Kernel Methods for Graphs: A Comprehensive Approach

Francesco Camastra and Alfredo Petrosino

Department of Applied Science, University of Naples Parthenope
Centro Direzionale Isola C4, 80143 Naples, Italy
{francesco.camastra, alfredo.petrosino}@uniparthenope.it

Abstract. The development of learning algorithms for structured data, i.e. data that cannot be represented by numerical vectors, is a relevant challenge in machine learning. Kernel Methods, which is a leading machine learning technology for vectorial data, recently tackled the structured data. In this paper we focus our attention on Kernel Methods that face up to data that can be represented by means of graphs, by providing an in-depth review through a comprehensive approach to the research hints and the main open problems in this area of research.

1 Introduction

Machine learning methods generally work on fixed-length vectorial data. However, in real world applications we can deal with data that cannot be represented by vectorial data but by structures. These types of data are called *structured data*. Examples of structured data are molecular formulae and images when represented by hierarchical structures. For instance, an image can be segmented into regions than may be recursively segmented into several sub-regions resulting in a hierarchical tree structure. Several supervised and unsupervised neural network models for structured data, such as trees and graphs, have been proposed [1,2]. Recently Kernel Methods [3] for structured data [4] have been developed. In this manuscript we will focus our attention on Kernel Methods that tackle structured data, i.e. data that can be represented by means of graphs. We shall provide a comprehensive framework that allows to apply the main Kernel Methods for graphs to various applications, like bioinformatics (e.g. molecular data mining, chemogenomics, etc.) and computer vision (e.g. 3D object structural recognition, spatio-temporal data mining, etc.). The paper is organized as follows: In Section 2 main facts on labelled graphs are recalled; the general issues of Kernel Methods are reviewed in Section 3; Product Graph Kernel is presented in Section 4; Marginalized Kernels and its extensions are described in Section 5; Cyclic Pattern Kernels for Graphs are discussed in Section 6, respectively; finally some research hints and the main open problems are discussed in Sections 7 and 8.

I. Lovrek, R.J. Howlett, and L.C. Jain (Eds.): KES 2008, Part II, LNAI 5178, pp. 662–669, 2008.

2 Labelled Graphs

A *graph* G is described by a finite set of vertices $V = \{v_1, \ldots, v_n\}$, a finite set of edges $E = \{e_1, \ldots, e_p) \subseteq V \times V$ and a function $\Psi : E \to V \times V$. A *labelled graph* is a tuple (V, E, Λ, λ) where $\Lambda = (l_1, \ldots, l_r)$ is a set of labels and $\lambda : \Lambda \to V \cup E$ is a function which assigns a label to each edge and vertex. In a *directed graph* the function Ψ maps each edge to the tuple composed of its initial and terminal node. A *loop* is an edge e in a directed graph such that $\psi(e) = (v, v)$. Two edges e_1 and e_2 are *parallel* iff $\Psi(e_1) = \Psi(e_2)$. A *walk* is a sequence of vertices $v_i \in V$ and $e_i \in E$ with $w = v_1, e_1, v_2, e_2, \ldots, v_n, e_n, v_{n+1}$ and $\Psi(e_i) = (v_i, v_{i+1})$. The *length* of the walk is given by the number n of edges in the sequence. A *path* is a walk such that $v_i \neq v_j \iff i \neq j$ and $e_i \neq e_j \iff i \neq j$. A *cycle* is a path with an edge e_{n+1} such that $\Psi(e_{n+1}) = (v_{n+1}, v_n)$. A graph is *connected* if a path exists between any couple of vertices. To describe the neighborhood of a vertex v in a graph G we need some further definitions. The set $\delta^+(v) = \{e \in E | \Psi(e) = (v, u)\}$ denotes the set whose elements come out from the edge v. Its cardinality $|\delta^+(v)|$ is called the *outdegree* of the vertex v. In similar way, the set $\delta^-(v) = \{e \in E | \Psi(e) = (u, v)\}$ defines the set whose elements come into the vertex v. The cardinality $|\delta^-(v)|$ is called the *indegree* of the vertex v. The maximal and the minimal indegree are $\Delta^-(G) = \max\{|\delta^-(v)|, v \in V\}$ and $\Delta^-(G) = \max\{|\delta^+(v)|, v \in V\}$, respectively. To represent a graph we use the *adjacency matrix* E, where E_{ij} of the matrix provides the number of edges between vertex v_i and v_j. The element E^n_{ij} of the n^{th} power of the adjacency matrix E^n gives the number of walks of length n from the vertex v_i and v_j. Now we introduce the concept of *labelled subgraph*. Let $G = (V, E, \Lambda, \lambda)$ and $G' = (V', E', \Lambda, \lambda')$ be two labelled graphs. G' is a labelled subgraph of G if $V' \subseteq V$, $E' \subseteq E$ and $\lambda'(x) = \lambda(x)$ for all $x \in V' \cup E'$. A maximal connected subgraph of the graph G is called a *connected component* of G. A vertex v of a graph G is called a *cut vertex* if the subgraph G' obtained from G removing v and all edges that comes into or comes out the vertex v has more connected components than G. A graph is *biconnected* if has no cut vertex. A biconnected maximal subgraph of a graph G is called a *biconnected component* of a graph G. The biconnected components of a graph are pairwise edge disjoint inducing a partition on the set E of the graph G. The partition implies that two edges are equivalent iff they belong to a common cycle. Therefore an edge of a graph belongs to a cycle iff the biconnected components of the graph have more than one edge. Edges that do not belong to cycles are called *bridges*. The set of the bridges of a graph G forms a subgraph of G denoted by $B(G)$.

3 Kernel Methods

Firstly we recall the definition of *Mercer kernel* (or *positive definite kernel*) [5].

Definition 1. *Let X be a nonempty set. A function $K : X \times X \to \mathbb{R}$ is called a* Mercer kernel *if it is* symmetric *and $\sum_{j=1}^{n} \sum_{k=1}^{n} c_j c_k K(x_j, x_k) \geq 0$ for all $n \geq 2$, $\{x_1, \ldots, x_n\} \subseteq X$ and $\{c_1, \ldots, c_n\} \subseteq \mathbb{R}$. Each Mercer kernel K can*

be represented as: $K(x,y) = \langle \Phi(x), \Phi(y) \rangle$ *where* $\langle \cdot, \cdot \rangle$ *is the inner product and* $\Phi : X \rightarrow \mathcal{F}$. $\mathcal{F}$ *is called the* Feature Space.

The simplest example of the Mercer kernel is the *inner product* $K(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$. A popular Mercer kernel is the *Gaussian* $K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{\sigma^2}}$, where $\sigma \in \mathbb{R}$. An important Mercer kernel for the rest of the paper is the *Dirac kernel* $\delta(x,y)$, that is 1 if $x = y$ and 0 otherwise. Mercer kernels can also be defined on structured data. An example of Mercer kernel on structured data is the *intersection kernel* between sets [3].

Definition 2. *Let* I_A *be the indicator function of a measurable set* A *(i.e* $I_A(x) = 1$ *if* $x \in A$, $I_A(x) = 0$ *otherwise). The indicator function defines the measure* μ *of the set* A *such as* $\mu(A) = \mu(I_A)$. *Given two sets* A_1 *and* A_2 *the* intersection kernel $K_\cap$ *is defined by:* $K_\cap(A_1, A_2) = \mu(A_1 \cap A_2)$.

Kernel Methods [3] are algorithms that implicitly perform, by replacing the inner product by an appropriate Mercer Kernel, a nonlinear mapping of the input data to a high dimensional Feature Space. The most popular Kernel Method is the Support Vector Machine (SVM) [3]. In Kernel Methods the data only occur under the form of an appropriate Mercer kernel input. Therefore they can also be used for structured data whenever it is possible to define a Mercer kernel on them. Hence the research on Kernel Methods for structured data results in designing appropriate Mercer kernels on them. The rest of the paper is focused on the description of the main Mercer kernels that it is possible to define on graphs.

4 Product Graph Kernel

In this section we review the *Product Graph Kernel* [6] that is based on the idea of counting the number of walks in product graphs. Product graphs can be defined in several ways. In this context we will only consider the direct product of two labelled graphs, that is defined as follows.

Definition 3. *The direct product of two labelled graphs* $G_1 = (V_1, E_1, \Lambda, \lambda_1)$ *and* $G_2 = (V_2, E_2, \Lambda, \lambda_2)$ *is denoted by* $G_1 \times G_2$. *The vertex set of the direct product is defined as:* $V(G_1 \times G_2) = \{(v_1, v_2) \in V_1 \times V_2 : \delta(\lambda_1(v_1), \lambda_2(v_2)) = 1\}$. *The edge set is defined as:*

$$\begin{aligned} E(G_1 \times G_2) = \{(e_1, e_2) \in E_1 \times E_2 : \exists (u_1, u_2), (v_1, v_2) \in V(G_1 \times G_2) \\ \wedge \Psi_1(e_1) = (u_1, v_1) \wedge \Psi_2(e_2) = (u_2, v_2) \\ \wedge (\delta(\lambda_1(e_1), \lambda_2(e_2)) = 1)\}, \end{aligned}$$

where $\delta(\cdot)$ *is the Dirac Kernel. Given an edge* $(e_1, e_2) \in E(G_1 \times G_2)$ *with* $\Psi_1(e_1) = (u_1, v_1)$ *and* $\Psi_2(e_2) = (u_2, v_2)$ *the value of* $\Psi_{G_1 \times G_2}$ *is:*

$$\Psi_{G_1 \times G_2}((e_1, e_2)) = ((u_1, u_2), (v_1, v_2)).$$

The label of the vertices and edges in graph $G_1 \times G_2$ *correspond to the labels in the factors. The graphs* G_1, G_2 *are called the factors of graph* $G_1 \times G_2$.

Now we can define the *product graph kernel*.

Definition 4. *Let G_1, G_2 be two labelled graphs, let $A_\times = A(G_1 \times G_2)$ be the adjacency matrix of their direct product and let $V_\times = V(G_1 \times G_2)$ be the vertex set of the direct product. With a sequence of weights $\lambda = \lambda_0, \lambda_1, \ldots$ (where $\lambda_i \in \mathbb{R}^+$) the product graph kernel is defined as $K_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[\sum_{n=0}^{\infty} \lambda_n A_\times^n\right]_{ij}$.*

The product graph kernel, that is a Mercer kernel [6], in conjunction with Gaussian Processes [7] has been used to determinate an optimal strategy for the play Tetris [8]. Now we show how the product graph kernel can be computed. Firstly, we discuss the exponential setting [9], i.e. we set $\lambda_i = \frac{\beta_i}{i!}$. The exponential of the square matrix A is defined as: $e^{\beta A} = \lim_{n\to\infty} \sum_{i=0}^{n} \frac{(\beta A)^i}{i!}$ with $\frac{\beta^0}{0!} = 1$ and $A^0 = \mathbb{I}$, where $\mathbb{I}$ denotes the identity matrix and β is a constant that must be set up. If we represent the matrix A in terms of a diagonal matrix D, i.e. $A = T^{-1}DT$, we obtain: $K_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} \left[T^{-1} e^{\beta D_\times} T\right]_{ij}$. The product direct kernel can also be computed by the geometric series. The geometric series $\sum_{i=0}^{\infty} \gamma^i$ converges iff $|\gamma| < 1$ and its limit is given by: $\sum_{i=1}^{\infty} \gamma^i = \frac{1}{1-\gamma}$. In similar way we can define the geometric series of a matrix A as follows: $\sum_{i=0}^{\infty} \gamma^i A^i$. If the matrix A is symmetric and $\gamma < 1$, the limit of the geometric series is given by: $\sum_{i=0}^{\infty} \gamma^i A^i = (\mathbb{I} - A)^{-1}$. Hence the product graph kernel by geometrical series is: $K_\times(G_1, G_2) = \sum_{i,j=1}^{|V_\times|} [(\mathbb{I} - A_\times)^{-1}]_{ij}$.

5 Marginalized Graph Kernel

In this section we will describe the *marginalized graph kernel* [10]. Given two labelled graphs G_1 and G_2, the idea behind marginalized graph kernel consists in comparing the label sequences generated by the two synchronized random walks of both graphs. Let $G = (V, E, \Lambda, \lambda)$ be a labelled graph, where $V = (v_1, \ldots, v_n)$ and $E = (e_1, \ldots, e_p)$ are the sets of vertices and edges, respectively. A random walk on the graph produces a *path*, that depends on the initial probability $P_s(v)$ distribution on V, a transition probability $P(v_i|v_{i-1})$ from the vertex v_{i-1} to the vertex v_i, and the probability $P_q(v_\ell)$ of stopping the random walk at vertex ℓ. If we do not use prior knowledge, $P_s(v)$ is a uniform distribution, $P(v_i|v_{i-1})$ is a uniform distribution over all adjacent vertices of the actual one, and $P_q(v_\ell)$ is a constant equal to $\frac{1}{n}$. From a given path obtained from the random walk, we can define the label sequence as an alternative sequence of vertex and edge labels, i.e. $h = (v_1, e_{1,2}, v_2, \ldots, v_{\ell-1}, e_{\ell-1,\ell}, v_\ell)$. The marginalized graph kernel between two labelled graphs $G_1 = (V_1, E_1, \Lambda, \lambda_1)$ and $G_2 = (V_2, E_2, \Lambda, \lambda_2)$ is defined as the expectation of the kernel between sequences $K(h_1, h_2)$ over all possible paths of all lengths:

$$K(G_1, G_2) = \sum_{h} \sum_{h'} K_\lambda(l(h_1), l(h_2)) p_1(h_1) p_2(h_2), \tag{1}$$

where p_1 and p_2 are probability distributions on the set of finite-length sequences of the vertices of the graphs G_1 and G_2, respectively and $K_\lambda(\cdot)$ is an appropriate kernel between label sequences, i.e. $K_\lambda : \Lambda^\star \times \Lambda^\star \to \mathbb{R}$, where $\Lambda^\star$ denotes the

set of finite-length sequence of labels of Λ. The marginalized graph kernel can be interpreted as a marginalized kernel [11], motivating the name and guaranteeing that is a Mercer kernel. Kashima et al. [10] have investigated a particular case of the marginalized graph kernel when the K_λ is the Dirac kernel and the probability p of the equation (1) can be expressed as:

$$p(v_1, \ldots, v_n) = p_s(v_1) \prod_{i=2}^{n} p_t(v_i|v_{i-1}). \tag{2}$$

To guarantee that p is a probability, i.e. $\sum_{v \in V} p(v) = 1$, it is necessary that some p_s and p_t fulfill some constraints. If we choose

- a stopping probability p_q (with $0 < p_q(v) < 1$) for each vertex v,
- an initial probability distribution p_0 (such that $\sum_{v \in V} p_0(v) = 1$),
- a transition probability p_a on $V \times V$ (with $\sum_{v \in V} p_a(u|v) = 1$) such that if $p_a(v|u) > 0$ then $(u, v) \in E$, i.e. $p_a(v|u)$ is positive only when an edge exists,
- and for any $u, v \in V^2$ $p_s(v) = p_0(v)p_q(v)$ and $p_t(u|v) = \frac{1-p_q(v)}{p_q(v)} p_a(u|v)p_q(u)$;

then p, defined in (2), is a probability distribution that corresponds to a random walk on the graph G with an initial, transition and stopping probability defined by p_0, p_a and p_s, respectively.

Marginalized graph kernel has two main drawbacks. The former consists in its computational burden. Therefore its usage is not advisable when the data set contains several thousands of graphs, as it generally happens in bioinformatics applications The latter is represented by the strategy implemented in the marginalized graph kernel. The kernel is based on the search of common paths between graphs which may be too simple to detect common patterns of interest between graphs. To overcome these limitations, extensions of marginalized graph kernel [12] have been proposed. They increase the degree of the specificity of the label vertices of the graph, taking into account contextual information about the vertices.

6 Cyclic Pattern Kernels for Graphs

In this section we describe the *cyclic pattern kernels for graphs* (CPKs) [13], that are based on the intersection kernel between sets. To use the intersection kernel on graphs, each graph G is represented by a set of cycles and tree patterns of G. Let $G = (V, E, \Sigma, \lambda)$ be a graph and $C = \{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_0\}$ be a sequence of $k-1$ edges that forms a cycle in G. The canonical representation of C is defined as follows. If we denote by $\rho(s)$ the set of cyclic permutations of a sequence s and its reverse, the *canonical representation* $\pi(C)$ [14] is defined by:

$$\pi(C) = \min(\sigma(w) : w \in \rho(v_0 v_1 \ldots v_{k-1}))$$

where for $w = w_0 w_1 \ldots w_{k-1}$

$$\sigma(w) = \lambda(w_0)\lambda(\{w_0, w_1\})\lambda(w_1) \ldots \lambda(w_{k-1})\lambda(\{w_{k-1}, w_0\}).$$

The set of *cyclic patterns* $\mathcal{C}(G)$ of a graph G is defined by:

$$\mathcal{C}(G) = \{\pi(C) : C \in S(G)\}$$

where $S(G)$ denote the set of cycles of G. More information can be added to the kernel considering the set of the bridges of the graph. The set of tree patterns $T(G)$ assigned to graph G is defined by:

$$T(G) = \{\pi(T) : T \text{ is a connected component of } B(G)\}$$

where $B(G)$ is the set of the bridges of the graph G. Assuming that $C(G)$ and $T(G)$ are disjoint, the cyclic pattern kernel is defined as follows:

Definition 5. *Let G be a graph, the set $F(G)$ is defined as $C(G) \cap T(G)$. The cyclic pattern kernels between two graphs G_1 and G_2 is given by:*

$$K_{CP}(G_1, G_2) = |\mathcal{C}(G_1) \cap \mathcal{C}(G_2)| + |\mathcal{T}(G_1) \cap \mathcal{T}(G_2)|.$$

CPKs can be computed by the algorithm [13], whose input and output are respectively a graph G and the set of cycles S, described below:

1. S=B= $\oslash$
2. Compute the set Γ formed by the biconnected components of G
3. for $G' \in \Gamma$ do
4. if G' has one edge e then $B = B \cup e$ else $S = S \cup C(G')$
5. $S = S \cup \{\pi(t) : t$ is a connected component of $B.\}$
6. return S

The biconnected components can be effectively computed by a depth-first search algorithm in a linear time [15]. Nevertheless, the computation of CPKs is *NP-hard* [13] mining, in this way, their real utility. To overcome the problem of intractability, Horvath et al. [13] proposed a restriction of CPKs. They observed that the simple cycles of a graph G can be computed with polynomial complexity [16]. Besides, for a given graph G and a $k \geq 0$, it can decide [16] if the number of cycles in G is bounded by k. Using these facts, Horvath et al. [13] proposed to restrict the kernel to the case when the number of simple cycles is bounded by a constant for every graph in the database. This assumption is quite reasonable in real-world graph databases. They proposed a variant of the previous algorithm [13] that computes the cyclic pattern kernels only for graphs that have at most k simple cycles. The variant, that takes as input a graph G with n vertices and m edges and an integer $k \in \mathbb{N}$ and produces as output the set of cycles S, has the following steps:

1. $K = 0$
2. $S = B = \oslash$
3. Compute the set Γ formed by the biconnected components of G
4. for $G' \in \Gamma$ do
5. if G' has more than one edge e then
6. $X =$READ-TARJAN$(G', k - K + 1)$

7. if $|X|$ is equal to $k - K + 1$ then return $\oslash$
8. else
9. $S = S \cup \{p(C) : C \in X\}$
10. $K = K + |X|$
11. else $B = B \cup e$
12. $S = S \cup \{\pi(t) : t$ is a connected component of $B.\}$
13. return S

where READ-TARJAN is a procedure that implements the Read-Tarjan's algorithm that allows to decide if the number of cycles in a graph G is bounded by k. SVMs with cyclic pattern kernels have been used effectively [13] on NCI-HIV database, a popular benchmark for the classification of chemical compounds.

7 The Kernel Approach

In this section we summarize the main peculiarities of Kernel Methods and we discuss the possible line of research. The main advantage of the kernel approach is its flexibility. The kernel methods approach for graphs results in designing an appropriate Mercer kernels on them. Computed the kernel, it can be used, without requiring further adjustments, in usual Kernel Methods for vectorial data. This allows to use the designed kernel either in supervised (e.g. SVMs, Gaussian Processes [7]) or in unsupervised Kernel Methods (e.g. *Kernel Clustering Methods* [17]). Although Kernel Methods for Graphs have been only investigated in their supervised form, Kernel Clustering Methods for graphs offer several potential advantages with respect to the unsupervised neural learning algorithms for graphs such as the SOM for structured data [2]. SOM can only produce piecewise linear separations among clusters. On the contrary, Kernel Clustering methods produce nonlinear separations allowing to tackle more highly nonlinear data sets. Therefore a challenging goal for Kernel Methods is represented by developing and validating Kernel Clustering Methods for graphs. Kernel Clustering Methods are not the unique algorithms able to produce nonlinear separations among clusters. Nonlinear separations can also be obtained by *Spectral Clustering Methods* [17]. The development of spectral clustering methods for graphs and, in general, for structured data remains an open problem.

8 Conclusion

The development of effective learning algorithms for structured data, i.e. data that cannot be represented by vectors, is one of the most challenging topic for machine learning. Recently Kernel Methods have tackled the structured data getting new force on the research about structured data. In this paper we have focused our attention on Kernel Methods that cope with data that can be represented by graphs. The aim of this paper has been to provide a comprehensive approach to discuss about the main Kernel Methods for graphs. One of the main challenging goals of Kernel Methods of graphs consists in extending the validation of these techniques, widely applied in bioinformatics and chemioinformatics, to other applicative areas such as computer vision applications.

References

1. Frasconi, P., Gori, M., Sperduti, A.: A general framework for adaptive processing of data sequences. IEEE transactions on Neural Networks 9(5), 768–786 (1997)
2. Hagenbuchner, M., Sperduti, A., Tsoi, A.: A self-organizing map for adaptive processing of structured data. IEEE transactions on Neural Networks 14(23), 491–505 (2003)
3. Shawe-Taylor, J., Cristianini, N.: Kernels Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
4. Gärtner, T.: A survey of kernels for structured data. SIGKDD Explorations 5(1), 49–58 (2003)
5. Berg, C., Christensen, J., Ressel, P.: Harmonic analysis on semigroups. Springer, New York (1984)
6. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: Hardness results and efficient alternatives. In: Proceedings of 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, pp. 129–143. IEEE Press, Los Alamitos (2003)
7. Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. MIT Press, Cambridge (2006)
8. Gärtner, T., Driessens, K., Ramon, J.: Graph kernels and gaussian processes for relational reinforcement learning. In: Horváth, T., Yamamoto, A. (eds.) ILP 2003. LNCS (LNAI), vol. 2835, pp. 146–163. Springer, Heidelberg (2003)
9. Gärtner, T., Lloyd, J., Flach, P.: Kernels for structured data. In: Matwin, S., Sammut, C. (eds.) ILP 2002. LNCS (LNAI), vol. 2583, pp. 66–83. Springer, Heidelberg (2003)
10. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proceedings of 10th International Conference on Machine Learning, pp. 321–328. IEEE Press, Los Alamitos (2003)
11. Tsuda, K., Kin, T., Asai, K.: Marginalized kernels for biological sequences. Bioinformatics 18, S268–S275 (2002)
12. Mahé, P., Ueda, N., Akutsu, T., Perret, J.L., Vert, J.P.: Extensions of marginalized graph kernels. In: Proceedings of 21st International Conference on Machine Learning (ICML 2004), pp. 552–559. IEEE Press, Los Alamitos (2004)
13. Horvath, T., Gärtner, T., Flach, P., Wrobel, S.: Cyclic pattern kernels for predictive graph mining. In: Proceedings of KDD 2004, pp. 158–167. ACM Press, New York (2004)
14. Zaki, M.: Efficiently mining frequent trees in a forest. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 71–80. ACM Press, New York (2002)
15. Tarjan, R.: Depth-first search and linear graphs algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)
16. Read, R., Tarjan, R.: Bounds on backtrack algorithms for listing cycles, paths and spanning trees. Networks 5(3), 237–252 (1975)
17. Filippone, M., Camastra, F., Masulli, F., Rovetta, S.: A survey of kernel and spectral method for clustering. Pattern Recognition 41(1), 174–192 (2008)