Mani-Web: Large-Scale Web Graph Embedding via Laplacian Eigenmap Approximation

Konstantinos Stamos, Nikolaos A. Laskaris, and Athena Vakali

***Abstract*—The Web as a graph can be embedded in a low-dimensional space where its geometry can be visualized and studied in order to mine interesting patterns such as web communities. The existing algorithms operate on small-to-medium-scale graphs; thus, we propose a close to linear time algorithm called Mani-Web suitable for large-scale graphs. The result is similar to the one produced by the manifold-learning technique Laplacian eigenmap that is tested on artificial manifolds and real web-graphs. Mani-Web can also be used as a general-purpose manifold-learning/dimensionality-reduction technique as long as the data can be represented as a graph.**

***Index Terms*—Laplacian eigenmap, large scale, manifold learning, spectral graph theory, web communities.**

I. Introduction

THE Web is an enormous information domain with its physical structure that is best described in the form of a graph. Distinct pages are represented via nodes and the related hyperlinks are represented via graph edges. The graph is readily available to be studied via web-link-mining [1], [2], with its outcomes that are expected to improve the overall experience for the users, save resources for the website owners, and provide a better understanding of the Web.

Given a large-scale graph we should be able to mine interesting patterns iff they exist and represent them in way that they can be exploited. Such patterns include the communities (clusters), outlier nodes, and in general the intrinsic geometry/layout of the graph. The problem for small-to-medium graphs (i.e., thousand of nodes) only is dealt with traditional graph mining algorithms and manifold learning techniques with high-time complexity. Motivated by the inability of the existing methods to handle very large-scale graph data, our contributions are summarized as follows.

1) We suggest a general purpose manifold-learning technique, called Mani-Web (http://oswinds.csd.auth.gr/∼maniweb), a linear-time approximation of the Laplacian eigenmap [3].
2) We benchmark the scalability and the correctness of the algorithm experimentally by the usage of artificial manifolds and real web-graphs and we provide preliminary evidence that Mani-Web could be applied in the context of a content distribution network for latency and content management improvement.

Manuscript received February 4, 2010; revised March 14, 2011; accepted May 18, 2011. Date of publication August 4, 2011; date of current version October 12, 2012. This paper was recommended by Associate Editor Y. Jin.

The authors are with the Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 541 24, Greece (e-mail: kstamos@csd.auth.gr; laskaris@aiia.csd.auth.gr; avakali@csd.auth.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCC.2011.2160166

TABLE I
COMPARISON OF MANIFOLD LEARNING METHODS ON GRAPHS ACCORDING TO GRAPH SIZE $|V|$

Technique	Time complexity
Isomap	$O(\|V\|^3)$
MVU	$O(\|V\|^3)$
Diffusion maps	$O(\|V\|^3)$
LLE	$O(\|V\|^2)$
Laplacian Eigenmap	$O(\|V\|^2)$
Hessian LLE	$O(\|V\|^2)$
LTSA	$O(\|V\|^2)$
Mani-Web	$O(\|Vr\|\|V\|)$, $\|Vr\| \ll \|V\|$
CiBC	$O(\sqrt{\|V\|}^3)$

II. Related Work

The manifold-learning approach assumes that the data belong to a low-dimensional manifold that is embedded in the original high-dimensional space. A typical strategy is to construct a graph encoding the interrelations of the data as in Fig. 2. Then, the task is to properly unfold this graph and embed it in a low-dimensional space. The advantage of this approach is that complex overlapping geometries, clusters, and data of nonlinear nature can be effectively represented. Additionally, in cases where the graph is naturally available as in the Web the methods can be applied directly.

A typical list of graph-based manifold learning methods includes: isomap [4], MVU [5], diffusion maps [6], LLE [7], Laplacian eigenmap [3], Hessian LLE [8], and LTSA [9]. They involve eigen-analysis, graph traversal techniques, and other computationally demanding operations, which are in general sensitive to the scale of the graph. A comparison in terms of time complexity with respect to the graph size $|V|$ is available in Table I. All of them are of polynomial complexity $O(|V|^2)$ or $O(|V|^3)$. Although the complexity is reasonable and benefits from the sparsity of the graphs as we scale up they become incomputable. Of them particular interest has the Laplacian eigenmap because it has a natural tendency to enhance the formation of the communities.

A (Web) community can be considered to be a coherent cluster of pages that has significantly more hyper-links that point among pages that belong to the community itself than the rest of the graph. This structural attribute emerges from the fact that pages with common subject often reference one another as related sources. A user, who enters a Web community, is "trapped"

in the sense that the probability of visiting a page outside the community is lower, because of the fewer outgoing hyperlinks.

The community detection, an NP-hard [10] problem, has drawn a lot of attention [11]–[18] recently. The respective algorithms are characterized by exponential [19] or high polynomial complexity [11] in terms of number of pages and hyperlinks. A recent algorithm that has interesting application in the context of a content distribution network is CiBC [18] that is suitable for medium size graphs.

Visualization can be the product of a low-dimensional representation of a graph. Toward this direction there are several algorithms [20]–[23], and [24], [25] for the non-Web-related content. However, none of them has been focused on visualizing web-graphs in particular.

III. "Walking" on a Manifold

In this section, we give a qualitative definition of a manifold, its relation to the web-graph, and initial insight into the ideas developed later. Strict mathematical definitions can be found in [26].

A. What is a Manifold

A *manifold* is a topological space that only locally exhibits flat conventional geometry. On global scale, it demonstrates profound structural hyperorganization. In the space of a manifold the conventional Euclidean distances are less important as they do not capture the manifold's geometry. Consider, as toy example, Archimedes' spiral manifold in Fig. 2. The spiral itself comprises of a swarm of 1000 points in a two-dimensional (2-D) space coordinate. Although we can evaluate the Euclidean distance from point labelled 1 to 2, the result would be contrary to our intuition. A more appropriate distance evaluation should incorporate the intrinsic geometry of the manifold by imitating a "walk" on the manifold, from point 1 to 2.

In order to perform a walk on the manifold, a graph that connects nearby points in the space is formed. The graph encodes local relations, building essentially a graph of the data. The graph itself limits the ways in which the data points can be accessed from one another since "communication" is possible only through the available edges.

B. Web-Graph Representation

Let us consider a population V of web pages (nodes). For each pair $\langle i, j\rangle, i, j \in V$ an integer value $E_{i,j} \in (0, 1)$ is defined. The value 1 indicates the existence of a hyperlink (edge) originating from i and leading directly to j, while 0 indicates no direct association. A graph $G(V, E)$ (or just G) is constructed, where E is a set of edges that connects the nodes in G. Thus, the $|V| \times |V|$ adjacencymatrix A is formed that contains all the pairwise $E_{i,j}$ values. We consider a connected undirected graph, without self-loops and multiple edges between any two nodes. Therefore, A is symmetric with zeroed diagonal.

Despite the fact that the web-graph is not symmetric and contains self-loops and multiple edges, we can easily meet the aforementioned requirements. We may safely perform a symmetrizing conversion of A since the user can access the previous pages by the usage of the "Back" button of the web browser, even if there is no backward association available. Moreover, self-loops indicate self-association; thus, they do not contribute much to the structural information. Multiple edges, on the other hand, can simply be aggregated into a single edge.

TABLE II
VARIABLES REFERENCE

G	Graph
V	Nodes
E	Edges
A	Adjacency matrix
$Anorm$	Normalized adjacency matrix
i	A node
j	Another node
Vr	boundary-nodes
Ar	Adjacency matrix of the reduced-graph
Dr	Degree matrix of the reduced-graph
Lr	Laplacian matrix of the reduced-graph
Yr	Embedding of reduced-graph
Y	Embedding of graph
$EigValr$	Eigenvalues of the reduced graph embedding
$Tolerance$	Quality setting of Mani-Web
$GeometricFluctuations$	The fluctuations of flow per boundary-node
$FlowIterations$	Iterations of flow per boundary-node
Cpu	Time elapsed
$Sources$	The nodes of the graph as sources of flow
$InitialFlow$	The initial flow of the sources
$Flows$	The distributed flow originating from the increasing in number boundary-nodes
$FlowsSingle$	The distributed flow originating from each boundary-node

IV. Mani-Web

In this section, the Mani-Web algorithm is introduced with the help of a toy example. The variables are summarized in the reference Table II. The algorithm itself consists of three phases namely maniReduce(), maniReconstruct(), and maniMap() (the appearance of "()" will denote functions). The complete listing of the code is available in Fig. 1 (by the usage of MATLAB syntax, ready to copy-paste).

The outline of the algorithm is as follows: Select a few boundary-nodes, construct a reduced-graph by the usage of the boundary-nodes, produce a Laplacian eignemap for the reduced-graph, and finally extrapolate the map for all nodes. The result is a map of all nodes in a (Euclidean) coordinate space with only marginal error against the original Laplacian eigenmap. Then, by the selection of the second, third (and optionally fourth) dimensions we produce a low-dimensional image of the original graph.

A. Starting the algorithm: $maniweb()$

A is the input adjacency matrix in sparse format, and Tolerance $\in (01]$ is the quality setting for the algorithm. Vr is the set of boundary-nodes, Yr is the Laplacian eigenmap embedding of the reduced-graph, EigValr is the eigenvalues associated with the Laplacian eigenmap of Ar, Ar is the adjacency matrix of the reduced-graph, Y is the final embedding of all nodes, GeometricFluctuations is the geometric fluctuation of the algorithm, FlowIterations is the iterations of each step of the algorithm, and Cpu is the CPU time wasted.

```
function [Vr Yr EigValr Ar Y GeometricFluctuations FlowIterations Cpu] = ...
      maniweb(A, Tolerance) %A the adjacency matrix of a connected graph, ...
      Tolerance in range (0 inf], then plot3(Y(:,2), Y(:,3), Y(:,4))
Start = cputime();
[Vr Flows GeometricFluctuations FlowIterations] = maniReduce(A, Tolerance);
[Ar FlowsSingle] = maniReconstruct(Vr, Flows);
[Yr EigValr Y] = maniMap(Ar, FlowsSingle);
Cpu = cputime() - Start;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Vr Flows GeometricFluctuations FlowIterations] = maniReduce(A, Tolerance)
Anorm = anorm(A);
V = 1 : length(Anorm);
Vr = 1;
Flows = [];
GeometricFluctuations = [];
FlowIterations = [];
FlowOfPreviousBoundaryNode = 0;
for i=1:length(Anorm)-1,
  [Flow Iterations] = flow(Anorm, Vr, Tolerance);
  Flows = [Flows Flow];
  FlowIterations = [FlowIterations Iterations];
  Targets = setdiff(V,Vr);
  [FlowOfNewBoundaryNode, BoundaryNodeIndex] = min(Flow(Targets));
  NewBoundaryNode = Targets (BoundaryNodeIndex);
  Vr = [Vr NewBoundaryNode];
  GeometricFluctuation = abs(FlowOfNewBoundaryNode - ...
        FlowOfPreviousBoundaryNode) / max(FlowOfNewBoundaryNode, ...
        FlowOfPreviousBoundaryNode)
  GeometricFluctuations = [GeometricFluctuations GeometricFluctuation];
  if GeometricFluctuation <= Tolerance,
    break
  end
  FlowOfPreviousBoundaryNode = FlowOfNewBoundaryNode;
end
[Flow Iterations] = flow(Anorm, Vr, Tolerance);
Flows = [Flows Flow];
FlowIterations = [FlowIterations Iterations];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Ar FlowsSingle] = maniReconstruct(Vr, Flows)
FlowsSingle = Flows;
NboundaryNodes = length(Vr);
for i=NboundaryNodes:-1:2,
  FlowsSingle(:,i) = FlowsSingle(:,i) ./ FlowsSingle(:,i-1);
end
for i=1:NboundaryNodes,
  FlowsSingle(:,i) = normalize01(FlowsSingle(:,i));
end
NaNEntries = isnan(FlowsSingle);
FlowsSingle(NaNEntries) = 0;
InfEntries = isinf(FlowsSingle);
FlowsSingle(InfEntries) = 0;
Ar = FlowsSingle(Vr,:);
Ar = max(Ar, Ar');
Ar = Ar - diag(diag(Ar));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Yr EigValr Y] = maniMap(Ar, FlowsSingle)
Ndims = length(Ar);
Nnodes = size(FlowsSingle, 1);
Y = zeros(Nnodes, Ndims);
[Yr EigValr] = laplacianEigenmap(Ar);
for i = 1 : Nnodes,
  Flow = FlowsSingle(i,:);
  for d=1:Ndims,
    Y(i,d) = Flow*Yr(:,d) / sum(Flow);
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Flow Iterations] = flow(Anorm, Sources, Tolerance)
Nnodes = length(Anorm);
Flow = zeros(Nnodes, 1);
Flow(Sources) = 1;
InitialFlow = zeros(Nnodes, 1);
InitialFlow(Sources) = 1;
PreviousFlow = zeros(Nnodes, 1);
Iterations = 0;
while true,
    Iterations = Iterations + 1;
    Flow =  ((1/(1+Tolerance)) * Anorm * Flow) + ((1-(1/(1+Tolerance))) * ...
          InitialFlow);
    if isempty(find(Flow == 0, 1)),
      if isempty(find(abs(Flow - PreviousFlow) ./ max(Flow, PreviousFlow) > ...
            Tolerance, 1)),
        break;
      end
      PreviousFlow = Flow;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Yr EigValr] = laplacianEigenmap(Ar)
Dr = diag(sum(Ar));
Lr = Dr - Ar;
[Yr, EigValr] = eig(full(Lr), full(Dr));
EigValr = diag(EigValr);
[EigValr, indices] = sort(EigValr,'ascend');
Yr = Yr(:, indices);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Anorm = anorm(A)
Anorm = A;
Anorm = max(Anorm, Anorm');
Anorm = Anorm - sparse(diag(diag(Anorm)));
D = diag(sum(Anorm).^(-1/2));
Anorm = D * Anorm * D;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Flows = normalize01(Flows)
Flows = (Flows - min(Flows)) ./ (max(Flows) - min(Flows));
```

Fig. 1. Mani-Web.

B. *Phase:* $maniReduce()$

The aim of this phase is to scale down the graph. The idea is to detect representative nodes, the so-called boundary-nodes, that wrap the global geometry of the manifold.

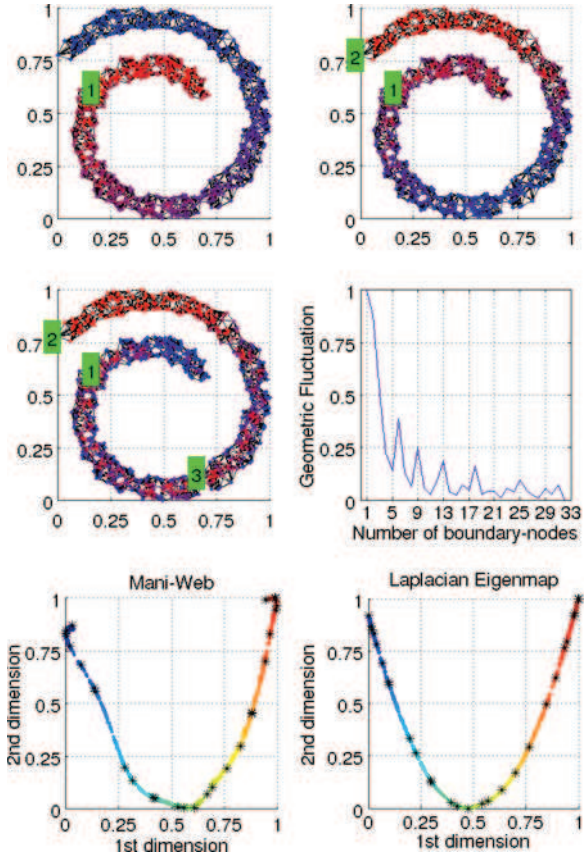

Fig. 2. Archimedes spiral.

At this point we have to deal with the following questions.

1) *What "distance" means?*
2) *How to select the boundary-nodes?*
3) *How many boundary-nodes should we select?*

What *"distance" means?:* In Section III the notion of walking on a graph was discussed which can be used to measure distance between a set of source-nodes and a set of target-nodes. Such a walk can be emulated by the usage of the iterative version of the PageRank-like algorithm presented in [27], [28] and implemented in flow(). Given the normalized version of the graph Anorm [produced by anorm()], the source-nodes Sources (the target-nodes are the rest), and Tolerance, an InitialFlow value that originates from the Sources is diffused throughout the graph until convergence. Usually a few iterations are enough in a small-world graph (like Web) to achieve high-quality results. The function returns Flow which is a column that contains the flow of each node from the perspective of the Sources. This can be considered to be an $1 - D$ ranking/ordering/closeness of the nodes. The node that does not belong to Sources and has the lowest flow is the most distant to Sources. Another return value is the Iterations, the number of iterations performed until convergence.

For example, consider the Archimedes' spiral in Fig. 2. The coloration of the nodes represents the smooth distribution of the flow in the graph. Therefore, the most distant node from the perspective of node 1 is the node 2, and from the perspective of nodes $1, 2$ is node 3.

How to select the boundary-nodes?: Assuming that we know the number of necessary boundary-nodes beforehand then we have to examine all the possible combinations of nodes in order to find the combination that maximizes the overall mutual distance among the boundary-nodes (in other words, minimizes their mutual flow). Of course, this approach is of factorial complexity and infeasible for large graphs.

The suggested approach is to perform a progressive selection of the boundary-nodes. Starting from an initial arbitrary node say node 1 we select the most distant one say node 2, then the most distant one from nodes $1, 2$ say node 3 and so on. This approach is illustrated in Fig. 2 where the boundary-nodes are progressively selected by by the maximization of their mutual distance (in other words, by the maximization of the flow).

How many boundary-nodes should we select?: By increasing the number of Sources (as the number of boundary-nodes increases) the overall flow distributed to the graph increases as well. This gradual increment of the overall flow produces a convergent sequence of values. The sequence contains the flow fluctuation of each boundary-node against the previous one at the time of its selection. Therefore, it is reasonable to use the Tolerance as convergence criterion of the sequence. Letting the overall geometry of the manifold and distribution of the flow to decide.

Fig. 2 illustrates the concept of geometric convergence by depicting the fluctuations between two consecutive boundary-nodes as the sampling progresses. Effectively we may capture the geometric convergence either by setting the Tolerance at a reasonable value (e.g., 0.01 or 0.001) or by observing the convergence on-line.

The result of the reduction: In maniReduce() we have A which is the adjacency matrix and Tolerance for input. The Vr is the set of the boundary-nodes, Flows is a $|V| \times |Vr|$ matrix that contains the flow of each node that originates from the rolling set of boundary-nodes. More specifically, the first column of Flows contains the flows that originate from the first boundary-node, the second column from the first two boundary-nodes, the third column from the first three, and so on. The GeometricFluctuations is the sequence of geometric fluctuations that produced during the sampling.

C. Phase: $maniReconstruct()$

To construct Ar, we need to encode the pairwise flows among the Vr nodes by taking into account the in-between structure in the original graph. By dividing each column of Flows with the previous one, by the application of normalization [normalize01()], and by the selection of the Vr rows we produce the Ar. A final postprocessing step is to make sure that no NaN/Inf values exist (due to arithmetic representation), the matrix is symmetrical and of zero diagonal.

If effect, Ar is a graph that should wrap the original graph in terms of Laplacian eigenmap embedding as its nodes reside at the boundary of the manifold's geometry. Finally, we have Ar and FlowsSingle that represents a $1-D$ ordering from the perspective of each boundary-node.

D. Phase: $maniMap()$

This final phase produces the actual embedding/map of all the nodes V of the original graph G into a $|Vr|$-dimensional space. More precisely in maniMap() the input is the Ar, FlowsSingle. The output is Yr the Laplacian eigenmap embedding of Ar, EigValr the set of eigenvalues of the Laplacian eigenmap of Ar, and Y is the $|Vr|$-dimensional embedding of all nodes that approximates the Laplacian eigenmap as if it was applied directly to the original map.

The key idea is to produce the Laplacian eigenmap of the boundary-nodes by the usage of Ar and use it as a fence bounding the global geometry of the manifold. Then the Cartesian positions of all nodes and for all dimensions are extrapolated. For all dimensions and nodes.

1) Get the respective row in FlowsSingle for a node.
2) For each dimension calculate the weighted average of Yr with the respective row of FlowsSingle. That is the position of the node at this dimension. This position is derived from the various levels of influence of every boundary-node in terms of flow. In other words, the Cartesian positions of all the boundary-nodes are being diffused through the graph via the precalculated flow.

In order to visualize the result, or keep only the most important portion of geometric information we select the columns $2, 3$, and optionally 4 of Y to produce a $2-D$ or $3-D$ image. Applying Mani-Web in the toy example [see Fig. 2] is a faithful approximation of the original Laplacian eigenmap is produced for Tolerance $= 0.001$. The boundary-nodes are highlighted in black and they, in fact, bound the geometry of the spiral. The rest nodes are uniquely colored for comparison. We should note that the small discrepancy in the image is caused by the fact that we have very long paths in the graph (1000 hops on the average) causing the propagation of flow to stop before full convergence. However, this is not the case in the Web as the number of hops between any two nodes is a few tens due to the "small world" phenomenon.

V. Underlying Theory

A. Flow as $1-D$ Ranking

The flow propagation results in a nonlinear smooth ranking in one dimension which is biased toward the sources with a marginal tolerance Tolerance. The column Flow holds the updated flow of all nodes in Flow(), while the Tolerance acts as a balancing factor that causes the final ordering of nodes to be biased towards the Sources. We would like to let the network $[(1/(1+\text{Tolerance})) * \text{Anorm} * \text{Flow}]$ to be more important in terms of ranking, thus, capturing the structure of the graph in terms of flow. The other part $[(1-(1/(1+\text{Tolerance}))) * \text{InitialFlow}]$ is important to cause the bias, but we would like this to be as small as possible in order not to interfere much during the discovery of the structure of the graph.

Expression (1) is the minimization problem solved by the flow(). The problem is to smoothly assign Flow values to all nodes. For the proof see [28] and [27]. This expression has a natural geometric explanation for the graph. For nodes, i.e.,

$i, j \in V$ the term, i.e., $\sum_{i,j}^{|V|} A_{i,j} \cdot \| \frac{1}{\sqrt{D_{ii}}} \cdot \text{Flow}_i - \frac{1}{\sqrt{D_{jj}}} \cdot \text{Flow}_j \|^2$ expresses the error of ranking. If $\| \frac{1}{\sqrt{D_{ii}}} \cdot \text{Flow}_i - \frac{1}{\sqrt{D_{jj}}} \cdot \text{Flow}_j \|$ has a high value, meaning that i, j have a very different flow, then by being multiplied with $A_{i,j}$ we get a high penalty if indeed $A_{i,j} \neq 0$.

$$\underset{\text{Flow}}{\text{argmin}} \left(\frac{1}{2} \cdot \left(\sum_{i,j}^{|V|} A_{i,j} \cdot \left\| \frac{1}{\sqrt{D_{ii}}} \cdot \text{Flow}_i - \frac{1}{\sqrt{D_{jj}}} \cdot \text{Flow}_j \right\|^2 + \text{Tolerance} \cdot \sum_{i=1}^{|V|} \|\text{Flow}_i - \text{InitialFlow}_i\|^2 \right) \right) \tag{1}$$

The term Tolerance $\sum_{i=1}^{|V|} \| \text{Flow}_i - \text{InitialFlow}_i \|^2$ has also a geometric meaning. If there is a lot of divergence between the initial and the final flow we get a penalty according to Tolerance.

B. Heuristic $1 - D$ Relation of Flow With the Laplacian Eigenmap

The Laplacian eigenmap has a natural geometric meaning for the reduced-graph (see [3] for proof). It minimizes (2) by finding the appropriate Yr vectors for each boundary-node. The geometric interpretation is evident in (3), and the left-hand part of the equation occurs by trivial matrix operations on the right-hand. The matrix Lr is the Laplacian matrix, the analogous of Laplace–Beltrami operator but for discrete structures, with $Lr = Dr - Ar$ and Dr being a diagonal matrix with the degree of each node Vr or just the sum of each column of Ar. Equation (3) has the same geometric interpretation as the first term of the flow (1).

$$\underset{Yr^T \cdot Dr \cdot Yr = I}{\text{argmin}} \quad \text{tr}(Yr^T \cdot Lr \cdot Yr) \tag{2}$$

$$\sum_{i,j}^{|Vr|} Ar_{i,j} \cdot \|Yr_i - Yr_j\|^2 = \text{tr}(Yr^T \cdot Lr \cdot Yr) \tag{3}$$

By the solution of the generalized eigenvalue problem in (4) we solve essentially the minimization problem in (2). We get an ordered set of ascending eigenvalues and their respective eigenvectors, with the first eigenvalue being the trivial case that always has the value 0 for a graph with a single component. The meaning of eigenvalues is evident in (5) where the magnitude of each eigenvalue expresses a penalty of mapping error. Thus, ignoring the first eigenvalue/eigenvector and using the second, third, and possibly fourth eigenvectors a low-dimensional mapping is produced.

$$Lr \cdot Yr = \text{EigValr} \cdot Dr \cdot Yr \tag{4}$$

$$\sum_{i,j}^{|Vr|} Ar_{i,j} \cdot \|Yr_i - Yr_j\|^2 = \sum_{i}^{|Vr|} \text{EigValr}_i \tag{5}$$

We observe that, for a single dimension, the left-hand term of (3) is closely related to (1) if Tolerance $\to 0$. Thus, by normalizing the flows of all nodes we can deduce an approximate mapping of all nodes in the $|Vr|$-dimensional space of the Laplacian eigenmap by diffusing and averaging the embedding Yr of the boundary-nodes by the usage of the flow originating from each boundary-node Vr. Of course, the mapping will not be perfect but it will be correct within Tolerance.

C. Time and Space Complexity

The time complexity of Mani-Web in Big-O notation can be defined as $O(\text{maniweb}) = O(\text{maniReduce}) + O(\text{maniReconstruct}) + O(\text{maniMap})$. For each part:

1) $O(\text{maniReduce}) = |Vr| \cdot O(\text{flow})$. Because $|Vr|$ is small, $O(\text{maniReduce}) = O(\text{flow}) = O(\text{Iterations} \cdot |V|)$. Although Iterations is unknown we are aware that for the Web due to "small world phenomenon" it should be small. Thus, $O(\text{flow}) = O(|V|)$. For graphs that exhibit structures with very long paths like the spiral Iterations can be significantly large requiring, also, small values of Tolerance.
2) $O(\text{maniReconstruct}) = O(|Vr| \cdot |V|)$ as the algorithms just parses the FlowsSingle matrix.
3) $O(\text{maniMap}) = O(\text{laplacianEigenmap}) + O(|Vr| \cdot |V|)$ and as the laplacianEigenmap operates on a very small matrix we can safely ignore it and say that $O(\text{maniMap}) = O(|Vr| \cdot |V|)$.

From the aforementioned the time complexity is $O(\text{maniweb}) = O(|Vr| \cdot |V|)$.

In an analogous manner the space complexity in Big-O notation can be defined as $O(\text{maniweb}) = O(\text{maniReduce}) + O(\text{maniReconstruct}) + O(\text{maniMap})$. The entire algorithm demands storage of the full graph in sparse format that is $O(|E|)$ and for each particular function we have the following.

1) $O(\text{maniReduce}) = O(|Vr| \cdot |V|)$ as the biggest matrix produced is the Flows.
2) $O(\text{maniReconstruct}) = O(|Vr| \cdot |V|)$ as the biggest matrix is the FlowsSingle.
3) $O(\text{maniMap}) = O(|Vr| \cdot |V|)$ as the biggest matrices are the FlowsSingle and Y.

Thus, $O(\text{maniweb}) = O(|E| + |Vr| \cdot |V|)$ then the space complexity is $O(\text{maniweb}) = O(|E|)$.

D. Input Graph Restrictions

In Section III we defined the graph G to be symmetric, non-loop, connected, and positively weighted. These requirements are related to the graph normalization anorm(). Of course we may only require the graph to be connected and positively weighted by altering anorm() to produce the traditional transition matrix of PageRank where each column sums to 1. The results are similar to the default normalization method. Another reason for which we chose this normalization method and to force the full graph to be symmetric is to prevent sinks where a few nodes absorb all the flow since random jumps are not included in our approach as we meant to approximate Laplacian eigenmap. The ability to replace the normalized matrix with another that does not need to obey the symmetricity and no-self

connections requirement is an advantage of Mani-Web against the original Laplacian eigenmap.

VI. Mani-Web benchmark

As the Mani-Web is an approximation of the original Laplacian eigenmap it is reasonable to assess the performance and quality representation by comparison.

A. Representation quality

We quantify quality by estimating the errors between the embedding produced from Mani-Web against the Laplacian eigenmap on artificial manifolds. We evaluate the $k = 1 : |V|$ nearest neighbor set for all nodes and we average all the relative disagreements. That way we examine both local and global preservation of the embedding where values close to zero indicate good representation.

Fig. 3 illustrates each artificial manifold in the first column, the Laplacian eigenmap result in the second, and the Mani-Web result in the third (Tolerance $= 0.001$). The presented manifolds starting from the top row are: 3-D clusters, corner planes, Gaussian, occluded disks, punctured sphere, swiss roll, swiss hole, toroidal helix, and twin peaks. Fig. 4 records the nearest neighbor error for varying levels of Tolerance. As it can be seen in both figures the results are both visually acceptable and typically below 10%. These artificial manifolds challenge Mani-Web as they do not demonstrate "small world" phenomena thus the flow() converges slowly. The datasets include 800 points and the respective graphs are constructed by connecting the eight nearest neighbors that uses the MANIfold Learning MATLAB Demo available at http://www.math.ucla.edu/∼wittman/mani. Fig. 5 shows the linear increment of the boundary nodes as Tolerance is reduced.

B. Scale up

For experimentation purposes we have constructed low density random graphs with $|V| = 2^{(2:22)}$. Fig. 6 demonstrates the true strength of Mani-Web (Tolerance $= 0.01$), close to linear complexity, even for very large graphs (4 million nodes). As can be seen, the original Laplacian eigenmap, managed to run only for at most $|V| = 32\,768$ both because of the time and space complexity (Matlab eigs implementation [29], [30]).

VII. Mani-Web on Web Graphs

We have crawled two websites, the National Geographic with $|V| = 58\,347$ and the Wikipedia on DVD with $|V| = 19\,488$ and run Mani-Web for Tolerance $= 0.01$.

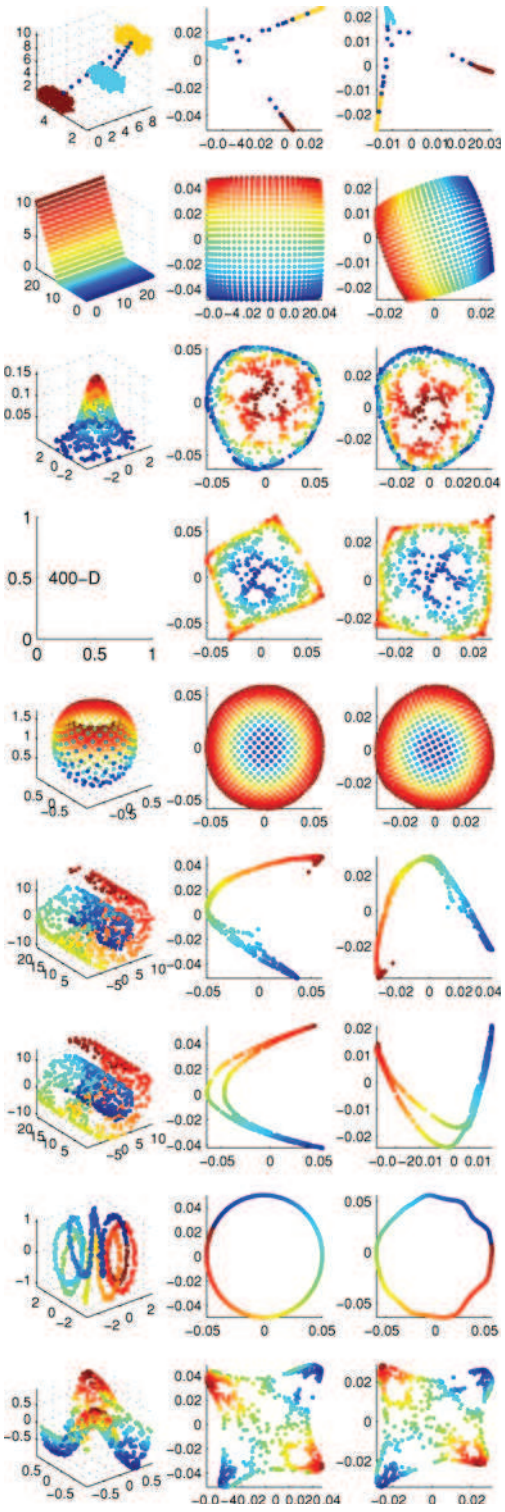

Fig. 3. Manifold Laplacian eigenmap. Mani-Web.

A. Locating Web-Communities

The result of Mani-Web for the National Geographic is depicted in Fig. 8. This image is not the entire map, but only a zoomed area. The entire graph is full of Web communities of varying sizes appearing as high-density areas. Fig. 9 is a plot of the eigenvalues that indicates that only two dimensions are enough to represent the geometry.

The embedding of the Wikipedia on DVD site is illustrated in Fig. 7, zoomed at a specific area. The absence of Web communities is apparent as there are no crisp densities. This can be explained by the fact that each article may link to many others unrelated thematically just because they share a common keyword for which an article exists. Finally, Fig. 9, displays the

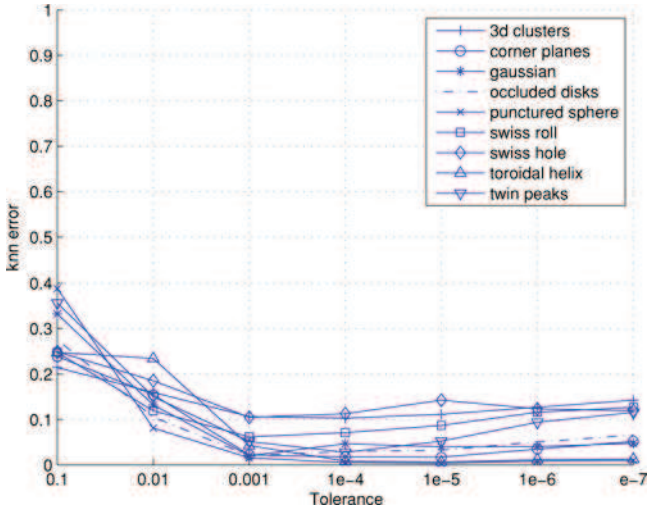

Fig. 4. Effect of Tolerance in representation quality.

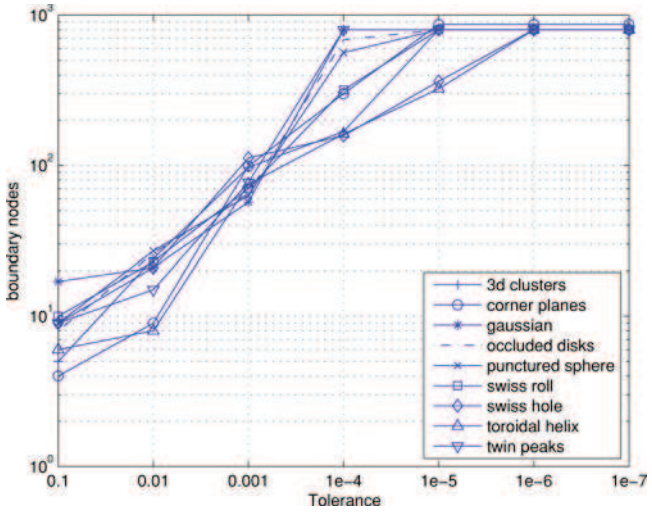

Fig. 5. Effect of Tolerance on $|Vr|$.

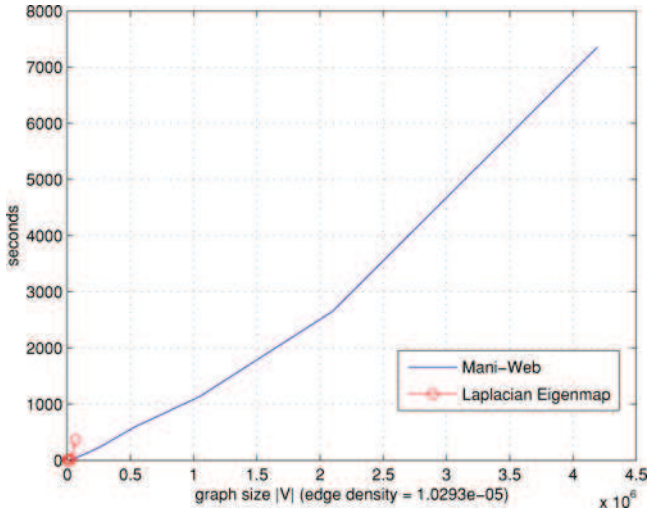

Fig. 6. Scalability of Mani-web.

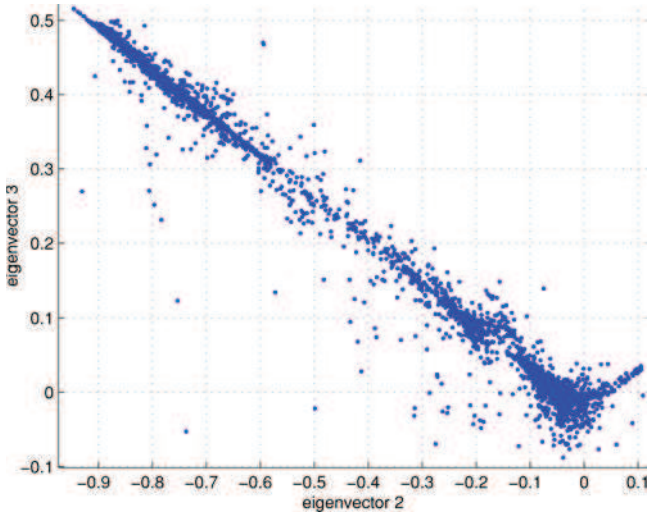

Fig. 7. Wikipedia on DVD Web site: Mani-web.

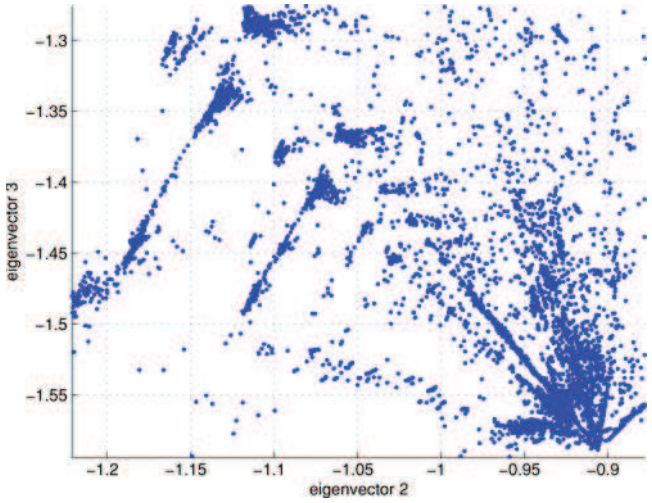

Fig. 8. National Geographic Web site: Mani-web.

eigenvalues, which indicate that only two dimensions suffice for the mapping.

B. Web-Communities as Client Attractors

Given a set of cooperating surrogate servers, a set of clients, and a website, the problem is to identify web-communities and to place them inside the limited capacity caches of the surrogate servers. By distributing the web-communities in the surrogate servers we proactively predict the clients' requests as a random surfer will most likely visit pages of the same community. A good community is the one that predicts many requests and is as small as possible.

One of the state-of-the-art algorithms in this area is the CiBC [18]. According to CiBC the nodes of the graph are selected by preferring the ones with the smallest betweenness centrality [31] at the bounds of the graph. Then, the nodes are being merged into possibly overlapping communities and the process iterates until no further merging can lead to a better community. Due to its polynomial complexity CiBC is suitable for medium-scale

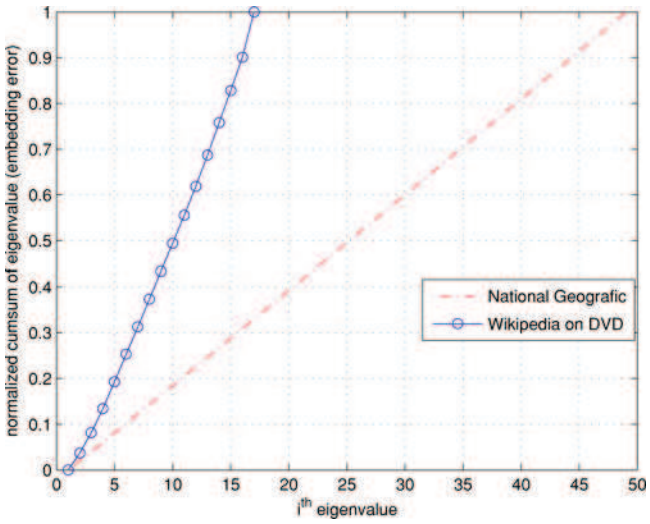

Fig. 9. Increment of error by using extra dimensions. $|Vr|$ is the maximum number of dimensions.

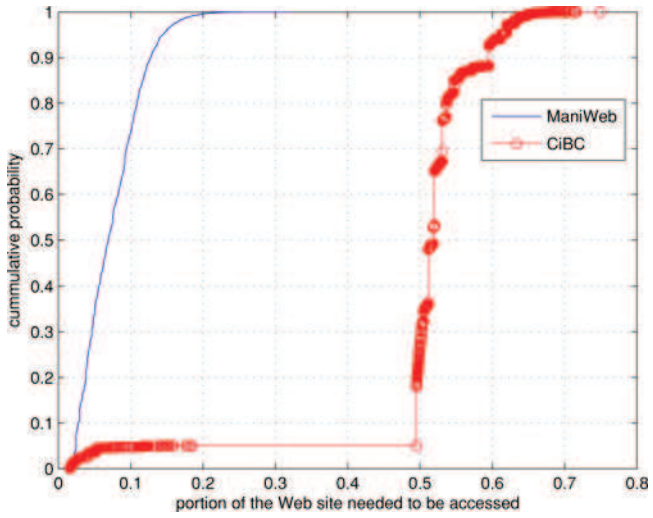

Fig. 10. CDN cache performance.

graphs on the context of an entire website only. As is focused on optimizing the content delivery of a CDN, CiBC detects communities that most likely capture many requests of a random surfer.

At this point we give a possible practical application of Mani-Web in the context of CDNs by comparison with the CiBC algorithm. By the usage of the National Geographic graph we applied the CiBC algorithm resulting into 98 communities. By the usage of the same number of communities we performed k-means clustering on the Mani-Web 2-Dresult and the resulting clusters are the considered communities for Mani-Web. Using the same tool that CiBC was originally tested on we have generated $150\,000$ client transactions. A transaction is a sequence of web pages that a random surfer could visit if she followed the web-graph linkage probabilistically.

The cost of any transaction is defined as the sum of the nodes that belongs to every community that are necessary to satisfy the transaction. If for many transactions only a small portion of the website is required to be present in the cache then we consider it to be a good clustering. Fig. 10 records the quality of Mani-Web and CiBC as a cumulative distribution function plot. The x-axis represents the portion of the website that is required for a transaction to be successful. The y-axis represents the cumulative probability for a transaction to be satisfied if a specific portion of the website is available. For a clustering to be good the curve must have a steep curve early on and to cover with high probability many transactions given a very small website portion. As can be seen Mani-Web demonstrates this characteristic outperforming CiBC. CiBC has detected a few very large clusters, that absorb many transactions that lead to unoptimized cache usage. On the other hand, Mani-Web managed to outline well-sized coherent clusters as is clearly visible from the curve. Although the results that encourage further investigation is necessary.

VIII. Mani-Web in practice

Mani-Web could be exploited by almost every topic relevant to dimensionality reduction, graph embedding, data mining, and visualization. This is true since whenever Laplacian eigenmap is suitable, Mani-Web can be applied too with appropriate tuning.

On the Web scene, we identify two major target groups that could benefit from Mani-Web: 1) the Web site owners and 2) the users who access the website, in the following areas.

1) *Website administration*: For instance, a website that contains thematic categories and embeds community organization could be more easily navigated by users, since once a user reaches a page of interest, relevant pages could be easily accessed (due to the communities increased linkage density). In a community-absent website, inferior user experience can be easily handled by the administrators who could use the Mani-Web maps to tune usage mining.
2) *Caching and prefetching*: Once a 2-D display is produced, the administrator, by the usage of a "lasso" tool, the revealed communities can be selected and become available as outsourcing units for the CDN. The selected communities are also suitable for caching and prefetching since they can predict users navigation, due to their dense linkage and the fact that they deal with coherent topics. Therefore, the communities may reduce the latency significantly, if they are placed in a CDN or in a traditional Proxy server.
3) *Information retrieval*: The reduced-graph, produced by Mani-Web can be exploited as an index structure of the original full data representation graph. In the context of search engines, an algorithm could use the Mani-Web reduced-graph to speed up the initial filtering stage of a query and at the refinement stage would have to partially retrieve more nodes according to the distributed flow. Another use of Mani-Web could be in interactive recommendation engines since given an initial user choice, the system can quickly examine the position of this choice in the Mani-Web map and then provide a set of relevant choices ranked by the flow.

IX. Conclusion

Bringing together, and in the context of Web, *manifold-learning theory*, *graph-embedding,* and *data-reduction*, Mani-Web as an efficient algorithm for embedding large-scale graphs within a low-dimensional coordinate space has been proposed. Mani-Web produces a map faithful to the Laplacian eigenmap but of close to linear time complexity with acceptable error. Future research directions may include time varying networks and content/label-loaded graphs.

References

[1] D. J. Cook and L. B. Holder, *Mining Graph Data*. New York: Wiley, 2006.

[2] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. New York: Springer-Verlag, Jan. 2007.

[3] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Comput.*, vol. 15, no. 6, pp. 1373–1396, 2003.

[4] J. B. Tenenbaum, V. Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000.

[5] K. Q. Weinberger and L. K. Saul, "An introduction to nonlinear dimensionality reduction by maximum variance unfolding," in *Unfolding, Proceedings of the 21st National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI, 2006.

[6] S. Lafon and A. B. Lee, "Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning and data set parameterization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, pp. 1393–1403, 2006.

[7] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, pp. 2323–2326, 2000.

[8] D. L. Donoho and C. Grimes, "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data," *PNAS*, vol. 100, no. 10, pp. 5591–5596, May 2003.

[9] Z. Zhang and H. Zha, "Principal manifolds and nonlinear dimensionality reduction via tangent space alignment," *SIAM J. Sci. Comput.*, vol. 26, pp. 313–338, Jan. 2005.

[10] N. Bansal, A. Blum, and S. Chawla, "Correlation clustering," in *Proceedings of the 43rd Symposium on Foundations of Computer Science* Washington, DC: IEEE Computer Society, 2002, p. 238.

[11] A. Sidiropoulos, G. Pallis, D. Katsaros, K. Stamos, A. Vakali, and Y. Manolopoulos, "Prefetching in content distribution networks via web communities identification and outsourcing," *World Wide Web*, vol. 11, no. 1, pp. 39–70, 2008.

[12] N. Eiron and K. S. McCurley, "Untangling compound documents on the web," in *Proceedings of the 14th ACM Conference on Hypertext and Hypermedia*, New York: ACM, 2003, pp. 85–94.

[13] W.-S. Li, K. S. Candan, Q. Vu, and D. Agrawal, "Query relaxation by structure and semantics for retrieval of logical web documents," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 4, pp. 768–791, 2002.

[14] W. Li, K. S. Candan, Q. Vu, and D. Agrawal, "Retrieving and organizing web pages by "information unit"," in *Proceedings of the 10th International Conference on World Wide Web*. New York: ACM, 2001, pp. 230–244.

[15] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee, "Self-organization and identification of web communities," *Computer*, vol. 35, no. 3, pp. 66–71, 2002.

[16] G. Greco, S. Greco, and E. Zumpano, "Web communities: Models and algorithms," *World Wide Web*, vol. 7, no. 1, pp. 59–82, 2004.

[17] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, no. 5, pp. 604–632, 1999.

[18] D. Katsaros, G. Pallis, K. Stamos, A. Vakali, A. Sidiropoulos, and Y. Manolopoulos, "CDNS content outsourcing via generalized communities," *IEEE Trans. Knowl. Data Eng.*, vol. 99, no. 1, 2008.

[19] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, p. 814, 2005.

[20] K. A. Smith and A. Ng, "Web page clustering using a self-organizing map of user navigation patterns," *Decis. Support Syst.*, vol. 35, no. 2, pp. 245–256, 2003.

[21] E. D. Giacomo, W. Didimo, L. Grilli, and G. Liotta, "Graph visualization techniques for web clustering engines," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 2, pp. 294–304, 2007.

[22] S. Yamada and N. Nagino, "Constructing a personal web map with anytime-control of web robots," in *Proceedings of the 4th IECIS International Conference on Cooperative Information Systems*. Washington, DC: IEEE Computer Society, 1999, p. 140.

[23] U. Brandes and S. Cornelsen, "Visual ranking of link structures," in *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, London, U.K Springer-Verlag, 2001, pp. 222–233.

[24] D. Meng, Y. Leung, T. Fung, and Z. Xu, "Nonlinear dimensionality reduction of data lying on the multicluster manifold," *IEEE Trans. Syst., Man, Cybern., Part B*, vol. 38, no. 4, pp. 1111–1122, Aug. 2008.

[25] X. Deyu, M. Yee, L. Tung, F. Zongben, and Xu Sch. "Incremental embedding and learning in the local discriminant subspace with application to face recognition," *IEEE Trans. Syst., Man, Cybern. C*, no. 5, vol. 40, pp. 580–591, Sep. 2010.

[26] M. Spivak, *A Comprehensive Introduction to Differential Geometry*, vol. 5, 2nd ed. ed. Wilmington, Del.: Publish or Perish Inc., 1979.

[27] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf, "Ranking on data manifolds," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[28] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *COLT* vol. 3120 (Lecture Notes in Computer Science Series), J. S. Taylor and Y. Singer, Eds. New York: Springer-Verlag, 2004, pp. 624–638.

[29] R. Lehoucq and D. C. Sorensen, "Deflation techniques for an implicitly re-started arnoldi iteration," *SIAM J. Matrix Anal. Appl.*, vol. 17, pp. 789–821, 1996.

[30] D. C. Sorensen, "Implicit application of polynomial filters in a *k*-step Arnoldi method," *SIAM J. Matrix Anal. Appl.*, vol. 13, pp. 357–385, Jan. 1992.

[31] U. Brandes, "A faster algorithm for betweenness centrality," *J. Math. Sociol.*, vol. 25, pp. 163–177, 2001.

Konstantinos Stamos received the M.Sc., and Ph.D. in computer science from the Aristotle University of Thessaloniki, Thessaloniki, Greece, in 2010.

His research interests include graph-related problems, kernel-methods, web-content management, and chess.

Nikolaos A. Laskaris received the M.Sc. degree in medical physics, in 1995 and Ph.D. degree in biomedical signal processing, in 1998, both from Patras University, Patrai, Greece.

He joined the Laboratory for Human Brain Dynamics in the Brain Science Institute of RIKEN in Japan, for four years, from 1999 to 2003, where he worked in the field of NeuroInformatics. He joined the Artificial Intelligence Information Analysis laboratory, in 2005, and currently serves as an Assistant Professor at Informatics Department in Aristotle University of Thessaloniki. He is a co-author of more than 30 journal papers and many conference ones. His current research interests include signal analysis, computational intelligence, soft computing, data mining, and nonlinear dynamics and their applications in biomedicine and neuroscience.

Dr. Laskaris received a grant from the Greek GSRT (ENTER-program) for working toward the development of neuromorphic signal processing techniques at ELLAB of Physics Department in Patras University, from 2003 to 2004.

Athena Vakali is an Associate Professor at the Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece. Her publication record is now at more than 100 research publications which have appeared in high quality journals, book chapters, and in scientific conferences, and she has also been a co-editor of the book "Web Data Management Practices: Emerging Techniques and Technologies" published by Idea Group Publishing. She has been a member of the editorial board of the *Computers and Electrical Engineering Journal* (Elsevier), and since March 2007 she has been the coordinator of the IEEE Technical Committee on Scalable Computing technical area of Content Management and Delivery Networks and she has scientifically leaded more than 15 European and national research projects. Her research interests include Web information systems such as Web data management (clustering techniques), content delivery on the Web, Web data clustering, Web caching, text mining, and multimedia data management.