

# Properties Of Space Filling Curves And Usage With UB-trees

Tomáš Skopal, Michal Krátký, Václav Snášel

Department of Computer Science, Technical University Ostrava, Czech Republic  
tomas.skopal@vsb.cz, michal.kratky@vsb.cz, vaclav.snasel@vsb.cz

**Abstract.** In this paper we want to investigate certain properties of space filling curves and their usage with UB-trees. We will examine several curve classes according to range queries in UB-trees. Our motivation is to propose a new curve for the UB-tree which will improve the range queries efficiency. In particular, the address of this new curve is constructed using so-called proportional bit interleaving.

**Keywords:** space filling curve, UB-trees, space partitioning, Z-curve

## 1 Introduction

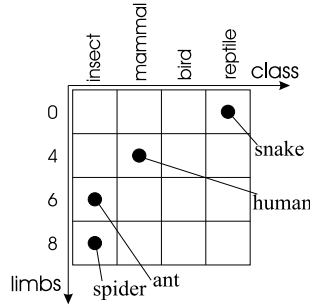
In Information Retrieval there is often requirement on data clustering or ordering and their consequential indexing. This requirement arises when we need to search for some objects in large amounts of data. The process of data clustering→indexing gathers similar data together, thereby allows effective searching in large data volumes.

Specific approaches of data clustering were established on the vector space basis where each data object is represented as a vector of its significant attributes. Such object vectors can be stored as points/vectors within a multidimensional vector space. In this article we are going to examine indexing method for discrete multidimensional vector spaces based on *space filling curves*. A space filling curve allows linearize a multidimensional space in such way that to each point in space is assigned single unique value – i.e. *address* on curve. Thus, the space could be considered as partitioned and even ordered due to the curve ordering. There exists one indexing structure which combines the principles of space filling curves and techniques of data indexing. This structure is called UB-tree.

### 1.1 Vector Spaces

**Definition 1.** Let  $\Omega = D_1 \times D_2 \times \dots \times D_n$  is a  $n$ -dimensional vector space. The set  $D_i$  is called the domain of dimension  $i$ . The vector (point, tuple)  $x \in \Omega$  is a  $n$ -tuple  $\langle a_1, a_2, \dots, a_n \rangle \in \Omega$ . In other words  $x$  can be interpreted as a point at coordinates  $(a_1, a_2, \dots, a_n)$  within  $n$ -dimensional space. The coordinate  $a_i$  is also called attribute value and can be represented as a binary number (string) of length  $l_i$ . Thus, to each domain  $D_i$  is assigned a bit-size  $l_i$  and its cardinality  $c_i = 2^{l_i}$ . Then holds  $0 \leq a_i \leq c_i - 1$ .

In discrete vector space  $\Omega$  we use integer coordinates. Usually, the space domains are also finite due to limited capacity of integer attribute. In figure 1 we see two-dimensional space "animal" where the first dimension represents "animal class" and the second "limb count". Animal "ant" is represented with vector  $[0,2]$  ([insect, 6] respectively).



**Fig. 1.** Two-dimensional vector space  $4 \times 4$

Representing objects as vectors in space brings following possibilities:

- Formal apparatus of vector spaces allows objects to be treated uniformly. Object is vector and there are defined certain operations on vectors in vector space.
- In metric vector spaces we can measure *similarity* between two objects (distance of vectors respectively)
- In vector space we can easily construct *range queries*

## 1.2 Space Partitioning

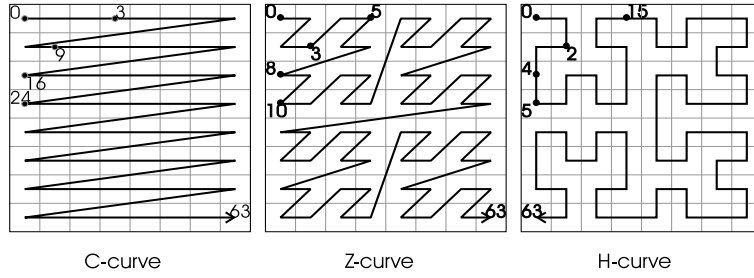
Because of the requirements of data indexing we need to transform the objects within vector space  $\Omega$  into some hierarchical index structure. This space transformation means some kind of space partition which in turn produces space subregions.

**Definition 2.** (*space filling function*)

We call a function  $f : S \subset \mathcal{N}_0 \rightarrow \Omega$  a *space filling function*, if  $f(S) = \Omega$  is a bijective function. Ordinal number  $a$ , where  $f(a) = o_i, o_i \in \Omega$  is called *address of vector  $o_i$  on curve  $f(S)$* .

**Lemma 1.** A *space filling function* defines one-dimensional ordering for a multidimensional space.

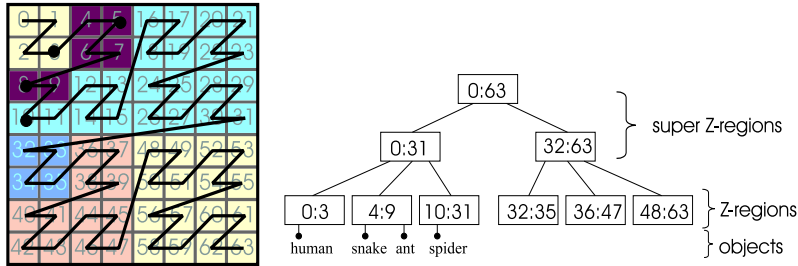
Space filling curves order points in space and may so define *regions* in the space. A region is spatially determined with an interval  $[\alpha : \beta], \alpha \leq \beta$  on the curve ( $\alpha, \beta$  are addresses). In figure 2 we can see our example space (extended to  $8 \times 8$ ) filled with three types of curves. Suppose we want to find the smallest region covering objects "snake" and "ant", we would obtain  $[3:16]$  for C-curve,  $[5:8]$  for Z-curve and  $[4:15]$  for H-curve.



**Fig. 2.** Compound (C) curve, Lebesgue (Z) curve, Hilbert (H) curve and addresses of points within our "animal" space

### 1.3 Universal B-trees

The idea of UB-tree incorporates  $B^+$ -tree and the space filling curves. Inner nodes of  $B^+$ -tree contain hierarchy of curve regions. Region of inner node always spatially covers all regions of its children. Regions on the same depth level in tree do not overlap. In leaves are stored data objects (their vectors respectively). The structure of UB-tree (with Z-curve) for our example is shown in figure 3. We can see that the objects on the leaf level are ordered left-to-right – the same way as on Z-curve. Further informations about UB-trees can be found in [Ba97, Ma99].



**Fig. 3.** Structure of UB-tree

So far, the published papers about UB-tree mentioned only the possibility of Z-curve. In following analysis we are going to examine also another curve orderings.

## 2 Properties of Space Filling Curves

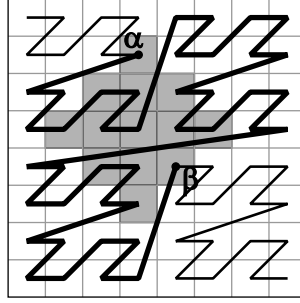
In [Ma99] we can find a curve quality criterion based on curve symmetry measure. This symmetry is somehow connected with *selfsimilarity* concept taken from fractal geometry.

We will introduce another aspects of curve quality classification. Our goal is to find ideal curve for range queries in vector space or actually in UB-tree.

## 2.1 Measure of Utilization

Range query processing in vector space or in UB-tree must examine each region which intersects query area. We can assume that large region overlaps will produce unnecessary space searching and thus they cause high access efficiency costs.

To reduce these costs we've developed *measure of utilization* which allows to rate each curve and choose the best one. In figure 4 we'll see greyed query area and the smallest possible region that entirely covers given query area.



**Fig. 4.** Smallest possible region  $[\alpha : \beta]$  covering entire query area

**Definition 3.** Let  $A(o_i)$  is a query area centered on coordinates of  $o_i \in \Omega$ . Let  $R(A(o_i))$  is the smallest possible region covering  $A(o_i)$ . Then utilization of  $A(o_i)$  is defined as

$$u(A(o_i)) = \frac{\text{volume}(A(o_i))}{\beta_{R(A(o_i))} - \alpha_{R(A(o_i))}}$$

and average utilization of space  $\Omega$  with query area  $A$  and space filling curve  $f$  is defined as

$$\text{avgutil}_{(\Omega, A, f)} = \frac{\sum_{o_i=f(0)}^{f(\text{volume}(\Omega))} u(A(o_i))}{\text{volume}(\Omega)}$$

**Notes** Average utilization factor helps us to find better curve from the range query point of view. Curve with *avgutil* close to 1 means that searching the vector space by range query processing is effective – no unnecessary space is searched. Interesting consequence of high rated curves is that points that are near in space (according to some metric) are also near on the curve (according to order).

**Choosing query area** The shape of range query area must comply with the nature of range queries. In metric vector spaces we search for similar objects given by some threshold distance value. In non-metric vector spaces we search for objects within coordinate ranges. The former aspect represents  $n$ -dimensional sphere and the latter  $n$ -dimensional

block. As we can see in figure 4 we've chosen  $n$ -dimensional sphere for further analysis.

Volume of sphere in multidimensional discrete space is defined as follows: Let  $K(r)$  denote number of points with integer coordinates contained within circle, i.e. circle volume in discrete space.

**Theorem 1.** (Gauss) in [Cha69]  
For  $K(r)$  the following asymptotic formula holds:

$$K(r) = \pi r^2 + \Delta(r)$$

$$\Delta(r) = O(r)$$

The circle consists of all the points satisfying  $x_1^2 + x_2^2 + \dots + x_n^2 \leq r^2$ . The volume of a sphere in  $R^n$  is a function of the radius  $r$  and will be denoted as  $V_n(r)$ . We know that  $V_1(r) = 2r$ ,  $V_2(r) = \pi r^2$  and  $V_3(r) = \frac{4}{3}\pi r^3$ . To calculate  $V_n(r)$  (changing to polar coordinates) we get

$$V_n(r) = \int_0^r \left( \int_0^{2\pi} V_{n-2}(\sqrt{r^2 - s^2}) s d\theta \right) ds$$

By using induction, closed forms for calculation of higher dimensional spheres can be derived

$$V_{2n}(r) = \frac{\pi^n \cdot r^{2n}}{n!}$$

$$V_{2n+1}(r) = \frac{2^{2n+1} \cdot n! \cdot \pi^n \cdot r^{2n+1}}{(2n+1)!}$$

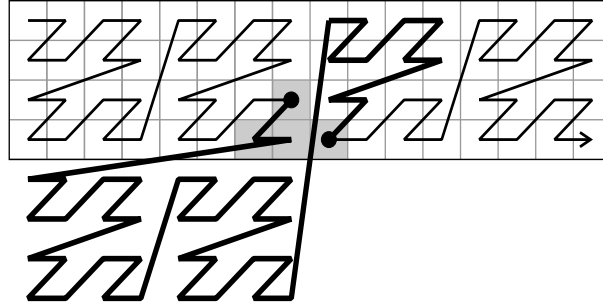
From theorem 1 we get the volume of  $n$ -dimensional sphere

$$K_n(r) = V_n(r) + O(V_{n-1}(r))$$

## 2.2 Curve Dependency Classification

We have find out that dependency on certain vector spaces characteristics has important influence on the average utilization factor. We'll venture to proclaim that the higher dependency on space characteristics, the higher average utilization. We have classified this dependency into three space filling curve classes:

**Dimension dependent curves** First class of curves takes in account only the dimension of vector space, e.g. classical Z-curve. This type treats the space as a multidimensional cube and is not suitable for spaces with different domain cardinalities. In figure 5 we can see that the curve overlap the real space. It is obvious that the average utilization of that curve will be very low due to large smallest covering regions.



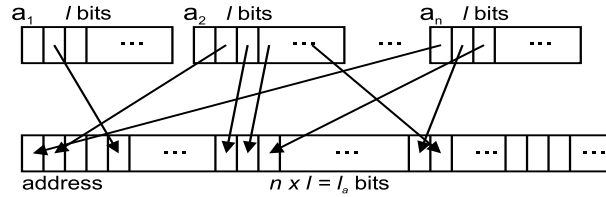
**Fig. 5.** Dimension dependent curve, e.g. Z-curve and one of the smallest covering regions

Address computation is also only dimension dependent. On the input stand  $n$  attributes, bit length is uniform –  $l$ . In figure 6 is shown one type of address construction, i.e. bit interleaving which is simply the permutation of bit vector  $\mathbf{a}$  of all attributes (concatenated to single vector). To every bit in every attribute is assigned a position in the resultant address. Address construction based on permutation can be easily realized with permutation matrix  $A$ .

$$addr = \mathbf{a} \cdot A$$

Permutation matrix can be created from unitary square matrix by multiple column (or row) swap. Reconstruction of original attributes from address is also simple – using the inverse matrix  $A^{-1}$  which is equal (if  $A$  is orthonormal – and permutation matrix is always orthonormal) to the transposed matrix, i.e.  $A^T = A^{-1}$ . Then

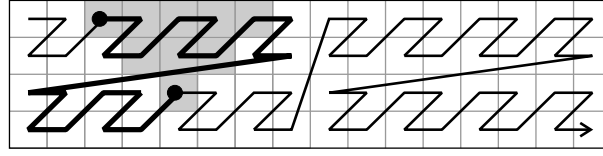
$$\mathbf{a} = A^T \cdot addr$$



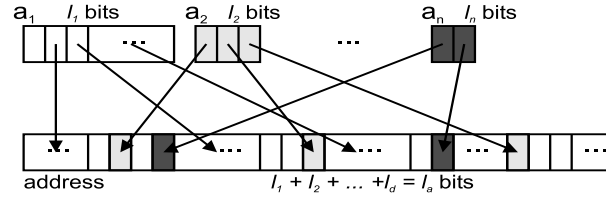
**Fig. 6.** Bit interleaving for dimension dependent address

**Domain dependent curves** The second class is not only dimension dependent but also domain dependent. Curves are constructed with considerations to domain cardinalities. Curves of that type do not overlap given vector space (e.g. C-curve). Address construction for domain dependent curves can be based on permutation matrix as well.

The curve for UB-tree we have announced in the beginning is one of this type and is called PZ-curve. PZ-curve advances the original Z-curve



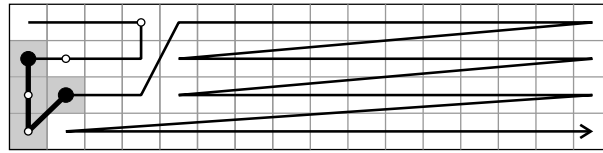
**Fig. 7.** Domain dependent curve, e.g. PZ-curve and one of the smallest covering regions



**Fig. 8.** Bit interleaving for domain dependent address

where the new quality we call *proportionality* of PZ-address, i.e. attribute bits are interleaved proportionally to each domain (see figures 7, 8). The PZ-address is intimately described in the next section.

**Data dependent curves** Third class of curves is dependent on everything known in the space even on the data stored within. However, this type is rather theoretical and we present it here only for notion and future motivation.



**Fig. 9.** Data dependent curve

Construction of address for data dependent curve is very complex. Generally, every bit of the output address is evaluated as a function of all the coordinates in input. Even if we'd accomplish address computation there is another complication. The curve is data dependent and therefore it must be completely recomputed after adding new data.

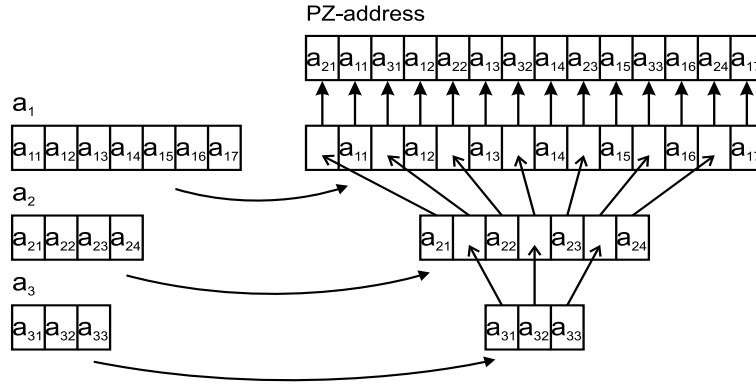
### 2.3 PZ-address

Construction of PZ-address is based on proportional bit interleaving. We can describe the interleaving with following simplified algorithm:

1. Vector of PZ-address is empty (all bit positions are empty). Order of attributes  $a_i$  is chosen as a parameter,  $i \in I$ .
2. Bits of attribute  $a_i$  are uniformly dispersed over the empty bit positions within the PZ-address vector.
3. Newly occupied bit positions are no longer empty – the vector of PZ-address is being filled step by step.
4. Step 2 is repeated until index set  $I$  is exhausted.

**Note:** The order of attribute processing is important. Attributes that are processed at first are more accurately dispersed into PZ-address.

For synoptic idea of the algorithm see figure 10.



**Fig. 10.** PZ-address construction. First, bits of  $a_1$  are dispersed into the empty PZ-address vector. Second,  $a_2$  is dispersed over the rest empty positions. Last attribute (here  $a_3$ ) is actually not dispersed but copied.

**Formal description** Let's have  $n$  attributes (coordinates in  $n$ -dimensional space). Attribute  $a_i$  is represented as a bit vector of length  $l_i$ . PZ-address (vector  $PZaddr$  with length  $l_a = \sum_{i=1}^n l_i$ ) is computed using following permutation matrix:

$$PZaddr = (a_{11} \ a_{12} \ \dots \ a_{21} \ a_{22} \ \dots \ a_{ij}) \cdot \begin{pmatrix} \underbrace{0 \ 0 \ \dots \ 0}_{ord(a_{11})-1} & 1 & 0 & 0 & 0 & \dots \\ \underbrace{0 \ 0 \ 0 \ 0 \ 0 \ \dots \ 0}_{ord(a_{12})-1} & & 1 & 0 & \dots \\ \vdots & & & & & \\ \underbrace{0 \ 0 \ 0 \ \dots \ 0}_{ord(a_{21})-1} & 1 & 0 & 0 & 0 & \dots \\ \vdots & & & & & \\ \underbrace{0 \ 0 \ 0 \ 0 \ \dots \ 0}_{ord(a_{ij})-1} & 1 & 0 & 0 & \dots \end{pmatrix}$$

where  $ord(a_{ij})$  is the position of  $j$ -th bit of attribute  $a_i$  in the resultant PZ-address.

$$ord(a_{ij}) = emptypos(i, j \cdot disperse(a_i))$$

where  $disperse(a_i)$  is the uniform bit dispersion of  $a_i$  over the remaining empty positions in PZ-address.

$$disperse(a_i) = integer \left( \frac{\sum_{k=i}^n l_k}{l_i} \right)$$

and where  $emptypos(i, k)$  is the  $k$ -th empty bit position in PZ-address after the attribute  $a_{i-1}$  is processed.

$$emptypos(i, k) = \min(F(i), k)$$

$F(i)$  is the set function ( $F(i) \subset \mathcal{N}$ ) of all empty positions in PZ-address before attribute  $a_i$  is processed.

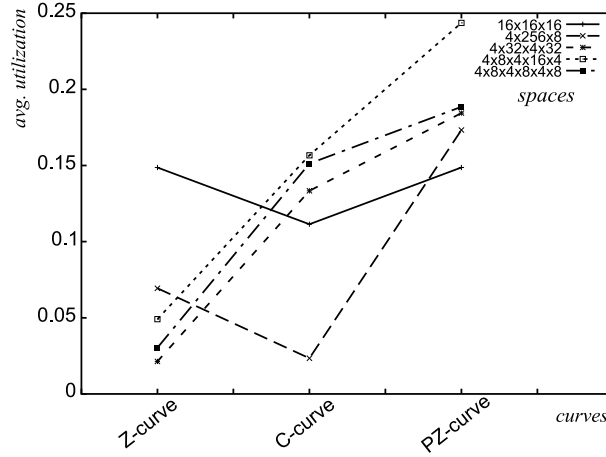
$$F(1) = \{1, 2, \dots, l_a\} \quad F(i+1) = F(i) - \bigcup_{j=1}^{l_i} \{ord(a_{ij})\}$$

$\min(A, k)$  is the  $k$ -th minimum of an ordered set  $A$

$$\min(A, 1) = \min(A) \quad \min(A, k) = \min(A - \bigcup_{q=1}^{k-1} \{\min(A, q)\})$$

## 2.4 Test results

Figure 11 shows us the influence of space dependency on particular curves and also their types. Proposed PZ-curve has relatively high average utilization rate, thus seems to be suitable for usage with UB-trees.



**Fig. 11.** Test results – with growing dependency grows also average utilization

### 3 Testing of the UB-tree range queries

One from the kind of the spatial queries is the *range query*. The algorithm of UB-tree range query is described in [Ba97] and [Ma99]. Range query processing finds all the tuples (objects) lying inside given  $n$ -dimensional query block.

All the regions overlapped by  $n$ -dimensional block are retrieved and searched during the processing of range query. The goal of our tests is to show that PZ-regions (created using PZ-address) part the  $n$ -dimensional space better than by using of Z-address. The goal is to show the query block overlaps less regions by usage of the PZ-address than by usage of the Z-address. If the query block overlaps less Z-regions, the less B-tree pages are retrieved and also less disk accesses are done and less CPU time is consumed.

We measure the rate of number of regions overlapped by  $n$ -dimensional block by usage of tested address (for example PZ-address) to number of regions overlapped by usage of Z-address. We will note the value as *eff*:

$$eff = \left( 1.0 - \frac{numregsTestedA}{numregsZA} \right) \cdot 100.0 \quad [\%]$$

The *eff* value for  $m$  query blocks is then calculated as:

$$eff_m = \left( 1.0 - \frac{\sum_{i=1}^m numregsTestedA_i}{\sum_{i=1}^m numregsZA_i} \right) \cdot 100.0 \quad [\%]$$

where

*numregsTestedA* is the number of regions overlapped by query block by usage of tested address (for example PZ-address)

*numregsZA* is the number of regions overlapped by query block by usage of Z-address

*numregsTestedA<sub>i</sub>* is *numregsTestedA* value for query block  $i$

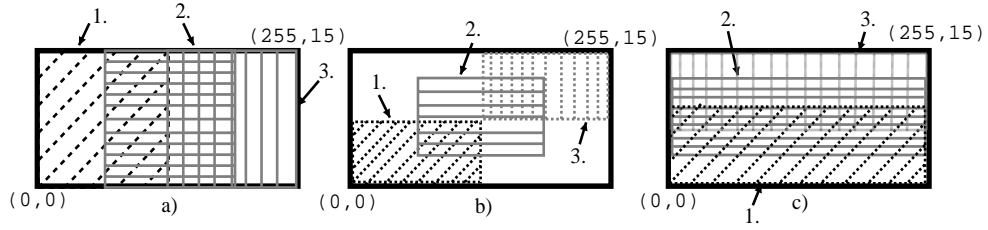
*numregsZA<sub>i</sub>* is *numregsZA* value for query block  $i$

Positive *eff* value means that number of accessed regions by usage of tested address is less than numbers of accessed regions by usage of Z-address.

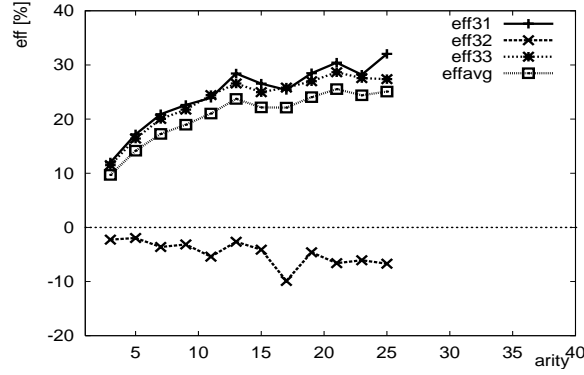
We will execute three tests which consist of three subtests (the calculation of  $eff_3^1$ ,  $eff_3^2$  and  $eff_3^3$  values) and compare PZ-address and Z-address. The first subtest computes  $eff_3^1$  value for three  $n$ -dimensional blocks (see Figure 12a). The second subtest computes  $eff_3^2$  value for three  $n$ -dimensional cubes (see Figure 12b). The third subtest computes  $eff_3^3$  value for three  $n$ -dimensional blocks (see Figure 12c). Thus, each of the test consists of three subtests, the nine tests were executed at the whole. The *eff<sub>avg</sub>* value was the average for the nine tests.

*Test 1:*

We see dependency of *eff* at the arity of the UB-tree in Figure 13. The 16000 tuples were inserted into the 4-dimensional space 16x64x16x64. We see the usage of PZ-address gives better results by computation  $eff_3^1$  value, the usage of PZ-address gives the worse results by computation  $eff_3^2$  value. In spite of this – the *eff<sub>avg</sub>* value is bigger than zero so



**Fig. 12.** The example of testing query blocks for 2-dimensional space 256x16. The testing query blocks for calculation of a)  $eff_3^1$  b)  $eff_3^2$  and c)  $eff_3^3$ .



**Fig. 13.** The results of the test 1.

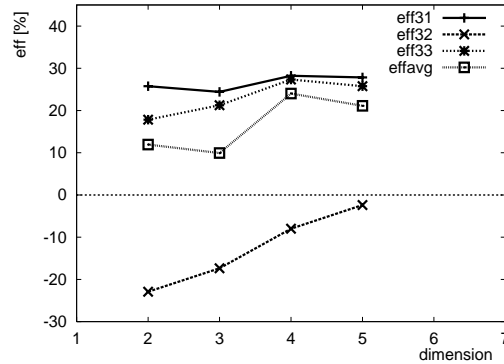
query block overlaps less number of regions by usage of PZ-address than by usage of Z-address. Thus, less B-tree pages are retrieved. We see  $eff_{avg}$  value grows with growing arity.

*Test 2:*

We see dependency of  $eff$  at the dimension  $n$  of space in Figure 14. The number of inserted tuples grows with the dimension  $n$ . The tuples were inserted into spaces with  $n = 2$  (2D space 16x64),  $n = 3$  (16x64x16),  $n = 4$  (16x64x16x64) and  $n = 5$  (16x64x16x64x64). We see the PZ-address gives a better results with growing dimension.

## 4 Conclusions

In this paper we have presented some properties of space filling curves according to the usage with UB-tree. The original design of UB-tree takes into account only the possibility of Z-curve. We have shown that



**Fig. 14.** The results of the test 2.

there exist several aspects giving a reason to propose another alternative curves. This reason is especially based on maximization of the range queries efficiency.

From this point of view we have designed such an alternative curve, i.e. PZ-curve, which tries to take advantage of some space knowledge. PZ-curve is domain dependent, i.e. is suitable for indexing specific vector spaces with differently ranged domains. Example of data modelled within this spaces could be the XML data. Modelling and indexing XML data we closely discuss in [KPS02a,KPS02b].

## References

- [Ba97] Bayer R.: *The Universal B-Tree for multidimensional indexing: General Concepts*. In: Proc. Of World-Wide Computing and its Applications 97 (WWCA 97 ). Tsukuba, Japan, 1997.
- [Cha69] Chandrasekharan, K.: *Introduction to Analytic Number Theory*. Springer-Verlag New York, 1969. ISBN 0387041419.
- [Ka83] Karacuba A.A.: *Introduction to Analytic Number Theory*. Nauka 1983. in russian.
- [KPS02a] Krátký M., Pokorný J., Snášel V.: *Indexing XML data with UB-trees*. Accepted at ADBIS 2002, Bratislava, Slovakia
- [KPS02b] Krátký M., Pokorný J., Skopal T., Snášel V.: *Geometric framework for indexing and querying XML documents*. Accepted at EurAsia ICT 2002, Tehran, Iran
- [Ma99] Markl, V.: *Mistral: Processing Relational Queries using a Multidimensional Access Technique*, <http://mistral.in.tum.de/results/publications/Mar99.pdf>, 1999