Oriented k-windows: A PCA driven clustering method

D.K. Tasoulis[1], D. Zeimpekis[2], E. Gallopoulos[2], and M.N. Vrahatis[1]

[1] Department of Mathematics, University of Patras, GR-26110 Patras, Greece {dtas,vrahatis}@math.upatras.gr
[2] Computer Engineering & Informatics Dept., University of Patras, GR–26500 Patras, Greece {dsz,stratis}@hpclab.ceid.upatras.gr

Summary. In this paper we present the application of Principal Component Analysis (PCA) on subsets of the dataset to better approximate clusters. We focus on a specific density-based clustering algorithm, k-Windows, that holds particular promise for problems of moderate dimensionality. We show that the resulting algorithm, we call Oriented k-Windows (OkW), is able to steer the clustering procedure by effectively capturing several coexisting clusters of different orientation. OkW combines techniques from computational geometry and numerical linear algebra and appears to be particularly effective when applied on difficult datasets of moderate dimensionality.

1 Introduction

Density based methods are an important category of clustering algorithms [4, 13, 23], especially for data of low attribute dimensionality [6, 17, 26]. In these methods, clusters are formed as regions of high density, in dataset objects, surrounded by regions of low density; proximity and density metrics need to be suitably defined to fully describe algorithms based on these techniques though there are difficulties for problems of high attribute dimensionality; cf. [1, 16]. One recent technique in this category is "Unsupervised k-Windows" (UkW for short) [28], that utilizes hyperrectangles to discover clusters. The algorithm makes use of techniques from computational geometry and encapsulates clusters using linear containers in the shape of d-dimensional hyperrectangles that are aligned with the standard Cartesian axes and are iteratively adjusted with movements and enlargements until a certain termination criterion is satisfied; cf. [24, 28]. An advantage of the algorithm is that it allows a reduction in the number of objects examined at each step, something especially useful when dealing with large collections. Furthermore, with proper tuning, the algorithm can detect clusters of arbitrary shapes. We show here a general approach that appears to improve the effectiveness of UkW. This approach relies on techniques from linear algebra to orient the hyperrectangles

Mark Last et al. (Eds.): Advances in Web Intelligence and Data Mining (SCI) **23**, 319-328 (2006)
www.springerlink.com

into directions that are not necessarily axes-parallel. The technique used in this process is PCA [18], implemented via singular value decomposition (SVD) [14]. The paper is organized as follows. The details of unsupervised k-Windows are described in Section 2. Section 3 describes OkW, while Section 4 presents experimental evidence of its efficiency. Finally, Section 5 contains concluding remarks. We would be assuming that datasets are collections of d-dimensional objects, and can be represented by means of an $n \times d$ object-attribute matrix; unless specified otherwise, "dimension" refers to attribute cardinality, d.

2 Unsupervised k-Windows clustering

UkW aims at capturing all objects that belong to one cluster within a d–dimensional window. Windows are defined to be hyperrectangles (orthogonal ranges) in d dimensions; cf. [28] for details. UkW employs two fundamental procedures: *movement* and *enlargement*. The movement procedure aims at positioning each window as close as possible to the center of a cluster. The enlargement process attempts to enlarge the window so that it includes as many objects from the current cluster as possible. The two steps are illustrated in Figure 1. UkW provides an estimate for the number of clusters that describe

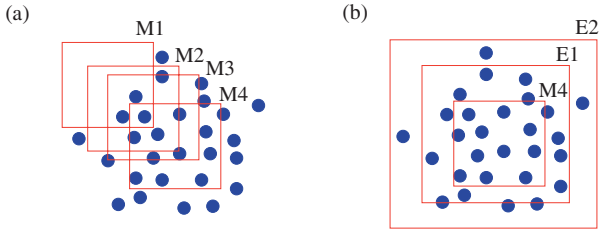

Fig. 1. (a) Sequential movements M2, M3, M4 of initial window M1. (b) Sequential enlargements E1, E2 of window M4.

a dataset. The key idea is to initialize a large number of windows. When the movement and enlargement of all windows terminate, all overlapping windows are considered for merging by considering their intersection. An example of this operation is exhibited in Figure 2.

The computationally intensive part of k-Windows is the determination of the points that lie in a hyperrectangle. This is the well known "orthogonal range search" problem [22]. The high level description of the algorithm is as follows:

algorithm UkW $(a, \theta_e, \theta_m, \theta_c, \theta_v, \theta_s, k)$ {
  execute W=DetermineInitialWindows(k,a)
  for each hyperrectangle w_j in W **do**
    repeat

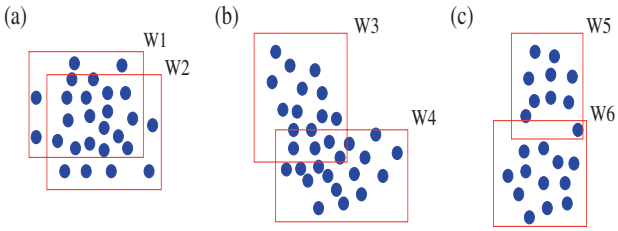

Fig. 2. (a) W_1 and W_2 satisfy the similarity condition and W_1 is deleted. (b) W_3 and W_4 satisfy the merge operation and are considered to belong to the same cluster. (c) W_5 and W_6 have a small overlap and capture two different clusters.

execute movement(θ_v,w_j)
execute enlargement(θ_e,θ_c,θ_v,w_j)
until the center and size of w_j remain unchanged
execute merging(θ_m,θ_s,W)
Output $\{$clusters $c_{l_1}, c_{l_2}, \ldots$ so that: $c_{l_i} = \{i : i \in w_j, \text{label}(w_j) = l_i\}\}$
}
function DetermineInitialWindows(k,a) {
initialize k d-dimensional hyperrectangles $w_{m_1}, \ldots, w_{m_k}$ with edge length a
select k points from the dataset and
center the hyperrectangles at these points
return a set W of the k hyperrectangles
}
function movement(θ_v, hyperrectangle w) {
repeat
find the objects that lie in the hyperrectangle w
calculate the mean m of these objects
set the center of w equal to m
until the Euclidean distance between m and
the previous center of w is less than θ_v
}
function enlargement(θ_e,θ_c,θ_v, hyperrectangle w) {
repeat
foreach dimension i **do**
repeat
enlarge w along current dimension by θ_e%
execute movement(θ_v,w)
until increase in number of objects
along current dimension is less than θ_c%
until increase in number of objects
is less than θ_c% along each dimension
}
function merging(θ_m,θ_s, set of hyperrectangles W) {
for each hyperrectangle w_j in W not marked **do**

mark w_j with label w_j
if $\exists\ w_i \neq w_j$ in W, that overlaps with w_j
compute the number of points n that
lie in the overlap area
if $(\frac{n}{|w_i|}) \geq \theta_s$ and $|w_i| < |w_j|$
disregard w_i
if $0.5 * (\frac{n}{|w_j|} + \frac{n}{|w_i|}) \geq \theta_m$
mark all w_i labeled hyperrectangles in W with label w_j
}

3 *k*-Windows with PCA based steering

Results with UkW in [28] showed that a procedure based on moving and enlarging hyperrectangles together with merging is effective in discovering clusters. All movements, however, were constrained to be parallel to any one of the standard cartesian axes. We next consider the possibility of allowing the hyperrectangles adapt both their orientation and size, as means to more effective cluster discovery. Assume, for example, that k d-dimensional hyperrectangles of the UkW algorithm have been initialized, as in function `DetermineInitialWindows` above. After executing the movement function, the center and contents of each hyperrectangle might change. It would then be appropriate to re-orient the hyperrectangle from its original position (axes-parallel in the first step, maybe different in later ones) so as to take into account the local variabilities present in the data. This is accomplished by means of PCA on a translation of the subset of points (objects) included in the specific hyperrectangle.

Let, for example, $A^{(i)} \in \mathbb{R}^{n_i \times d}$ be the object-attribute matrix corresponding to objects in a subset $\mathcal{P}$ of the original dataset. Matrix $B^{(i)} := A^{(i)} - \frac{1}{n_i} e e^\top A^{(i)}$, where $e \in \mathbb{R}^{n_i}$ is a vector of all 1's, contains the points $B^{(i)}$, that is the points of $\mathcal{P}$ written in terms of the cartesian axes centered at the centroid $g = \frac{1}{n_i} e^\top A^{(i)}$ of $\mathcal{P}$. Let the SVD of $B^{(i)}$ be $B^{(i)} = U^{(i)} \Sigma^{(i)} (V^{(i)})^\top$ and $\{u_j, \sigma_j, v_j\}, j = 1, ..., d$ the nontrivial singular triplets of $B^{(i)}$, in decreasing order, i.e. $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d \geq 0$. For simplicity we omit the index i. One way to build a hyperrectangle bounding the elements of $\mathcal{P}$ is to use the fact that the columns of V_i are the principal directions of the elements of $\mathcal{P}$. The hyperellipsoid with semiaxes σ_j encloses all of $\mathcal{P}$. It can, however, cover a much larger volume than the volume enclosed by the span of $\mathcal{P}$ and even the hyperellipsoid of minimal volume containing $\mathcal{P}$. On the other hand, computing either of these minimal bodies and keeping track of geometrical operations with them is difficult; e.g. see [20]. Instead, since OkW was based on hyperrectangles, we prefer to preserve that choice here as well. We search, then for hyperrectangles that bound the points of $\mathcal{P}$, but no longer requiring

that they are axes-parallel. A simple, yet effective choice, is to exploit the equality $B^{(i)}V^{(i)} = U^{(i)}\Sigma^{(i)}$ which shows that the projections of the n_i rows of $B^{(i)}$ on any of the principal axes, say v_j, is given by the elements of column j of $U^{(i)}$ multiplied by σ_j. Therefore, we can construct a bounding hyperrectangle for $\mathcal{P}$ by taking g as center and the values $2\sigma_j\|u_j\|_\infty$ as edge lengths. An alternative (sometimes better) approximation would be to project all points on j^{th} principal direction, and use as corresponding edge length parallel to axis j, the largest distance between the projected points. Note that adapting the edge lengths in this manner makes the algorithm less sensitive to the size of the original hyperrectangle edge length.

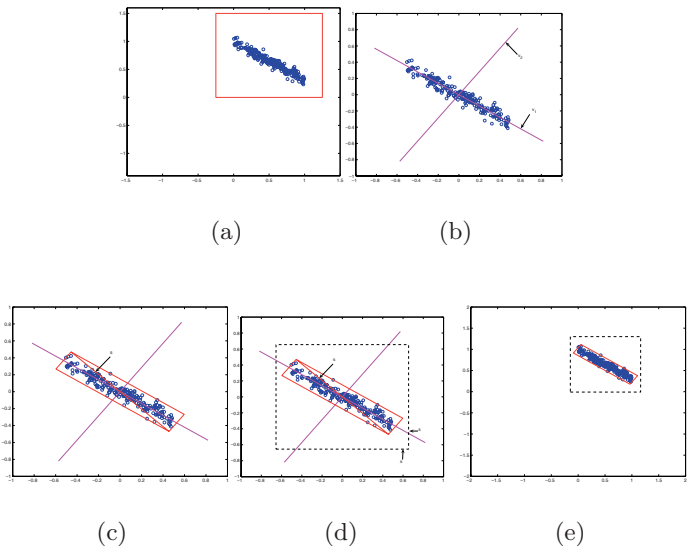

Fig. 3. The initial data and window (a) are shifted and the principal directions are computed (b). We compute the minimum bounding principal directions oriented hyperrectangle (c) and the corresponding hypercube (cf. next subsection 3.1) (d). Finally, we shift the data back (e).

An example of the above process is illustrated in Figure 3. Initially the data points are centered and the principal components of the corresponding matrix are computed. Then, we compute the minimum principal directions oriented hyperrectangle with edge lengths $2\sigma_j\|u_j\|_\infty, j = 1, ..., d$. Finally, we compute a suitable hypercube (we analyze this issue in the next subsection, 3.1) and shift the data back. Then, the enlargement process takes place along each dimension of the rotated window. We call the proposed algorithm *Oriented k-Windows (OkW)* and summarize it below.

algorithm OkW$(u, \theta_e, \theta_m, \theta_c, \theta_v, k)$ {

execute W=DetermineInitialWindows(k,u)
for each d dimensional hyperrectangle w_j in W **do**
execute movement(θ_v,w_j)
execute SVD-enlargement(θ_e,θ_c,θ_v,w_j)
execute merge(θ_m,θ_s,W)
Output $\{$clusters $c_{l1}, c_{l2}, \ldots$ so that: $c_{l_i} = \{i : i \in w_j, \text{label}(w_j) = l_i\}\}$

function SVD-enlargement(θ_e,θ_c,θ_v, hyperrectangle w) $\{$
repeat
compute the principal components V of the points in w
compute a hyperrectangle with edge length $2\sigma_1\|u_1\|_\infty, \ldots, 2\|u_d\|_\infty$
foreach dimension **do**
repeat
enlarge w along current dimension by $\theta_e\%$
execute movement(θ_v,w)
until increase in number of objects
along current dimension is less than $\theta_c\%$
until increase in number of objects
is less than $\theta_c\%$ along every dimension
$\}$

3.1 Computational issues

A challenge in the proposed approach is the efficient handling of non axes-parallel hyperrectangles. Unlike the axes-parallel case, where membership of any d-dimensional point can be tested via the satisfaction of $2d$ inequalities (one per interval range, one per dimension), it appears that oriented hyperrectangles necessitate storing and operating on 2^d vertices. To resolve this issue, we follow a trick from computational geometry designed to handle queries for non-axes parallel line segments, which amounts to embedding each segment in an axes-parallel rectangle [11]. Similarly, in OkW, we embed each hyperrectangle in an axes-parallel "bounding hypercube" that has the same center, say c, as the hyperrectangle, and edge length, s, equal to the maximal diagonal of the hyperrectangle. This edge length guarantees enclosure of the entire oriented hyperrectangle into the axes-parallel hypercube. Thus, the bounding hypercube can be encoded using a single pair (c, s). To perform range search and answer if any given point, x, in the bounding hypercube also lies in the oriented hyperrectangle, it is then enough to check the orthogonal projection of the point onto the d principal directions. If $l_j, j = 1, ..., d$ denote the edge lengths of the hyperrectangle, then x lies in the hyperrectangle if it satisfies all inequalities $|x^\top v_j| \leq \frac{l_j}{2}$ for $j = 1, ..., d$. Therefore, we only need to store $O(d)$ elements, namely $(c, s, l_1, ..., l_d)$.

OkW necessitates the application of SVD on every centered data subset in the course of each enlargement step. It is thus critical to select a fast

SVD routine, such as LAPACK [3], for boxes of moderate size without any particular structure, or from [7], for sparse data. In any case, it is important to exploit the structure of the dataset to reduce the cost of these steps. The call to an SVD routine for problems without specific structure carries a cost of $O(n_i d^2)$, which can be non-negligible. In the applications of greater interest for OkW, however, the attribute dimensionality d is typically much smaller than n and even n_i, so that we can consider the asymptotic cost of the SVD to be linear in the number of objects. Furthermore, when d is much smaller than n_i, it is preferable to first compute the QR decomposition of $B^{(i)}$ and then the SVD of the resulting upper triangular factor [14].

4 Experimental Results

UkW and OkW were implemented in C++ under Linux using the gcc 3.4.2 compiler. The SVD was computed using a C++ wrapper[3] around SVDPACK[4]. All experiments were performed on an AMD Athlon(tm) 64 Processor 3200+, with $1GB$ of RAM. We outline here some results and defer to [27] for more detailed experiments.

Two artificial and two real datasets were used. The former are considered to be difficult, even for density-based algorithms. The first, $Dset_1$, consists of four clusters (three convex and one non-convex) for a total of 2,761 objects, while the second one, $Dset_2$, consists of 299 objects, organized in two clusters forming concentric circles. The values of the parameters $\{\theta_e, \theta_m, \theta_c, \theta_v\}$ were set to $\{0.8, 0.1, 0.2, 0.02\}$ for both UkW and OkW. We used 64 initial windows for $Dset_1$ and 32 for $Dset_2$. Results are depicted in Figure 4. For both datasets, OkW was able to discover the correct clustering. For $Dset_1$, UkW was able to identify correctly only the three convex clusters, while it split the non-convex one into three clusters. Finally, for $Dset_2$, UkW split the outer circle in four clusters. UkW was able to identify correctly the outer cluster only when the number of initial windows was increased to over 200.

The real datasets were $Dset_{UCI}$ and $Dset_{DARPA}$. The former is the Iris dataset from the UCI machine learning repository [8], that consisted of 150 objects of 4 attributes each, organized in three classes, namely Setosa, Versicolour, Virginica. The final real dataset was from the KDD 1999 intrusion detection contest [19]. It contained 100,000 objects of 37 (numerical) attributes, $77,888$ objects corresponded to "normal connections" and $22,112$ corresponding to "Denial of Service" (DoS) attacks. The confusion matrices for $Dset_{UCI}$ were obtained from each one of UkW and OkW for 32 initial windows with the same parameter setting, are depicted in Table 1. Both algorithms successfully identified the number of clusters. However, the number of points that have different class and cluster labels, were 8 for UkW and 7 for OkW. Finally,

[3] http://www.cs.utexas.edu/users/suvrit/work/progs/ssvd.html.
[4] http://www.netlib.org/svdpack/

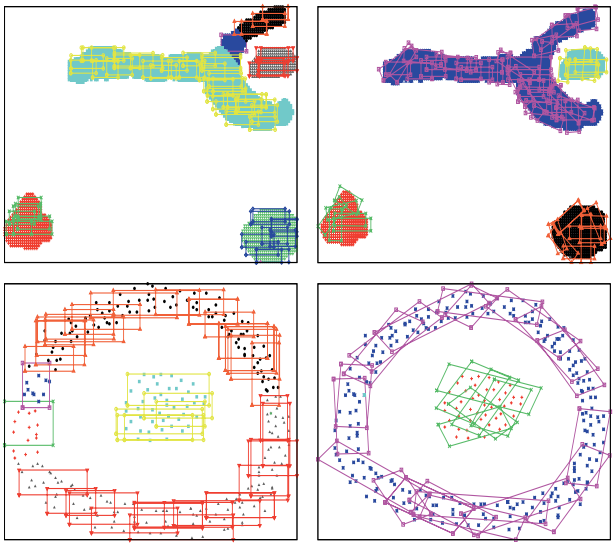

Fig. 4. $Dsets_{1,2}$ with the result of UkW (left) and OkW (right).

the application of UkW on $Dset_{\text{DARPA}}$ with 32 initial windows, estimated the presence of 6 clusters. One of them contained 22,087 DoS objects; the remaining clusters contained objects corresponding to normal connections, with the exception of one cluster that also contained 37 DoS objects. Therefore, UkW's performance was moderately adequate. The application of OkW on the above dataset estimated the presence of five clusters. All, except one cluster contained exclusively either normal or DoS objects. Only 2 DoS objects were assigned in a cluster of 747 normal objects. Therefore, OkW resulted in considerably fewer misclassifications, and thus greater accuracy, than UkW.

Table 1. Confusion data for $Dset_{\text{UCI}}$: The elements in each pair corresponds to the confusion matrices for UkW and OkW.

	Iris class		
Cluster id	Setosa	Versicolour	Virginica
1	50, 50,	0, 0	0, 0
2	0, 0	46, 48	4, 5
3	0, 0	4, 2	46, 45

5 Related work and discussion

The use of PCA in OkW offers a paradigm of synergy between computational linear algebra and geometry that appears to significantly enhance the ability

of k-Windows to capture clusters having difficult shapes. OkW follows the recent trend in clustering algorithms to capitalize on any local structure of the data to produce better clustering of datasets of moderate dimensionality; see e.g. [9, 10, 12]. Preliminary results appear to indicate that OkW competes well with algorithms such as these and density based workhorses such as DBSCAN [13]. We plan to report on these issues in the near future [27] as well as on the challenges posed by OkW, e.g. when it is applied on problems of high dimensionality and the associated costs for large numbers of objects. We close by noting that oriented hyperrectangles have been under study in other contexts, including fast interference detection amongst geometric models and in computing approximations to the set of reachable states in a continuous state space in hybrid systems research [5, 15, 21, 25].

Acknowledgments

We thank the organizers of the "3rd World Conference on Computational Statistics & Data Analysis" and E. Kontoghiorghes in particular, for giving us the opportunity to present early segments of our work during that meeting. We thank the authors of [9] for allowing us to access their implementation of the 4C algorithm. We also thank P. Drineas for a useful discussion early in the course of this work and the reviewers for their careful reading and suggestions. This research was supported in part by a University of Patras "Karatheodori" grant. The second author was also supported by a Bodossaki Foundation graduate fellowship.

References

1. C.C. Aggarwal, A. Hinneburg, and D.A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Proc. 8th Int'l Conf. Database Theory (ICDT)*, pages 420–434, London, 2001.
2. P. Alevizos, D.K. Tasoulis, and M.N. Vrahatis. Parallelizing the unsupervised k-windows clustering algorithm. In R. Wyrzykowski, editor, *Lecture Notes in Computer Science*, volume 3019, pages 225–232. Springer-Verlag, 2004.
3. E. Anderson et al. *LAPACK Users' Guide*. SIAM, Philadelphia, 1999. 3d ed.
4. M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1999.
5. G. Barequet, B. Chazelle, L.J. Guibas, J.S.B. Mitchell, and A. Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum (CGF)*, 15(3):C387–396, Aug. 1996.
6. P. Berkhin. A survey of clustering data mining techniques. In J. Kogan, C. Nicholas, and M. Teboulle, editors, *Grouping Multidimensional Data: Recent Advances in Clustering*, pages 25–72. Springer, Berlin, 2006.
7. M.W. Berry. Large scale singular value decomposition. *Int. J. Supercomp. Appl.*, 6:13–49, 1992.
8. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

9. C. Bohm, K. Kailing, P. Kroger, and A. Zimek. Computing clusters of correlation connected objects. In *ACM SIGMOD international conference on Management of data*, pages 455–466. ACM Press, 2004.
10. K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *VLDB*, pages 89–100. Morgan Kaufmann Publishers Inc., 2000.
11. M. de Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000.
12. C. Domeniconi, D. Papadopoulos, D. Gunopulos, and S. Ma. Subspace clustering of high dimensional data. In *SIAM Data Mining 2004*, 2004.
13. M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int'l. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.
14. G.H. Golub and C.F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3d edition, 1996.
15. S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM SIGGRAPH*, pages 171–180, 1996.
16. A. Hinneburg, C. Aggarwal, and D. Keim. What is the nearest neighbor in high dimensional spaces? In *The VLDB Journal*, pages 506–515, 2000.
17. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
18. I. T. Jolliffe. *Principal Component Analysis*. New York: Spriger Verlag, 2002.
19. KDD Cup data. http://kdd.ics.uci.edu/databases/kddcup99/kd dcup99.html, 1999.
20. P. Kumar, J.S.B. Mitchell, and E.A. Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *J. Exp. Algorithmics*, 8, 2003.
21. J. McNames. Rotated partial distance search for faster vector quantization encoding. *IEEE Signal Processing Letters*, 7(9):244–246, September 2000.
22. F. Preparata and M. Shamos. *Computational Geometry*. Springer Verlag, New York, Berlin, 1985.
23. C.M. Procopiuc, M. Jones, P.K. Agarwal, and T.M. Murali. A Monte Carlo algorithm for fast projective clustering. In *Proc. 2002 ACM SIGMOD*, pages 418–427, New York, NY, USA, 2002. ACM Press.
24. M. Rigou, S. Sirmakessis, and A. Tsakalidis. A computational geometry approach to web personalization. In *IEEE International Conference on E-Commerce Technology (CEC'04)*, pages 377–380, San Diego, California, July 2004.
25. O. Stursberg and B.H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In O. Maler and A. Pnueli, editors, *HSCC*, volume 2623 of *Lecture Notes in Computer Science*, pages 482–497. Springer, 2003.
26. P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Addison-Wesley, Boston, 2005.
27. D.K. Tasoulis, D. Zeimpekis, E. Gallopoulos, and M.N. Vrahatis. Manuscript (In preparation), 2006.
28. M. N. Vrahatis, B. Boutsinas, P. Alevizos, and G. Pavlides. The new k-windows algorithm for improving the k-means clustering algorithm. *Journal of Complexity*, 18:375–391, 2002.