proceedings of ACM SIGMOD Conference*, 1984.

[ILGP96]   Y. Ioannidis, M. Livny, S. Gupta, N. Ponnekanti, "ZOO: A Desktop Experiment Management Environment", *Proceedings of the 22nd Conference on Very Large Data Bases* (VLDB), 1996.

[KF94]   I. Kamel, C. Faloutsos, "Hilbert R-tree: An Improved R-tree Using Fractals", *Proceedings of the 20th Conference on Very Large Data Bases* (VLDB), 1994.

[KTF98]   A. Kumar, V.J.Tsotras, C.Faloutsos, "Designing Access Methods for Bi-temporal Databases", IEEE Transactions on Knowledge and Data Engineering, 10(1): 1- 20, January - February 1998.

[LS89]   D. Lomet, B. Saltzberg, "Access Methods for Multiversion Data", *Proceedings of ACM SIGMOD Conference*, 1989.

[MK90]   Y. Manolopoulos, G. Kapetanakis, "Overlapping B⁺-trees for Temporal Data", *Proceedings of the 5th Jerusalem Conference on Information Technology* (JCIT), 1990.

[NS98]   M. A. Nascimento, J. R. O. Silva, "Towards Historical R-trees", *Proceedings of ACM Symposium on Applied Computing* (ACM-SAC), 1998.

[NST98]   M. A. Nascimento, J. R. O. Silva, Y. Theodoridis, "Access Structures for Moving Points", TIMECENTER Technical Report TR–33, August 1998.

[Pat97]   J. Patel et al., "Building a Scalable Geo-Spatial DBMS: Technology, Implementation, and Evaluation", *Proceedings of ACM SIGMOD Conference*, 1997.

[Sam84]   H. Samet, "The Quadtree and Related Hierarchical Data Structures", *ACM Computing Surveys*, 16(2): 187-260, 1984.

[SFD93]   M. Stonebraker, J. Frew, J. Dozier, "The SEQUOIA 2000 Project", *Proceedings of the 3rd International Symposium on Advances in Spatial Databases* (SSD), 1993.

[SFGM93]   M. Stonebraker, J. Frew, K. Gardels, J. Meredith, "The SEQUOIA 2000 Storage Benchmark", *Proceedings of ACM SIGMOD Conference*, 1993.

[ST98]   B. Salzberg, V. Tsotras, "A Comparison of Access Methods for Temporal Data", to appear in

ACM Computing Surveys. Also available as TIMECENTER Technical Report TR-18.

[SWCD97]   A. P. Sistla, O. Wolfson, S. Chamberlain, S. Dao, "Modeling and Querying Moving Objects", *Proceedings of the 13th IEEE Conference on Data Engineering* (ICDE), 1997.

[T98]   N. Tryfona, "Modeling Phenomena in Spatiotemporal Applications: Desiderata and Solutions", *Proceedings of the 9th International Conference on Database and Expert Systems Applications* (DEXA), 1998.

[TSPM98]   Y. Theodoridis, T. Sellis, A. Papadopoulos, Y. Manolopoulos, "Specifications for Efficient Indexing in Spatiotemporal Databases", *Proceedings of the 10th International Conference on Scientific and Statistical Database Management* (SSDBM), 1998.

[TVM98]   T. Tzouramanis, M. Vassilakopoulos, Y. Manolopoulos, "Overlapping Linear Quadtrees: a Spatiotemporal Access Method", *Proceedings of the 6th ACM International Workshop on Geographical Information Systems* (ACM-GIS), 1998.

[TVS96]   Y. Theodoridis, M. Vazirgiannis, T. Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications" *Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems* (ICMCS), 1996.

[WXCJ98]   O. Wolfson, B. Xu, S. Chamberlain, L. Jiang, "Moving Objects Databases: Issues and Solutions", *Proceedings of the 10th International Conference on Scientific and Statistical Database Management* (SSDBM), 1998.

[XHL90]   X. Xu, J. Han, W. Lu, "RT-tree: An Improved R-tree Index Structure for Spatiotemporal Databases", *Proceedings of the 4th International Symposium on Spatial Data Handling* (SDH), 1990.

[ZMR96]   J. Zobel, A. Moffat, K. Ramamohanarao, "Guidelines for Presentation and Comparison of Indexing Techniques", *ACM SIGMOD Record*, 25(3): 10-15, 1996.