A Hilbert Curve-Based Distributed Index for Window Queries in Wireless Data Broadcast Systems

JUN-HONG SHEN, YE-IN CHANG
Department of Computer Science and Engineering
National Sun Yat-Sen University
TAIWAN, R.O.C.
shenjh@cse.nsysu.edu.tw, changyi@cse.nsysu.edu.tw

Abstract

Location-dependent spatial query in the wireless environment is that mobile users query the spatial objects dependent on their current location. The window query is one of the essential spatial queries, which finds spatial objects located within a given window. In this paper, we propose a Hilbert curve-based distributed index for window queries in the wireless data broadcast systems. Our proposed algorithm allocates spatial objects in the Hilbert-curve order to preserve the spatial locality. Moreover, to quickly answer window queries, our proposed algorithm utilizes the neighbor-link index, which has knowledge about neighbor objects, to return the answered objects. From our experimental study, we have shown that our proposed algorithm outperforms the distributed spatial index.

Keywords: Location-dependent spatial query, power constraint, space-filling curve, spatial index, wireless data broadcast.

1 Introduction

Recently, location-dependent spatial query (LDSQ) on the wireless data broadcast is a new concerned issue in the wireless environment. LDSQ in the wireless environment is that mobile users query the spatial objects dependent on their current location. Examples of LDSQs include querying local traffic reports and the nearest restaurants with respect to user's current location [7]. Because of its high scalability, wireless data broadcast is an efficient way to disseminate data to a large number of mobile users. Therefore, wireless data broadcast is particularly suitable for providing spatial objects for a tremendous number of mobile users. Since mobile users may move (mobility), many existing techniques for processing spatial objects and queries in the tradition spatial databases may not fit with the wireless environment. Moreover, because of the power constraint of mobile devices, an important challenge is to provide efficient indexing and searching mechanisms for energy efficient querying of LDSQs [3].

LDSQs include window queries, nearest-neighbor (NN) queries, and *k*-nearest-neighbor (*k*NN) queries. Window queries find data items that are located within a given window, which is a rectangle in a 2-dimensional space [8]. NN queries return only one data item in the spatial space closest to a given query point. *k*NN queries return *k* data items in the spatial space closest to a given query point [8]. Among them, the window query, one of the essential spatial queries, is very useful for spatial selection. In this paper, we focus on the window query.

For a file being broadcast on a channel, the following two parameters are of concern [6]: (1) Access time: The average time elapsed from the moment a client wants a record identified by a primary key, to the point when the required record is downloaded by the client. (2) Tuning time: The amount of time spent by a client listening to the channel. This will determine the power consumed by the client to retrieve the required data. Since battery power is a scarce resource in mobile devices, it is crucial for saving energy consumption of the devices during the query process. Therefore, in this paper, the main concern is to reduce the tuning time.

In the literature, there has been much work providing index structures to support the efficient access on LDSQs on the wireless data broadcast. In [9], Zheng *et al*. proposed the grid-partition index to support NN queries. The studies in [3, 7] are specified for *k*NN queries. The Hilbert curve index [8] provides an index structure to support window queries and *k*NN queries. In [5], Lee and Zheng proposed the distributed spatial index (*DSI*) to improve the performance of the Hilbert curve index.

Among the above work, *DSI* [5] can provide a good performance for window queries on the wireless data broadcast. However, this work does not utilize the property that the answered objects for window queries may be the neighbor of each other to further reduce the tuning time. Therefore, in this paper, we propose a Hilbert curve-based distributed index (*HCDI*) to support window queries by using the mentioned property. In our experimental results, we have shown that our proposed algorithm can perform better than the distributed spatial index.

The rest of this paper is organized as follows. In Section 2, we give a brief description of the Hilbert curve. In Section 3, we present our proposed Hilbert curve-based distributed index. In Section 4, we study the performance of the proposed algorithm, and make a comparison with the distributed spatial index by simulation. Finally, a conclusion is presented in Section 5.

2 Background

In the wireless environments, a broadcast cycle consists of a collection of data items, which are cyclically broadcast on the wireless channel. Mobile clients listen to the wireless channel to retrieve the data item of interest. In this section, we briefly describe the Hilbert curve, one kind of space-filling curves.

A space-filling curve is a continuous path which passes through every point in a multi-dimensional space once to form a one-one correspondence between the coordinates of the points and the one-dimensional sequence numbers of the points on the curve [4]. Some examples of space-filling curves are the Peano curve, the RBG curve and the Hilbert curve. Among them, the Hilbert curve can preserve the spatial locality of points. The spatial locality means that points that are close to each other in a multi-dimensional space are remained to close to each other in a one-dimensional space. Figure 1-(a) shows the Hilbert curve of order 1. The curve can keep growing recursively by following the same rotation and reflection pattern at each point of the basic curve. Figures 1-(b) and 1-(c) show the Hilbert curves of orders 2 and 3, respectively. In this paper, to preserve the spatial locality, we allocate spatial objects to the broadcast channel in the ascending order of the Hilbert curve.

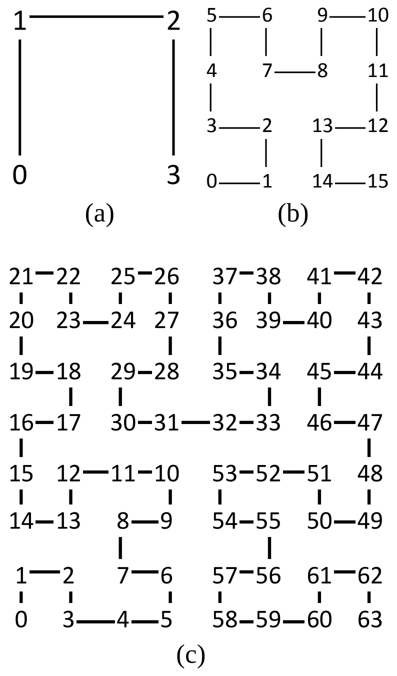

Figure 1. The Hilber curve: (a) order 1; (b) order 2; (c) order 3.

3 The Hilbert Curve-Based Distributed Index

To provide an efficient way to process window queries in the wireless broadcast environments, we propose a Hilbert curve-based distributed index (*HCDI*). In this section, we first state the assumptions of our proposed algorithm and then present our proposed algorithm.

3.1 Assumptions

This paper focuses on the wireless environment. Some assumptions should be restricted in order to make our work feasible [1]. These assumptions include:

1. Spatial objects are represented as a point in a two-dimensional space.
2. Spatial objects appear once in a broadcast cycle, *i.e.*, the uniform broadcast.
3. A bucket is a logical transmission unit on a broadcast channel. Index tuples can be put into an index bucket and a spatial object can be put into a data bucket. The size of a data bucket is far larger than that of an index bucket, and the size of a data bucket is a multiple of that of an index bucket.
4. Clients make no use of their upstream communications capability.
5. When a client switches to the public channel, it can retrieve buckets immediately.
6. The server broadcasts buckets over a reliably single channel.

3.2 Neighbor Links

For a window query of spatial objects, the answered objects may be the neighbors of each other. To provide an efficient way to support the window query, we propose *neighbor links* to guide clients to receive related objects. The policy for adding neighbor links to the base unit is that the base unit has neighbor links pointing to the neighbor units of the northern, southern, eastern, western, northeastern, southeastern, northwestern and southwestern directions that have the Hilbert-curve value greater than its one. The efficient way to find these neighbor units of the current base unit in the Hilbert curve can be found in [2]. In [2], Chen and Chang presented a method to find the sequence numbers of the neighboring blocks next to the current base unit based on its bit shuffling property in the Peano curve, and the transformation rules between the Peano curve and the Hilbert curve. That is, the sequence numbers of the neighboring blocks next to the current base unit can be easily found in the Peano curve, and then transformed to the ones in the Hilbert curve.

3.3 The Proposed Algorithm

The proposed algorithm for efficiently processing window queries is proceeded as follows.

1. Allocate spatial objects in the ascending order of the Hilbert curve of order *s*.
2. Allocate one index bucket before each data bucket. Each index bucket contains the neighbor-link index and the local index. Let the objects covered by the same Hilbert-curve value of order (*s*-1) as a group. The allocation of the index bucket is processed as follows.
   a. Add neighbor links (the neighbor-link index) to the corresponding index buckets from base units (blocks) of order (*s*-1) to those of order 1.
   b. Check if the index tuples of the neighbor-link index in the index buckets have the same pointer (offset).If it is true, remove the index tuples with the short range.
   c. Add the local index, which has information about the objects in the same group, to the corresponding index buckets

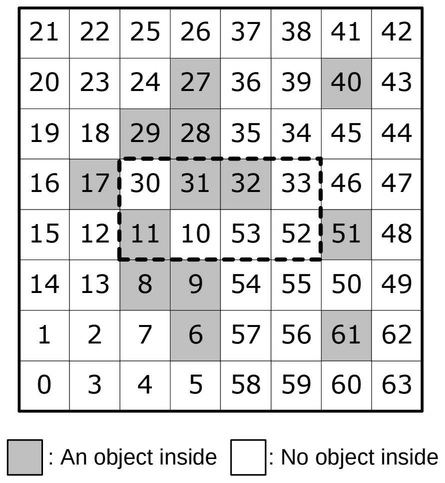

Figure 2. A Hilbert curve of order 3

Now, we use an example to illustrate our proposed algorithm. Figure 2 shows an example of the Hilbert curve of order 3, where the gray block contains a spatial object inside and the white one contains no object. In Step 1, we allocate spatial objects to the one-dimensional space in the order of the Hilbert curve of order s (= 3).

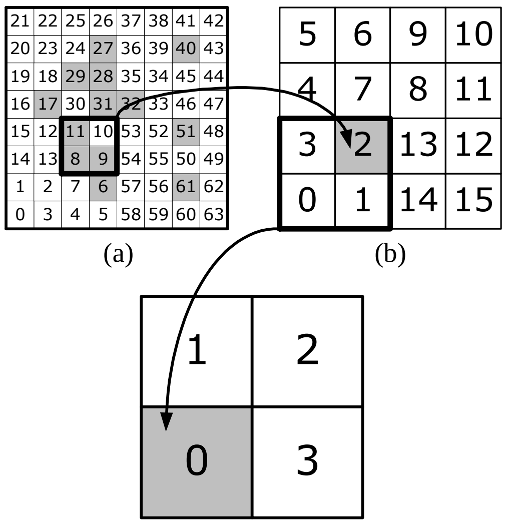

Figure 3. Block levels: (a) order 3; (b) order 2; (c) order 1.

Next, in Step 2, we allocate one index bucket before each data bucket. The objects covered by the same Hilbert-curve value of order (s-1) (= 2) are considered as a group. Block j of order (i+1), $0 < j < 2^{(i+1)} \times 2^{(i+1)}$, is covered in block ($\lfloor j/4 \rfloor$) of order i. Take objects 8, 9 and 11 shown in Figure 3-(a) for example. Their Hilbert-curve values are covered by block 2 (= $\lfloor 8/4 \rfloor = \lfloor 9/4 \rfloor = \lfloor 11/4 \rfloor$) of order 2 shown in Figure 3-(b). Therefore, these objects are in the same group.

Then, in Step 2-(a), to provide an efficient way to support the window query, neighbor links (the neighbor-link index) are allocated to the corresponding index buckets for each object in the group from base blocks of order (s-1) (= 2) to those of order 1. Consider the group containing objects 8, 9 and 11 shown in Figure 3-(a) for example. The neighbor links of their corresponding block 2 of order 2 shown in Figure 3-(b) are pointing to blocks 4, 7, and 8 of order 2, which contain objects inside. Moreover, the neighbor links of their corresponding block 0 (= $\lfloor 2/4 \rfloor$) of order 1 shown in Figure 3-(c) are pointing to blocks 1, 2 and 3 of order 1, which contain objects inside. The neighbor links of the corresponding blocks of order 1 and order 2 for the objects shown in Figure 2 are listed in Table 1. Note that in Table 1, the range in parentheses following the neighboring block indicates the Hilbert-curve values of order 3 of the objects covered by this neighboring block. In Figure 3, we can observe that blocks 3, 5, 9, 11, 13 and 14 of order 2 contain no object inside; therefore, Table 1 has no information about these blocks.

Table 1. Neighbor links

Order	Start Block	Neighboring Blocks (Range)
1	0	1 ([17,31]), 2 ([32,40]), 3 ([51,61])
	1	2 ([32,40]), 3 ([51,61])
	2	3 ([51,61])
	3	-
2	1	2 ([8,11])
	2	4 ([17,17]), 7 ([28,31]), 8 ([32,32])
	4	6 ([27,27]), 7 ([28,31])
	6	7 ([28,31]), 8 ([32,32])
	7	8 ([32,32])
	8	10 ([40,40]), 12 ([51,51])
	10	-
	12	15 ([61,61])
	15	-

* "-": no neighbor link

The index tuples of the index buckets are shown in Figure 4. Take index bucket $I1$ for object 8 shown in Figure 4 for example. From Figures 3-(a) and 3-(b), we get that object 8 is covered in block 2 of order 2. Moreover, in Table 1, we get the neighboring blocks of block 2 of order 2 and their corresponding ranges, 4 ([17, 17]), 7 ([28, 31]), and 8 ([32, 32]). Therefore, in index bucket $I1$, the neighbor-link index has these three index tuples pointing to the index buckets for the first objects in these ranges, *i.e.*, $\langle$[17, 17], $I4\rangle$, $\langle$[28, 31], $I6\rangle$, and $\langle$[32, 32], $I9\rangle$ shown in Figure 4. (Note that since the size of a data bucket is a multiple of that of an index bucket, it is easy to convert the pointer of an index tuple to an offset to the corresponding bucket.) Furthermore, from Figures 3-(a), 3-(b) and 3-(c), we get that object 8 is contained in block 0 of order 1. In Table 1, we get the neighboring blocks of block 0 of order 1 and their corresponding ranges, 1 ([17, 31]), 2 ([32, 40]), 3 ([51, 61]). As a result, in index bucket $I1$, the neighbor-link index has these three index tuples, $\langle$[17, 31], $I4\rangle$, $\langle$[32, 40], $I9\rangle$, and $\langle$ [51, 61], $I11\rangle$ shown in Figure 4.

In Step 2-(b), to save the space in the index bucket, we check if the index tuples of the neighbor-link index in the index buckets have the same pointer (offset). For the index tuples with the same pointer, since the index tuple with the wide range can contain more information about the neighboring objects than that with the short one, the latter one is removed from the index bucket. Take index tuples ⟨[17, 17], *I*4⟩ and ⟨[17, 31], *I*4⟩ in index bucket *I*1 for example. Range [17, 31] of index tuple ⟨[17, 31], *I*4⟩ is wider than range [17, 17] of index tuple ⟨[17, 17], *I*4⟩. Therefore, index tuple ⟨[17, 17], *I*4⟩ is removed from index bucket *I*1. The removed index tuples are the ones with a strikethrough line shown in Figure 4.

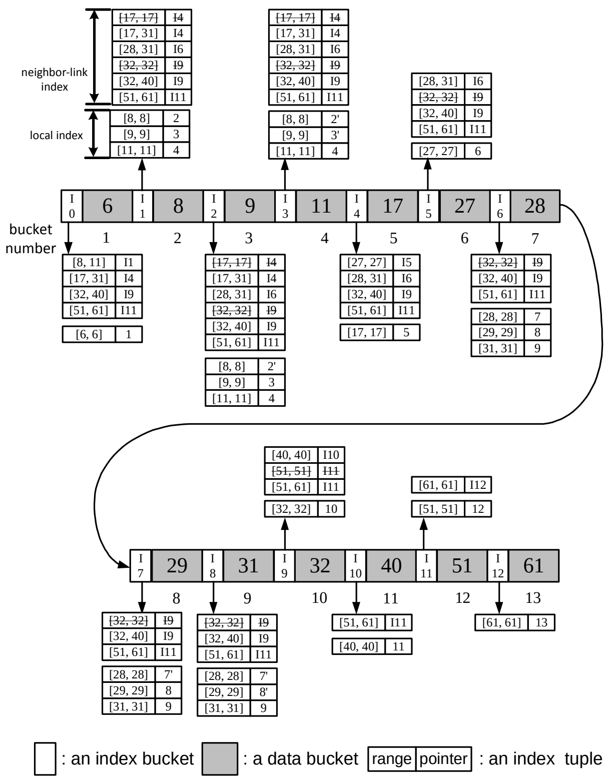

Figure 4. The allocation of the Hilbert curve-based distributed index

In Step 2-(c), we add the local index, which has information about the objects in the same group, to the corresponding index buckets. Take index bucket *I*1 for object 8 shown in Figure 4 for example. Since objects 8, 9, and 11 are in the same group, the local index in index bucket *I*1 has three index tuples pointing to the data buckets containing these objects, *i.e.*, ⟨[8, 8], 2⟩, ⟨[9, 9], 3⟩, and ⟨[11, 11], 4⟩ shown in Figure 4. Note that in Figure 4, when index bucket *I*2 is broadcast, bucket number 2 containing object 8 has been broadcast; therefore, index tuple ⟨[8, 8], 2'⟩ in index bucket *I*2 points to bucket number 2 in the next cycle. The index tuples of the local index in index buckets *I*3, *I*7 and *I*8 are in the same manner.

Each index bucket has an index tuple pointing to the beginning of the next cycle to retrieve objects that have been broadcast. Each data bucket has an offset to direct clients to the nearest index bucket to start receiving the related buckets.

3.4 Window Queries

To process a window query with our proposed algorithm, all the segments along the Hilbert curve that are intersected with a given query window should be found [5]. The access protocol for window queries is proceeded as follows.

1. Tune in to the broadcast channel to receive the current bucket to get the offsets to the nearest index bucket and the beginning of the next cycle, and then go into the doze mode.
2. Tune in to receive the bucket, a data bucket or an index bucket. If an index bucket is received, index tuples should be examined.
   a. Check index tuples of the neighbor-link index, which have the shortest range covering one of the intersected segments, to get the offsets to the related index buckets.
   b. Check index tuples of the local index to get the offsets to the related data buckets.
3. Determine the nearest offset of the related bucket to tune in, and then go into the doze mode to save power consumption.
4. Repeat Step 2 to Step 3 until all the intersected segments are checked.

Take the query window (the dash-line box) in Figure 2 for example. This query window can be divided into three segments covered with the Hilbert curve, [10, 11], [30, 33] and [52, 53]. Assume that the client first tunes in to the channel at the beginning of the index bucket *I*0 in Figure 4. After checking this bucket, the client gets the offsets to index buckets *I*1, *I*4, *I*9 and *I*11, which have information about the intersected segments. Next, the client tunes in at the nearest-related index bucket *I*1. By examining the neighbor-link index in this bucket, the client finds tuple ⟨[28, 31], *I*6⟩ that has the shortest range covering segment [30, 33]. Since the range covering segment [30, 33] of index bucket *I*4 is longer than that of index bucket *I*6, the offset to index bucket *I*4 is replaced by that to index bucket *I*6. At the same time, from the local index of index bucket *I*1, the client gets the offset to object 11, which is in [10, 11].

After receiving object 11, the client reaches index bucket *I*6. From the local index of this bucket, the client gets the offset to receive object 31, which is in [30, 33]. After that, the client reaches object 32 through index bucket *I*9. The client finally examines index bucket *I*11 and knows that there is no object in segment [52, 53]. Up to this point, all the intersected segments have been checked and the query processing is terminated. The answered objects for this query are objects 11, 31, and 32.

4 Performance

In this section, we study the performance of the proposed algorithm. We compare our proposed algorithm with the distributed spatial index, *DSI* [5]. *DSI* divides the spatial objects, which are allocated in the Hilbert-curve order, into frames. Each frame has an index table that maintains information about spatial objects which are exponentially away from the current frame.

4.1 The System Model

In our simulation model, two integer numbers of *IntSize* (= 1) byte are used to represent two coordinates in a two-dimensional space, so that an integer number of 2 bytes is used to represent a Hilbert-curve value. Each spatial object occupies *DataSize* (= 1024) bytes. The search region of a window query is controlled by *WinSideRatio*, the ratio of the side length of a window query to that of the search space. Given *WinSideRatio* = 0.1 and the side length of the search space is equal to 2^8 (= 256), the search region of a window query is a 26 ($\lceil 256 \times 0.1 \rceil$) $\times$ 26 square. In our simulation, 10,000 points are uniformly generated in a square Euclidean space [5], and 10,000 queries are randomly issued. Therefore, our experimental results are the average of 10,000 queries. The average tuning time is measured in terms of bytes.

4.2 Experimental Results

For the distributed spatial index (*DSI*), we set the number of objects in a frame and the exponential base for index tuples to 8 and 2, respectively. In this performance evaluation, we vary *WinSideRatio* from 0.1 to 0.3. Figure 5 shows the average tuning time. In this figure, as the value of *WinSideRatio* increases, the average tuning time of both algorithms increases. As the value of *WinSideRatio* increases, the search region increases.

It means that the objects in the search region to be examined increase, resulting in the increase of the average tuning time. In Figure 5, we can observe that the average tuning time of our proposed algorithm is shorter than that of *DSI*. Our proposed algorithm has an average improvement of 36% on the average tuning time over *DSI*. This is because our proposed *HCDI* utilizes the neighbor-link index to reduce the number of the tune-in buckets, resulting in the reduction of the tuning time.

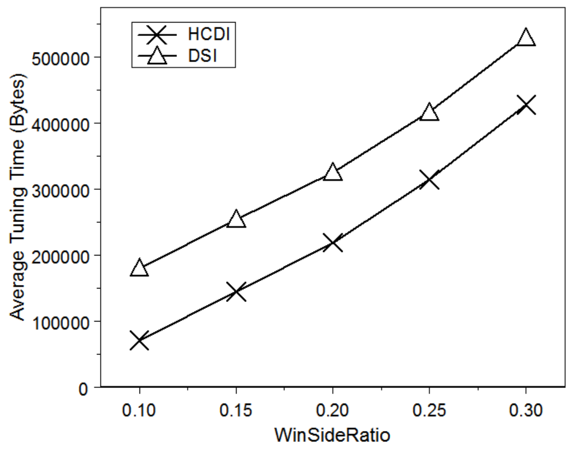

Figure 5. The average tuning time

5 Conclusion

In this paper, we have proposed a Hilbert curve-based distributed index with neighbor links for window queries over the single wireless broadcast channel. For a window query of spatial objects, the answered objects may be the neighbors of each other. Our proposed algorithm utilizes neighbor links, which point to the neighbor objects of the current object, to efficiently process the window query. From our simulation results, we have shown that the proposed algorithm needs the shorter average tuning time than the distributed spatial index.

Acknowledgement

This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-95-2221-E-110-079-MY2 and by “Aim for Top University Plan” project of NSYSU and Ministry of Education, Taiwan.

References

[1] Y. I. Chang and C. N. Yang, *A Complementary Approach to Data Broadcasting in Mobile Information Systems*, *Data and Knowledge Eng*., Vol. 40, No. 2, Feb., 2002, pp. 181–194.
[2] H. L. Chen and Y. I. Chang, *Neighbor Finding Based on Space Filling Curves*, *Information Systems*, Vol. 30, No. 3, May, 2005, pp. 205–226.
[3] B. Gedik, A. Singh, and L. Liu, *Energy Efficient Exact kNN Search in Wireless Broadcast Environments*, *Proc. of the 12th ACM Int. Workshop on Geographic Info. Systems*, 2004, pp. 137–146.
[4] J. K. Lawder and P. J. H. King, *Querying Multi-Dimensional Data Indexed Using the Hilbert Space-Filling Curve*, *ACM SIGMOD Record*, Vol. 30, No. 1, March, 2001, pp.19–24.
[5] W. C. Lee and B. Zheng, *DSI: A Fully Distributed Spatial Index for Location-Based Wireless Broadcast Services*, *Proc. of the 25th IEEE Int. Conf. on Distributed Computing Systems*, 2005, pp. 349–358.
[6] J. H. Shen and Y. I. Chang, *A Skewed Distributed Indexing for Skewed Access Patterns on the Wireless Broadcast*, *The Journal of Systems and Software*, Vol. 80, No. 5, May, 2007, pp. 711–723.
[7] B. Zheng, W. C. Lee, and D. L. Lee, *Search K Nearest Neighbors on Air*, *Proc. of the 4th Int. Conf. on Mobile Data Management*, 2003, pp. 181–195.
[8] B. Zheng, W. C. Lee, and D. L. Lee, *Spatial Queries in Wireless Broadcast Systems*, *Wireless Networks*, Vol. 10, No. 6, Nov., 2004, pp. 723–736.
[9] B. Zheng, J. Xu, W. C. Lee, and D. L. Lee, *Grid-Partition Index: A Hybrid Approach to Nearest-Neighbor Queries in Wireless Location-Based Services*, *The VLDB Journal*, Vo. 15, No. 1, Jan., 2006, pp. 21–39.