الجمهورية الجزائرية الديمقراطية الشعبية

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

وزارة التعليم العالي و البحث العلمي

Ministry of Higher Education and Scientific Research

جامعة غرداية

University of Ghardaia

كلية العلوم و التكنولوجيا

Faculty of Science and Technology

قسم الرياضيات و الإعلام الآلي

Department of Mathematics and Computer Science

MEMOIR

Presented for the degree of **Master**

In: Computer Science **Specialty**: Intelligence System for Knowledge Extraction

By: MOUAD Chenini and ABDERRAHMANE Sebgag

Theme

TREE KERNEL COMPUTATION BASED ON BINARIZATION

Jury members

M. Adjila Abderrahmane	MAA	Univ. Ghardaia	Examiner
M. Bellouar Slimane	Doctor	Univ. Ghardaia	Supervisor
M. Djelloul Ziadi	Professeur	Univ. Rouen France	Supervisor
M. Ouled naoui Slimane	Doctor	Univ. Ghardaia	President
M. Kerrache Chaker Abdelaziz	Doctor	Univ. Ghardaia	Examiner

College year 2017/2018

" علي بن أبي طالب رضي الله عنه "

كل إناء يضيق بما جعل فيه إلا وعاء العلم فإنه يتسع –

Abstract

Machine learning use intelligent methods of data analysis from massive collections and under the pressure of applications, we are confronted with problems in which the data structure carries essential information. Linear methods of data analysis and learning were among the first to be developed. They have also been intensively studied, in particular many applications are data that can be represented in structured form (sequences, trees, graphs,. . .).
The kernel methods make it possible to find nonlinear decision functions. However, the advent of kernel methods has lead to research renewal as these methods are generic and can be applied to a wide variety of domains when we are able to conceive a kernel function.
Tree kernel has been proposed for applications to machine learning in natural language processing or for the calculation of XML documents similarity. Our aim is to investigate the tree kernels proposed by (Moschitti, 2006a) and his algorithm for the evaluation of *ST* and *SST* kernels and to study the effect of these kernels on the similarity between the two analysed trees, We evaluated the impact of tree kernels in k-ary tree and its equivalent binary tree.
We carried out a comparative study between tree kernel in k-ary tree and binary tree equivalent to it. the Comparison included similarity and running time. We concluded that proposed method perfect than Knuth method in some cases.

Key words :

Tree binarization, Kernel methods, tree kernel, subtree kernel, subset tree kernel, binarization.

Résumé

Tout d'abord l'apprentissage automatique exploite les méthodes intelligentes d'analyse de données à partir d'une grande collection. De même que les méthodes habituelles sont des méthodes linéaires. En outre divers applications possède des données qui peuvent être illustré sous forme structurée (séquences, arbres, graphes,. . .).
Les méthodes à noyaux permettent de trouver des fonctions de décision non linéaires. Cependant, l'avènement des méthodes à noyaux a conduit à un renouvellement des recherches dans la mesure où ces méthodes sont génériques et peuvent sappliquer à une grande variété de domaines lors que lon est capable de concevoir une fonction noyau.
Les noyaux d'arbres ont été proposés pour des applications à l'apprentissage automatique en langue naturelle ou pour le calcul de la similarité des documents XML.
Notre objectif est d'étudier les noyaux d'arbres proposés par (Moschitti, 2006a) et son algorithme pour l'évaluation des noyaux *ST* et *SST* et pour étudier l'éffet de ces noyaux sur la similitude entre deux arbres, nous avons évalué l'impact de noyaux d'arbres dans l'arbre *n*-aire et son arbre binaire équivalent.
Nous avons réalisé une étude comparative entre un noyau d'arbre dans un arbre *n*-aire et un arbre binaire équivalent. La comparaison incluait la similarité et le temps d'exécution. Nous avons conclu que la méthode de binarisation proposée parfaite que la méthode de knuth dans certains cas.

Mot clé :

Noyaus d'arbres, subtree, subset tree, arbre binaire, binarisation des arbres.

ملخص

يستخدم التعلم الآلي أساليب ذكية لتحليل البيانات من مجموعات ضخمة وتحت ضغط من التطبيقات ، ونحن نواجه مشاكل في هيكلة البيانات ذات المعلومات المهمة والأساسية. فطرق االتعلم الآلي التي تحلل و تعالج البيانات الخطية من بين أول الطرق التي تم تطويرها. كما تم دراستها دراسة مكثفة ،وهي غالباً ما تعالج مستندات مسطحة، الا انه عمليا نجد العديد من البيانات يمكن تمثيلها على شكل مركب (سلاسل أشجار مخططات،...). فقد ظهرت الأنوية لتعالج هذا النوع من البيانات المركبة مرتكزة على مبرهنات وأسس. وقد أدت أساليب النواة إلى تجديد البحث لأن هذه الأساليب عامة ويمكن تطبيقها على مجموعة واسعة من المجالات عندما نكون قادرين على تصور وظيفة النواة.
هدفنا هو دراسة نواة الشجرة ($Tree\ kernel$) المقترحة بواسطة ($Moschitti, 2006a$) من من هذا المنظور قمنا باستهداف نواة ST و SST ودراسة تأثير هذه النوى على حساب التشابه بين شجرتين التي يتم تحليلها، حيث اقترحنا خوارزمية لتحويل حساب النواة من شجرة عامة إلى شجرة ثنائية، ودراسة خصائص هذا التحويل من حيت المحافظة على البنية و نتيجة التشابة المتحصل عليه أجرينا مقارنة بين نواة الأشجار في شجرة ($k - ary$) والشجرة الثنائية ($Binary\ tree$) المكافئة لها. تضمنت المقارنة التشابه و وقت التشغيل . لقد استنتجنا أن خوارزمينا للتحويل من شجرة عامة إلى شجرة ثنائية أفضل من خوارزمية التحويل ل $Knuth$ في بعض الحالات

كلمات مفتاحية

نوى الأشجار، الشجرة الثنائية، طرق النواة، نواة الشجرة الفرعية

Dedicated

To

My Mother

A strong and gentle soul who taught me trust in Allah, believe in hard work and that so much could be done with little.

My Father

For earning an honest living for us and for supporting and encouraging me to believe in myself.

My Sisters and Brothers

The greatest gift my parents have give me, who taught me the powerful of hope, I am proud to be your brother

My Uncle

For being my professor during my educational career

My Colleagues

Thank all of my colleagues of the Computer Science Department, university of Ghardaia for their helps.

My Friends

Yakoub, Abdelhak, Abdennour, Abdelallah, Nadjet, Khaoula, I don't need word or express, i don't need to ask for a smile, or a hand to hold me .... All I need is to be your friend, forever!

sincerely

Abderrahmane.

It is with our deepest gratitude and warmest
affection that we dedicate this thesis

To my tender Mother Fatima: You represent for me the source of tenderness and the example of dedication that has not ceased to encourage me. You have done more than a mother can do to make her children follow the right path in their lives and their studies.

To my dear Father Lakhdar: No dedication can express the love, esteem, dedication and respect I always have for you. Nothing in the world is worth the effort provided day and night for my education and my well being. This work and the fruit of your sacrifices that you made for my education and training along these years.

To my dear brothers: Nabil,
Soufiane, Mohamed and Moussa.
To my sisters: Abir, Serine, Aicha and Zahira.
To my dearest friends: Walid and Yacine.

To our Professor Prof. Ziadi Djelloul
who has been a constant source
of Knowledge and inspiration.

To all members of my promotion.
To all my teachers since my first years of studies.
To all those who feel dear to me
and whom I have failed to mention.

Mouad Chenini

Acknowledgement

“words fly away, writings remain”

In the name of "ALLAH", The most beneficent and merciful who gave as strength and knowledge to complete this thesis.

First of all, we would like to express our deepest sense of Gratitude to our supervisors Professor **Ziadi Djelloul** and Doctor **Bellaouar Slimane** who offered their continuous advice and encouragement throughout the course of this thesis. We thank them for the systematic guidance and great effort they put into training us in the scientific field. Like a charm!

We are deeply grateful to all members of the jury for agreeing to read the manuscript and to participate in the defense of this thesis.

We thank all the teachers they taught us in the five past years for the vast amount of information.

For all those who participated in the development of this work.

Contents

List of Figures

List of Algorithms

1

Introduction

"If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas."
– George Bernard Shaw

1.1 Context

THE last decade is marked by the explosion of data, the big data phenomenon, especially those digital texts in tree structure which are considered among the most important types of data.

This informational flood has made the operations of analysis and manual classification of these resources a delicate task. This captured the attention of computer science community and therefore several machine learning algorithms and technical data representations have been developed.

Machine learning aims to provide automatic tools for imitate human ability to improve one's behavior with experience. It is a growing field, that is used for a wide range of applications: natural language processing, bioinformatics, medical diagnosis, pattern recognition, search engines, fraud detection, analysis stock markets, software engineering, adaptive web, robotics, games,...

However, classical methods of machine learning are linear methods. They are often very well adapted to flat documents represented by vector models. In practice, many applications have data that can be represented naturally under a structured form. As an example, XML documents are naturally represented by trees, in the natural language processing, each sentence

can be represented by a syntactic tree. In bioinformatics, proteins can be represented as amino acid sequences and genomic DNA as a nucleotide sequence. This problem of representing structured data can be approached by changing the representation data through non-linear functions while keeping the regularities and the dependencies inherent in the data. The kernel methods make possible this trick by projecting the data into a high dimensional feature space while avoiding the explicit computation of this projection.
Kernel methods look for a linear relationship in the feature space. Thus, the input data can be compared through the inner products of their representations in the feature space. However, kernel methods avoid direct access to this space while it is possible to replace the inner product with a positive semi-definite kernel function that computes the similarity between two elements directly in the input space. The advantage of using the kernel functions, that is possible to use feature spaces of high dimensions (even infinite), with a complexity independent on the feature space size, but that depends only on the complexity of the kernel function herself.

1.2 Motivations

An analysis of the literature on the tree kernels concludes that the most of these kernels belong to a family called convolution kernels (Haussler, 1999). This family introduces a method for constructing kernel on sets whose elements are discrete structures like sequences, trees and graphs. Discrete structures are recursive objects that can be decomposed into sub-objects until reaching an atomic unit. Unfortunately, the complexity of the convolution kernels is very high and does not allow the computation of the kernel function on very complex structures. This can prevent their applications in real scenarios.
Our interest in this thesis is the tree kernels for "non-linear" documents where the text can be formatted in structure such as trees. Indeed, documents are more and more organized in information fields, especially in XML format. This structure does not only make it possible to have rich (and heterogeneous) information, but also allows the automatic processing systems to manage these data more easily (especially for data collection).

Because of the effectiveness of the binary tree structure in various systems and applications, binary tree have been devloped in many area and it is used in various fields, such as in binary searche tree (BST), the cost of insertion, remove, lookup is $O(\log N)$. We want to find a relationship and a transformation to calculate the kernel from the K-ary tree to an equivalent binary tree.
The pulp is to study the effect of different binarization methods on the calculation of tree kernels.

1.3 Organization

This manuscript is organized in three main chapters:

- **Chapter 2** introduces some preliminaries on structured data (sequence, graph and tree), and kernel methods.
- **Chapter 3** gives overview of tree kernel and some related works, section 3.1 and 3.2 it focuses on the presentation of the subtree (ST) and subset tree (SST) kernel.
- **Chapter 4** is reserved for our contribution by presenting our proposed method of the binarization. Moreover we conduct experiments to measure the impact of tree binarization on tree kernel.

Preliminaries

" Writing the perfect paper is a lot like a military operation. It takes discipline, foresight, research, strategy, and, if done right, ends in total victory"

– Ryan Holiday

THIS chapter introduces the basic concepts necessary to understand the work presented in the next chapters. first, we give an overview of the discipline of structured data as well as the theoretical foundations relating to it. Then we highlight the kernel methods, and the substance of our research.

2.1 Structured data

Due to the accelerated appearance of several forms of data, it became necessary to organize these data to exploit and extract information from them. There are three type of data classification perspective: structured data, unstructured data and semi-structured data. Structured data that is formed by a simpler combination components into more complex elements often involve the frequent use of the simplest objects of the same type. Usually, it is easier to compare simpler components with basic function or using an inductive argument on the structure of objects. For this reason, a lot of researchs have been devoted to them in recent years. Examples of structured data include familiar examples, strings and sequences, but also complex object like trees, images and graphs (Shawe-Taylor and Cristianini, 2004; Arimura, 2008).

"Universal" data models that allow to represent structures.

- **Irregular**: we can compare data in different formats (eg. a character sequences with n-tuple).
- **Implied** data and structures (grammar, schema) are mixed.

- **Partial**: coexistence of structured and unstructured data (eg. XML, graphs / labelled trees. The data are heterogeneous level of structure and semantics).

In the next sections of this chapter we focus on sequences and more generally on structured data (trees, graphs).

2.2 Sequences

The sequences are considered as part of the structured data because a sequence can be decomposed into subpart, so the sequences have the property of the structured data (Aseervatham, 2007).
For the sequence representation of symbols, the sequential aspect is important. A symbol means a letter on an alphabet. It can be a character, a syllable, a word or a concept. This representation is adopted in several areas: in bioinformatics applications, where the proteins can be represented as an amino acid sequence, genomic DNA as nucleotide sequences. In the field of the natural language processing, a document can be represented as a sequence of characters, words, sentences or paragraphs.

2.3 Trees

This section is dedicated to trees, one of the most important algorithmic concepts of computing. Trees are used to represent a set of hierarchically structured data. Several distinct notions are hidden in this terminology (graphs, trees, binary trees, ...) These definitions are specified this section.
To present the trees in a homogeneous way, some terms borrowed from the graphs are useful. We will present the graphs, then successively, trees and thier terminologies.

2.3.1 Graphs

Formally, a graph is defined by a couple $G = (S,A)$ such as S is a finite set of nodes and A is a set of edges $(s_i, s_j) \in S^2$ (Solnon, 2008)

A graph can be directed or not:

- In a *directed graph*, couples $(s_i, s_j) \in A$ are oriented, i.e (s_i, s_j) is an orderly couple, where s_i is the initial node, and s_j is the terminal node. A couple (s_i, s_j) is called an arc(or edge), and it can be represented graphically by $s_i \longrightarrow s_j$. Figure 2.1 show an example of directed graph.

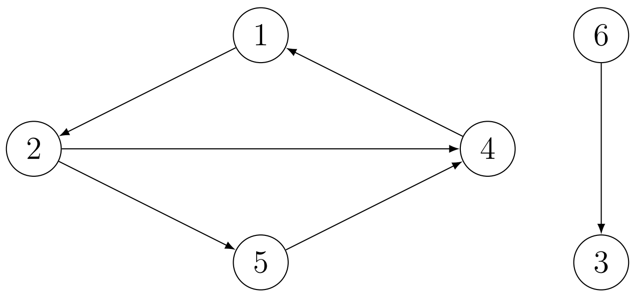


Figure 2.1: Example of directed graph

Example 2.3.1. The figure 2.1 represents the directed graph, $G(S,A)$ with $S = \{1,2,3,4,5,6\}$ and $A = \{(1,2),(2,4),(2,5),(4,1),(5,4),(6,3)\}$

- In an *undirected graph*, couples $(s_i,s_j) \in A$, are not oriented, i.e (s_i,s_j) is equivalent to(s_j,s_i). A pair (s_i,s_j) is called an arity. Figure 2.2 show an undirected graph.

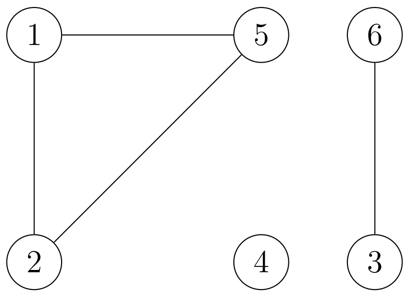


Figure 2.2: Example of undirected graph

Example 2.3.2. The figure 2.2 represents the undirected graph, $G(S,A)$ with $S = \{1,2,3,4,5,6\}$ and $A = \{(1,2),(1,5),(5,2),(6,3)\}$

In an undirected graph, a *cycle* is a sequence of consecutive edges (single chain) whose two vertices ends are identical.

A graph is without cycle or *acyclic*, if it does not have cycles. It is directed if the set **E** of arcs, is constituted *couple* of nodes. A couple being ordered unlike a pair.

The *degree* $d(v)$ of a node v is the number of edges incident to this node. In an directed graph, $d^-(v)$ is the number of incoming arcs at v, while $d^+(v)$ is the number of outgoing arcs of v.

2.3.2 Tree

A tree T is particular acyclic oriented graph in which each nodes except one has in-degree one. The node with in-degree 0 is known as the root $r(T)$ of the tree. Nodes v with out-degree $d^+(v) = 0$ are known as *leaf* nodes, while those with non-zero out-degree are *internal nodes.* The nodes to which an internal node v is connected are known as its *children*, while v is their *parent*. Two children of the same parent are said to be *siblings* (Shawe-Taylor and Cristianini, 2004). We give some terminologies inherent to trees by describing Figure 2.3

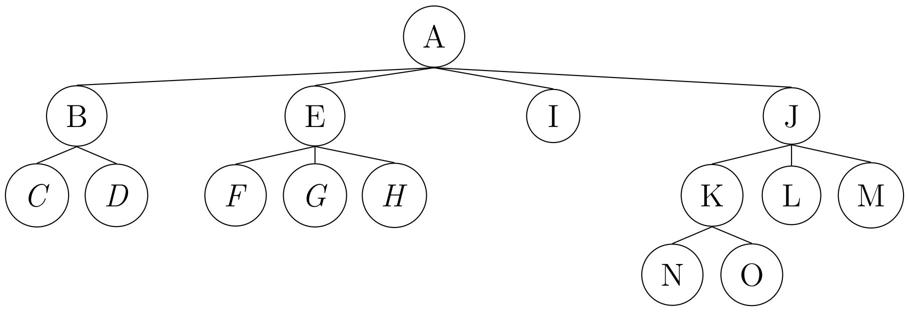


Figure 2.3: Example of a tree T

- A is the root of tree T.
- B is parent of C and D.
- B, E, I, J are children of A, K, L, M are children of J.
- C and D are siblings, F,G and H are siblings.
- A, B, E, I, J, K, L and M are internal nodes.
- C, D, F, G, H, I, N, O, L and M are leaves.
- The production of node is represent by its label follows by each label of its children, for example the production of $A \rightarrow B\ E\ I\ J$ is given by $(A(B,E,I,J))$.

Definition 2.3.1. in an k-*ary tree* the out-degree of any node is bounded by k, i.e. it can never be greater than k. If $k = 2$ the tree is known as a *binary tree*. A binary tree consist of node linked with two binary trees which are the right subtree and the left subtree.

2.3.3 Subtrees

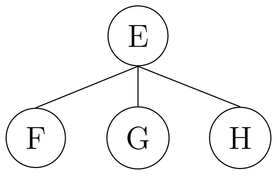


Figure 2.4: a subtree of tree T

Figure 2.4 is one of the subtrees of T.
A subtree is a portion of a tree that can be viewed as a complete tree in itself. Any node in a tree T, together with all the nodes below, comprise a subtree of T. The subtree corresponding to the root node is the entire tree. The subtree corresponding to any other node is called a

proper subtree.

We can distinguish several types of subtrees (Shawe-Taylor and Cristianini, 2004):

- A *SubTree (ST)* or *complete subtree* of a tree T at a node v is the tree obtained by taking all nodes and arcs accessible from v. Figure 2.5 illustrates a syntactic parse tree with some of its subtrees.

- A *co-rooted subtree* of a tree T is obtained by subtracting a sequence under complete trees and replacing them with their roots. Therefore, if a node v is included in a co-rooted subtree, then so are all of its siblings..

- A *SubSet Tree (SST)* or *general subtree* of a T tree is any rooted subtree of a complete subtree. Figure 2.6 shows a tree with some of its general subtrees.

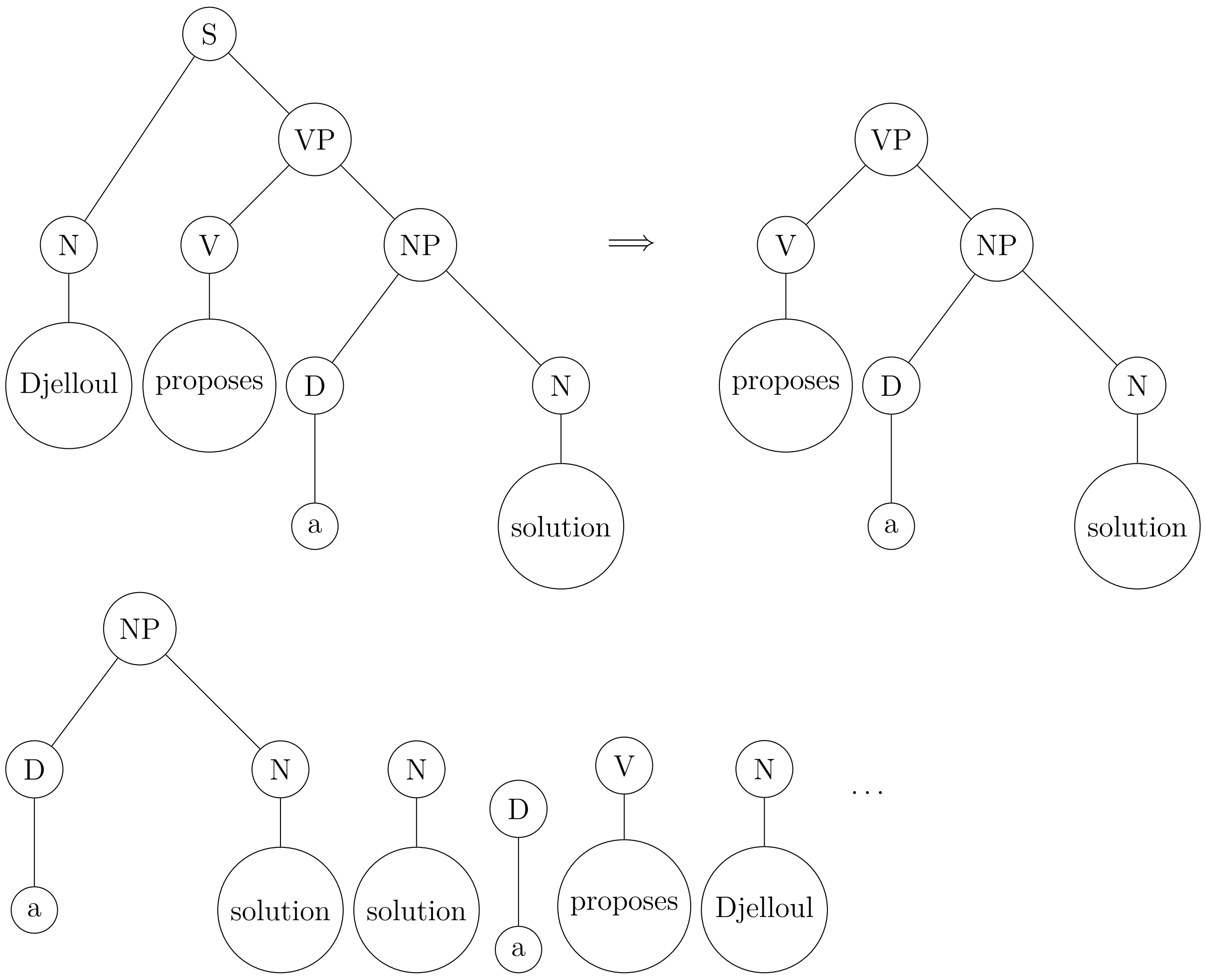


Figure 2.5: A syntactic parse tree with some of its subtrees.

2.4 Kernels

Linear methods of data analysis and learning were among the first to be developed. They have also been intensively studied, in particular because they lend themselves well to mathematical

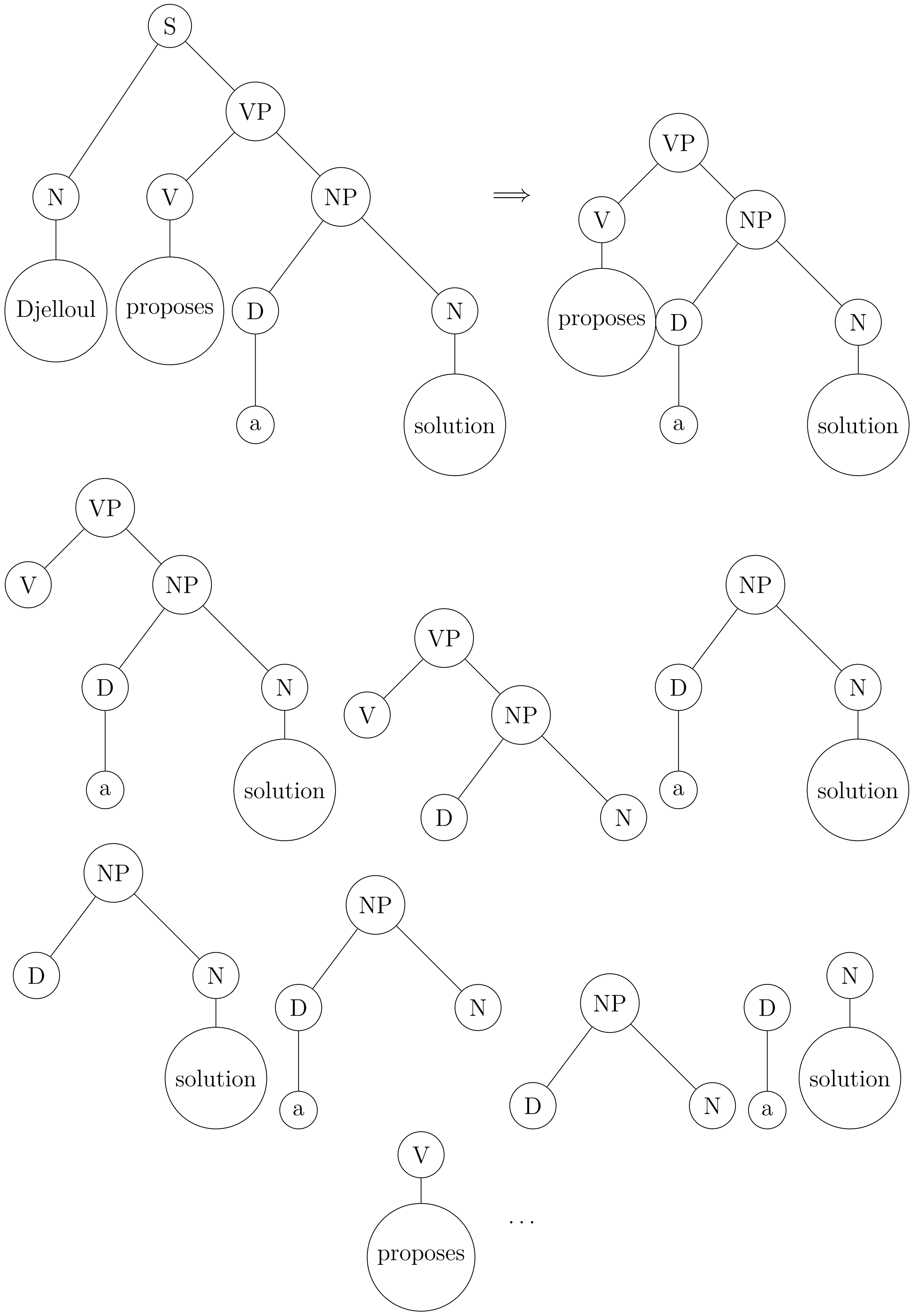


Figure 2.6: A tree with some of its subset trees (*SSTs*).

analysis. However, many applications require non-linear models to report dependencies and underlying patterns in the data. The kernel methods make it possible to find non-linear decision functions, while relying fundamentally on linear methods. A kernel function corresponds to a scalar product in a data feature space, often of large size. In this space, it is not necessary to manipulate the data explicitly, linear methods can be used to find linear regularities, corresponding to non-linear regularities in the original space. Thanks to the use of kernel functions, it becomes possible to have the best of two worlds: use simple and rigorously guaranteed techniques, and treat non-linear problems. That's why these methods have become very popular recently.
Trees and graphs naturally fit into composite and structured objects. The comparison of such objects has been discussed for a long time, especially in what was called the "recognition of structural forms". However, the advent of kernel methods has lead to renewed research as these methods are generic and can be applied to a wide variety of domains when one is able to conceive a kernel function. That is an appropriate measure of similarity.

2.5 Convolution kernels

The convolution principle (Haussler, 1999) introduces a method for constructing kernels on sets whose elements are discrete structures such as words, trees, or graphs. These discrete structures are recursive objects that can be decomposed into sub-objects until they reach an atomic unit.

The idea of the convolution kernel is to adopt a recursive approach in the similarity calculation. Formally, let x be a discrete structure belonging to a set X of the same type. The structure x can be decomposed into sub-objects $(x_1,\ldots,x_D)$ where x_d belongs to the set $\mathbf{X_d}$ for $1 \leq d \leq D$ and D is a positive integer.
We can represent the relation "$x_1,\ldots,x_D$ are parts of x" by the relation $R : \vec{x} = (x_1,\ldots,x_D) \longrightarrow X$ with $R(\vec{x};x)$ true if and only if $x_1,\ldots,x_D$ are parts of x. Let $R^{-1} = \{x : R(\vec{x},x)\}$ that returns for x all of the possible decompositions.

Suppose that $x,y \in X$ and that $\vec{x} = (x_1,\ldots,x_D)$ and $\vec{y} = (y_1,\ldots,y_D)$ are respectively decompositions of x and y. We also assume that for each $1 \leq d \leq D$, we have a k_d kernel on X_d that we can use to measure the similarity $k_d(x_d,y_d)$ between the part x_d and the part y_d. So let's define the convolution kernel (R-convolution) for two elements x,y of X as follows:

$$k(x,y) = \sum_{\vec{x}\in R^{-1}(x),\vec{y}\in R^{-1}(y)} \prod_{d=1}^{D} k_d(x_d,y_d). \tag{2.1}$$

It's easy to show that if k_d are valid kernels, the convolution kernel k is valid. If k_d is valid, then its Gram matrix is positive semi-definite. The same is true for the product and the sum of the positive semi-definite matrices. A kernel is valid if is symmetric and positive semi-definite. Moreover, recursive definitions for calculating a convolution kernel requires a significant calculation time. Therefore the use of iterative programming techniques and sophisticated data structures is necessary.

2.6 Kernel methods

A classic approach to deal with non-standard linear approach is to project all the data into one feature space that preserves the inherent grouping of data while simplifying the associated structure.
However, as this feature space can be very large, working directly with projected variables is generally considered an unrealistic option. The kernel trick, which we describe here, makes it possible to free oneself from it.

The kernel trick allows to transform searching for non-linear regularities into linear regularities via the (virtual) projection of the input space into a feature space. Formally, for a point x of the space x, we consider a function $\mathbf{\Phi}$ to a feature space $\mathbf{F}$ with inner product: (Shawe-Taylor and Cristianini, 2004)

$$\mathbf{\Phi} : x \mapsto \mathbf{\Phi}(x) \in \mathbf{F} \subseteq \mathbf{R}^N \tag{2.2}$$

Learning algorithms do not need to know the coordinates projections in the feature space $\mathbf{F}$. On the other hand they must calculate the inner products of images of points in such a space. The complexity of calculating each inner product is proportional to the dimension N of $\mathbf{F}$ which can be great (even infinite). However, it is possible to calculate efficiently this inner product using the input space through the *kernel functions*:

$$k(x,z) = (\mathbf{\Phi}(x), \mathbf{\Phi}(z)) \tag{2.3}$$

The kernel methods provide a modular platform as shown in Figure 2.7. At first, the data is processed by using a kernel to create a *kernel matrix*. Then, a variety of pattern algorithms can be used to produce a *pattern function*. In other words, any kernel can be used with any pattern algorithm.

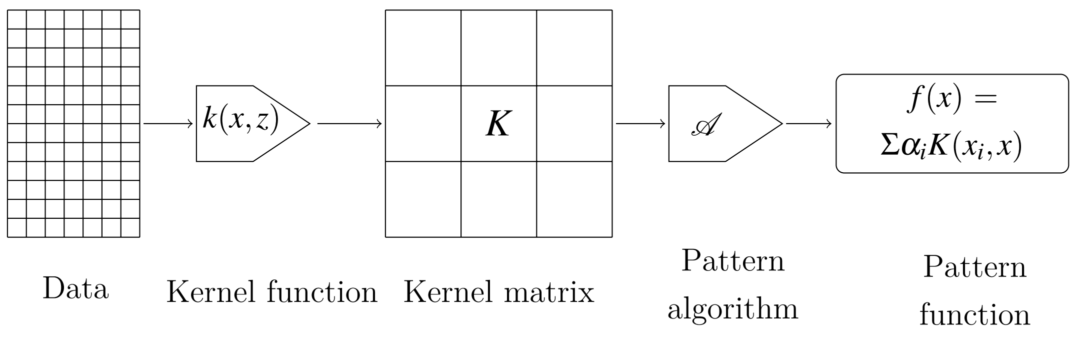


Figure 2.7: The stages of the application of kernel methods.(Shawe-Taylor and Cristianini, 2004)

2.7 Properties of kernels

Equation 2.3 defines the kernel function as the dot product of two-point images in a feature space. In this section, we discuss the properties of the kernels considering the properties of the inner product and its relation with the positive semi-definite notion of Gram matrices.
This section is interested in recalling some notions emanating from linear algebra and which constitute the foundations of the properties of the kernels functions.(Strang, 2009; Mohri et al., 2012).

Definition 2.7.1. An inner product space is a vector space E including a function $\langle .,. \rangle$: $\mathbf{EE} \rightarrow \mathbb{R}$ such :

- For each $x,y \in \mathbf{E} : \langle x,y \rangle = \langle y,x \rangle$ (symmetry).
- For each $x,y,z \in \mathbf{E}$ and $\alpha \in \mathbb{R} : (\alpha x,y) = \alpha \langle x,y \rangle$ and $\langle x+y,z \rangle = \langle x,y \rangle + \langle x,z \rangle$ (linearity).
- For each $x \in \mathbf{E}, x \neq 0 \Rightarrow \langle x,x \rangle > 0$, (positive).

Moreover, the inner product space is said to be strict if $\langle x,x \rangle = 0 \iff \mathbf{x} = \mathbf{0}$.

The function $\langle .,. \rangle$ is called inner product. Each inner product space is a normed linear space with the Euclidean norm $\|x\| = \sqrt{\langle x,x \rangle}$ and therefore a metric space with the distance $d(x,z) = \|x-z\| = \sqrt{(x-z,x-z)}$.

An inner product space is sometimes called a Hilbert space. In literature, other definitions require the properties of completeness and separability.

Definition 2.7.2. *Gram matrix*: Let us consider the vectors $S = \{x_1,\ldots,x_\imath\}$ in the set of the input space $\mathbf{X}$. The matrix $\mathbb{G}, \imath$ x $\imath$, inner products between these vectors (the inputs $G_{ij} = (\mathbf{x}_i, \mathbf{x}_j)$) is called the Gram matrix associated with S.
In the case of the kernel functions, $G_{ij} = (\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) = K(\mathbf{x}_i, \mathbf{x}_j)$. This matrix contains all useful information to calculate the distance between all the data pairs. Moreover, it is symmetrical: $G_{ij} = G_{ji}$.

Definition 2.7.3. *eigenvalues*, eigenvector and spectrum: Let $\mathbf{A}$ be a n x n matrix, a scalar λ is a eigenvalue of $\mathbf{A}$, if there exists a non-zero vector v such that $\mathbf{Av} = \lambda v$. In this case, v is eigenvector associated with the value λ, The set of eigenvalue of a matrix $\mathbf{A}$, denoted $\lambda(\mathbf{A})$ and called spectrum of $\mathbf{A}$.

Definition 2.7.4. $\lambda = 0$ is a eigenvalue of $\mathbf{A}$ if $\mathbf{A}$ is a singular matrix.

Definition 2.7.5. *Positive semi-definite matrix*: A symmetric matrix is positive semi-definite, if all its eigenvalues are non-negative. To recognize this type of matrix, it is sufficient to calculate the eigenvalues and test if $\lambda > 0$. However, this type of solution must be avoided because the calculation of eigenvalues is not so easy, especially for important dimension matrices (Shawe-Taylor and Cristianini, 2004).

Proposition 1. Closure properties: Let k_1 and k_2 be two kernel functions on **X** x **X**, where $\mathbf{X} \in \mathbb{R}^d$, $c \in \mathbb{R}^+$, $f(.)$ a real function on **X**, $poly(.)$ a polynomials with positive or null coefficients, $\phi(x)$ a function of X on $\mathbb{R}^D$, **A** is a positive semi definite matrix, x_a and x_b variables with $x = (x_a, x_b)$ and k_a and k_b kernel functions in their respective space, then the following functions are kernel functions :

- $k(x,x) = ck_1(x,x')$
- $k(x,x') = f(x)k_1(x,x')f(x')$
- $k(x,x') = poly(k_1(x,x'))$
- $k(x,x') = exp(k_1(x,x'))$
- $k(x,x') = k_1(x,x') + k_2(x,x')$
- $k(x,x') = k_1(x,x')k_2(x,x')$
- $k(x,x') = x^T Ax'$
- $k(x,x') = k_a(x_a,x'_a) + k_b(x_b,x'_b)$
- $k(x,x') = k_a(x_a,x'_a)k_b(x_b,x'_b)$

The proof of proposition 1 can be found in (Shawe-Taylor and Cristianini, 2004).

2.8 Example of kernel

Among the kernel functions, we can mention the three most used kernels in literature, namely the linear kernel, the polynomial kernel and the Gaussian kernel (Marref, 2013).

- **The linear kernel**: is a simple inner product: $k(x,x') = (x,x')$.
- **The polynomial kernel**: allows to represent decision boundaries by polynomials of degree d. The generic form of this kernel is: $k(x,x') = (a(x,x') + b)^d$
- **The Gaussian kernel**: is a very used kernel in practice, which is evaluated according to:
$$k(x,x') = exp - \frac{\|x - x'\|^d}{2\sigma^2} \text{ where } \sigma \text{ is the covariance of the entire data set}$$

3

Related Work

"Perhaps the most important principle for the good algorithm designer is to refuse to be content."

– Aho, Hopcroft, and Ullman, *The Design and Analysis of Computer Algorithms, 1974*

KERNEL methods have been widely used to extend the applicability of many well-known algorithms, such as the Perceptron (Aizerman et al., 1964), Support Vectors Machine (Cortes and Vapnik, 1995), or Principal Component Analysis (Zelenko et al., 2003). The hierarchical structure of the trees reflects the underlying dependence information from the domain it represents. Indeed, such an addiction is essential during the learning process, it embarks relevant information. In general, the flat representation approaches of trees, in the form of vectors, fail to capture these dependencies. However, the kernel methods avoid having direct access to the feature space, because it is possible to replace the inner product with a kernel function that calculates the similarity between two trees directly in their input space. Thus tree kernels are the appropriate tools to evaluate the similarity between two trees.

In recent years, tree kernels have been proposed for applications to machine learning in natural language or for the calculation of the similarity of the XML documents. They show a quadratic complexity and less accuracy than traditional attribute / value methods.

This chapter provides a study of some related works of tree kernels, in particular subtree and subset tree kernels.

3.1 Subtree kernel

Vishwanathan and Smola (2003) proposed an algorithm for matching discrete objects such as a strings and trees. When applied to trees, the subtree kernel (STK) uses a feature space indexed by subtrees. The component $\phi_s(T)$ of a subtree t_s represents the number of occurrences of t_s in T. The corresponding kernel is expressed as a weighted sum on all subtrees shared by two trees T_1 and T_2.

$$K(T_1,T_2) \quad = \quad \sum_{s\in\Sigma^*} \phi_s(T_1)\phi_s(T_2)w_s. \tag{3.1}$$

Where Σ^* is the set of all subtrees and w_s is the weight associated with the t_s tree. Indeed, the computation of the STK kernel is reduced to the computation of the string kernel, starting with the encoding of subtrees as a strings. To do so, we must define a lexicographic order between the labels of the tree, if they exist. Moreover, we add two symbols '[' and ']' with '[' < ']' and '[', ']' $< label(v)$ for all labels on the tree. The encoding of a tree of root v is realized by the function $tag(v)$ described as follows:

- If v is an unlabeled leaf then $tag(v) = [\,]$.
- If v is a tagged leaf then $tag(v) = [label(v)]$.
- If v is an unlabeled node with children $v1,\ldots,vc$ then define a sorted permutation π child nodes such as
$$tag(v_{\pi_{(i)}}) \leq tag(v_{\pi_{(j)}}) if \pi_(i) \leq \pi_{(j)}$$
. So, define
$$tag(v) = [tag(v_{\pi_{(1)}})tag(v_{\pi_{(2)}})\ldots tag(v_{\pi_{(c)}})]$$
.
- If v is a node labeled with children $v_1,\ldots,v_c$ then perform the same operations as the previous step and put
$tag(v) = [label(v)tag(v_{\pi_{(1)}})tag(v_{\pi_{(2)}})\ldots tag(v_{\pi_{(c)}})]$.

For example, the tree in Figure 3.1 can be represented by the string
"$[S[NP[D[a]][[N[solution]]][VP[V[proposes]]][N[Djelloul]]]$"

The Theorem 1.1 of Vishwanathan and Smola (2003) summarizes the results obtained in tree with l nodes and λ the maximum width of a label:

- $tag(v)$ can be computed in a time $O((\lambda+2)(l\log_2 l))$ and requires a space linear storage based on the number of nodes in the subtree of root v.
- Each substring s of $tag(v)$, starting with $'['$ and ending with $']'$ is balanced, has a corresponding subtree.

- If the trees T and $\widehat{T}$ are equivalent (T can be obtained from $\widehat{T}$ in permuting the child nodes) then their $tag(v)$ is the same. By the way, $tag(v)$ allows a reconstruction of a single element of the equivalence class.

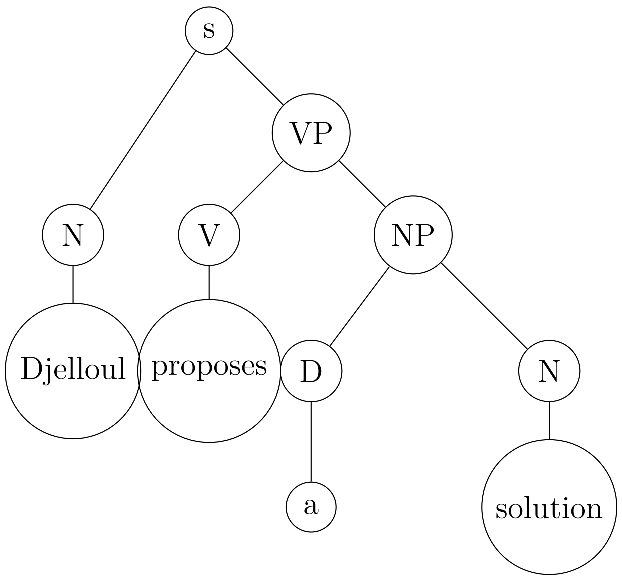


Figure 3.1: example of tree t

3.2 SubSet tree kernel

Subset Tree Kernel ($SSTK$) (Collins and Duffy, 2002a), also called The parse tree kernel (PTK), is based on counting common subset trees of trees. Syntactic trees are obtained from the representation of grammatical relations between words of a sentence. They are considered a typical structure in the natural language processing. Figure 2.5 shows a syntax tree for the sentence "Djelloul propose a solution".
The $SSTK$ use a feature space indexed by all subset trees of a syntax tree T. The component $\phi_S(T)$ of a subset tree t_s represents the number of occurrences of t_s in T. The T tree is represented by the vector

$$\phi(T) = [\phi_1(T), \phi_2(T), \dots, \phi_{|T|}(T)] \tag{3.2}$$

where $T = \{T_1, T_2, \dots, T_{|T|}\}$ is the space of substructures.
Let T_1 and T_2 be two trees, the corresponding kernel is defined as follows:

$$k(T_1, T_2) = \langle \phi(T_1), \phi(T_2) \rangle \tag{3.3}$$

$$= \sum_{s=1}^{|T|} \phi_s(T_1)\phi_s(T_2). \tag{3.4}$$

However, the explicit computation of this kernel is hard, since the number of subtrees is exponential depending on the size of the tree. The technique of the convolution kernel, makes it possible to calculate the inner product in the feature space of high dimension without enumerating explicitly all the features.
In view of this latter consideration, the SST kernel can be evaluated recursively. Let $I_s(v)$ be an indicator function equal to 1 if t_s is a subtree rooted to the tree T in v, 0 otherwise. So $\phi_S(T_1) = \sum_{v_1 \in V_1} I_S(V_1)$ and $\phi_s(T_2) = \sum_{v_2 \in V_2} I_S(V_2)$ where V_1 and V_2 represent respectively the set of nodes of trees T_1 and T_2.
Therefore, the *SST* kernel can be expressed as:

$$\begin{aligned} k(T_1,T_2) &= \sum_{s=1}^{|T|} \sum_{v_1 \in V_1} I_s(v_1) \sum_{v_2 \in V_2} I_s(v_2) \\ &= \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} \sum_{s=1}^{|T|} I_s(v_1) I_s(v_2) \\ &= \sum_{v_1 \in V_1} \sum_{v_2 \in V_2} \Delta(v_1,v_2) \end{aligned} \tag{3.5}$$

where

$\Delta(\mathrm{v}_1, v_2) = \sum_{s=1}^{|T|} I_s(v_1) I_s(v_2)$(3.6) it can be evaluated recursively as follows:

- If the productions at v_1 and at v_2 are different, then $\Delta(v_1, v_2) = 0$.
- If the productions at v_1 and at v_2 are identical and v_1 and v_2 only have leaf children (pre-terminal symbols), then $\Delta(v_1, v_2) = 1$.
- If the productions at v_1 and at v_2 are identical and v_1 and v_2 are not pre-terminals,so :

$$\Delta(v_1,v_2) = \prod_{j=1}^{nc(v_1)} (1 + \Delta(ch_{v_1}^j, ch_{v_2}^j)), \tag{3.7}$$

where $nc(v)$ is the number of children of v and ch_v^j returns the j^{th} child of node v. To be concerned about the influence of fragment size of subtrees on the value of the kernel, it is possible either to limit the depth of the subtrees considered, or penalize them according to their size. This can be obtained by introducing a decay factor $\lambda \in]0,1]$ and by modifying the basic case and the recursive definition as follows:

$$\Delta(v_1,v_2) = \lambda \text{ and } \Delta(v_1,v_2) = \lambda \prod_{j=1}^{nc(v_1)} (1 + \Delta(ch_{v_1}^j, ch_{v_2}^j)) \tag{3.8}$$

This corresponds to a modified kernel:

$$K(T_1, T_2) = \sum_{s=1}^{|T|} \lambda^{size_s} \phi_s(T_1) \phi_s(T_2)$$

where $size_s$ is the number of nodes in the subtree t_s. The use of the dynamic programming technique leads to a computation complexity of the SST kernel in the worst case $O(n^2)$, where n is the number of nodes of the largest input tree.

3.3 Fast Tree Kernel

Moschitti (2006a) shows that tree kernels are very useful in the natural language processing. He provides a simple algorithm to compute the tree kernel in quadratic time, and his study about the classifications and properties of various tree kernels, and he shows that the combinations of kernels always improves traditional methods.
Furthermore, the tree kernel have been applied to reduce such an effort for several tasks in natural language processing, for example the extraction of relations (Schölkopf et al., 1997), Named-entity recognition (Culotta and Sorensen, 2004; Cumby and Roth, 2003), and the semantic role labeling (Moschitti, 2004).

The main idea of his proposition is to compute the number of common substructure between two trees t_1 and t_2. To this end, Moschitti modified the kernel function proposed by (Collins and Duffy, 2002a). To do so, he slightly modifies the equation (3.4) by introducing a parameter $\sigma \in \{0,1\}$ that allows the evaluation of subtrees ($\sigma = 0$) or subset trees ($\sigma = 1$). So :

$$\Delta(v_1, v_2) = \prod_{j=1}^{nc(v_1)} (\sigma + \Delta(ch^j_{v_1}, ch^j_{v_2})), \tag{3.9}$$

The kernel $K(t_1,t_2)$ is the number of common substructures between t_1 and t_2.

To solve the problem of time complexity, an algorithm with a linear complexity for computation of the subtrees kernel (STK), was conceived in (Vishwanathan and Smola, 2003). Moschitti design an algorithms that run in "linear time on average" and he named it as (Fast tree kernel), the pseudo-code is described in algorithm 3. To compute with fast tree kernel, sum $\Delta(v_1, v_2)$ were $v_1 \in N_{T_1}$ and $v_2 \in N_{T_2}$ defined in (Equation 3.5) , when the productions associated with v_1 and v_2 are different. Thus, look for a node pair set $N_p = (v_1, v_2) \in N_{T_1}N_{T_2} : p(v_1) = p(v_2)$, to efficiently build N_p, they extract the lists L_1 and L_2 of the production rules from T_1 and T_2, sort them in the alphanumeric order and scan them to find the node pairs (v_1, v_2) such that $(p(v_1) = p(v_2)) \in L_1L_2$. This, may require only $O(|N_{T_1}| + |N_{T_2}|)$ time.

This low complexity allow the use of tree kernel with SVM on large training set. To confirm this hypothesis, Moschitti measured the impact of the algorithm on the time required by SVM for learning about $122,774$ examples of predicate arguments annotated in the PropBank database (Kingsbury and Palmer, 2002) and $37,948$ annotated cases in FrameNet (J Fillmore, 1982).
To solve the problem of less accuracy, a study on the different substructures of trees is carried out to obtain tree kernel that provide the greatest precision. Moreover, *SSTs* provide algorithms with richer information that may be essential to capture the syntactic properties of derivation trees as indicated. Furthermore, if the *SST* has too many irrelevant aspects, overfitting can occur and decrease the accuracy of the classification (Cumby and Roth, 2003). As

a result, the fewer features of the ***ST*** approach may be more appropriate.

Algorithms 1 Tree kernel computation

```
Output:kernel
1: function KERNEL(Node $t_1$,$t_2$)
2:     $P$ ←GeneratePairs($t_1$,$t_2$)
3:     $kernel \leftarrow 0$
4:     for each pair in $P$ do
5:         $kernel$ ←$kernel$ + getDelta(pair($n_1$),pair($n_2$))
6:     end for
7:     return $kernel$
8: end function
```

Algorithm 2 describes the function of comparing the nodes from the two trees t_1 and t_2.

Algorithms 2 Comparing nodes algorithm

```
1: Inputs: node $t_1$ of tree 1 and node $t_2$ of tree 2
2: Output: $delta$
3: Initialize:
4: $\sigma \leftarrow 0$                ▷ 0 to evaluate subtrees, 1 to evaluate subset trees
5: function GETDELTA(Node $t_1$,$t_2$)
6:     $delta \leftarrow 1$
7:     if $t_1$.getProduction() ≠ $t_2$.getProduction() then
8:         $delta = 0$
9:     else if $t_1$.getProduction() = $t_2$.getProduction() and $t_1$.isPreterminal() and $t_2$.isPreterminal() then
10:        $delta = 1$
11:    else
12:        for $i$ from 0 to size(childrens($t_1$)) do
13:            $delta$ = $delta$ * ( $\sigma$+ getDelta($t_1$.childern[$i$],$t_2$.childern[$i$]))
14:        end for
15:    end if
16:    return $delta$
17: end function
```

The algorithm in 3 show the evaluation of tree kernels used Fast Tree Kernel (FTK)

Algorithms 3 Pseudo-code for fast evaluation of the node pair sets used in the fast Tree Kernel.

```
function EvaluatePairSet(Tree $T1$, $T2$) returnNODE PAIR SET
    LIST $L_1$,$L_2$
    NODE PAIR SET $N_p$
    $L_1 \leftarrow T_1$.ordered list;
    $L2 \leftarrow T2$.ordered list                          ▷ the lists were sorted at loading time
    $n_1 \leftarrow$extract($L_1$)                            ▷ get the head element and
    $n_2 \leftarrow$extract($L_2$)                            ▷ remove it from the list
    while $n_1$ and $n_2$ are not NULL do
        if $n_1$.getProduction() > production of($n_2$) then
            $n_2$ = extract($L_2$)
        else if production of($n_1$) < production of($n_2$) then
            $n_1$ = extract($L_1$)
        else
            while $n_1$.getProduction() == $n_2$.getProduction() do
                while $n_1$.getProduction() == $n_2$.getProduction() do
                    add(($n_1$,$n_2$),$Np$)
                    $n_2$=get next elem($L_2$)                ▷ get the head element
                                                      ▷ move the pointer to the next element
                end while
                $n_1$ = extract($L_1$)
                reset($L_2$)                          ▷ set the pointer at the first element
            end while
        end if
    end while
    return $N_p$
end function
```

Example 3.3.1. Now, let us give an example for computation tree kernel between two trees t_1 Figures 3.3 and t_2 3.2.

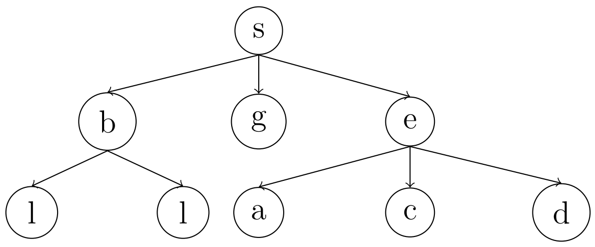


Figure 3.2: Tree t_1

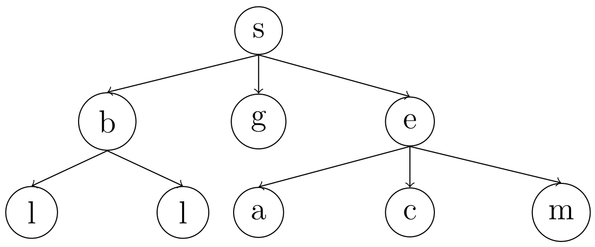


Figure 3.3: Tree t_2

The kernel computation for these two trees is given by:

$$\begin{aligned} K(T_1,T_2) &= \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1,n_2) \\ &= \Delta(s,s)+\Delta(s,b)+\Delta(s,g)+\Delta(s,e)+\ldots+\Delta(s,m) \\ &+\Delta(b,s)+\Delta(b,b)+\Delta(b,g)+\Delta(b,e)+\ldots+\Delta(d,m) \end{aligned}$$

We will now calculate each $\Delta(n_1,n_2)$, for example:
We have $\Delta(b,b) = 1$ since both are non-leaves nodes and have the same productions,
We have $\Delta(d,m) = 0$ since they do not have the same symbol,
We have $\Delta(e,e) = 0$ since they are non-leaves so we will see if the productions are the same.
We have $\Delta(e,e) = \Delta(a,a)\Delta(c,c)\Delta(d,m),\cdots$

3.4 Conclusion

In this chapter we have presented common tree kernel, namely ST, SST and FTK kernels, we highlight the results of FTK in terms of efficiency. In fact the experiments reveal that the time complexity of FTK is linear in the average for parse trees.

4

From *k*-ary Tree To Binary Tree

"Controlling complexity is the essence of computer programming. "
– Brian Kernigan

BINARY tree data structures play an important role in almost every area in computer science (which are for example widely used in database). Today the bursting of data makes a greater demand for the performance of the trees.
In the literature, researchers have provided several methods of binarization of an k-ary tree, which allow them to derive different results in several domains. Binary trees are simple structures which allow to solving (in the algorithmic sense) many problems. We wanted this conversion to find a relationship between k-ary tree and the equivalent binary tree, then we calculate the kernel in binary tree structure instead of k-ary tree strucuture.

From existing methods to represent a *k*-ary tree to its equivalent binary tree, we can cite:

- Knuth binarization (Knuth, 1969).
- Threaded Binarization (an algorithm based on threaded binary tree in forest). (Knuth, 1969; Horowitz et al., 1996).
- Kumar binarization (Kumar Ghosh et al., 2008).
- Recursive algorithm design ideas of converting the forest into the corresponding binary tree (Wang, 2011).

In our experiment we focus on Knuth binarization and our proposed binarization algorithm. We describe both methods in the next sections.

4.1 Knuth Binarization

Knuth (1969) states that, in natural way, we can represent any k-ary as a binary tree. The binary tree obtained here has one-to-one correspondence with the original tree. However, after converting one tree (other than the binary tree) to the equivalent in the binary tree structure, the root node of the computed binary tree has no right subtree. Algorithm 4 illustrates the function of binarizing trees the technique of conversion is as follows (Knuth, 1969; Horowitz et al., 1996):

- Connect the sibling nodes at any level.
- Keep the link from the parent to the leftmost child and eliminate the subsequent links to the children.
- The root is the center, rotate the tree by 45 degree clockwise. The obtained tree is a binary tree.

To illustrate this technique, the figures 4.1 and 4.2 show the method applied to the tree in Figure 3.2 :

In the first step we link all the child nodes for the same parent in the same level as shown in Figure 4.1. The children of each parent are connected together. As for example, b, g and e are the children of s, hence, they belong to the same family. Similarly as l and l belong to one family, and a ,c and d to another. However, l and a do not belong to the same family.

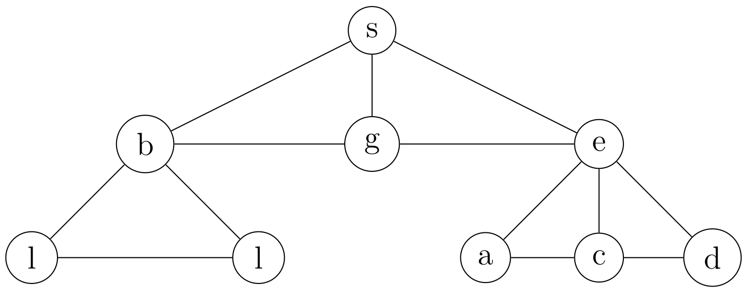


Figure 4.1: Connect all the child nodes at the same level for the same parent.

Now, from this representation in Figure 4.1, we remove the vertical links and keep the ones that connect the parent to the leftmost child, for example, b is the first child for s. Similarly a is the first child for e. The next obtained tree is shown in Figure 4.2.

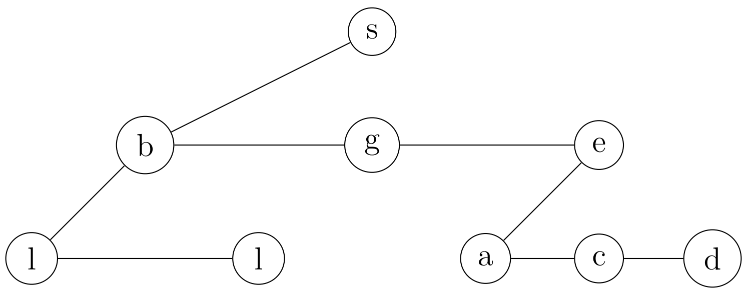


Figure 4.2: Preserve only the link from parent to the leftmost child and the first child to its sibling

This tree is rotated keeping the root as the center clockwise, giving the binary tree as shown in Figure 4.3. Generally, if two or more k-ary trees are considered in a forest, then the right child of the root of the final tree would contain the root of the second tree in the given forest.

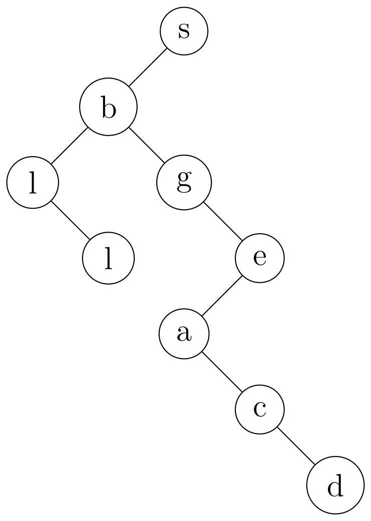


Figure 4.3: the equivalente binary tree of tree shown in Figure 3.2

Algorithms 4 Knuth Binarization

```
1: Inputs: $t$ a k-ary tree
2: Output : an equivalent binary tree to $t$
3: function ToBinaryTree(Node $t$) :An equivalent binary tree to $t$
4:     BTreeNode $n$                                  ▷ $n$ is an empty binary tree
5:     $n$.setValue($t$.value)
6:     if $t$.hasChildren() then
7:         Node $c \leftarrow t$.getFirstChild()
8:         $n$.setLeft(ToBinaryTree($c$))
9:     end if
10:    if $t$.getRightSibling() $\neq null$ then
11:        $n$.setRight(toBinaryTree($t$.getRightSibling()))
12:    end if
13: end function
```

4.2 Proposed technique of binarization

In the previous section we explained the Knuth method, and how binarizing tree with this method by giving an example.
During our study in binarization, we note that Knuth method are limited and do not keep the notion of subtree and loss the information, These difficult problems have prompted us to think of a new method that tries to address them.

The rest of this section explains our proposed method and we prove its efficiency.

The proposed method is as fallow :

- Each parent node in the k-ary tree remains parent in the equivalent binary tree.
- Recursively, at the left child of parent node, put the leftmost child in the k-ary tree.
- At the right child of parent node, put an # node.
- # node contain at the left node the sibling of its parent left child. and at the right node connect with # node recursively.

Figure 4.4 shows our proposed method of binarizing the k-ary tree given in Figure 3.2. One of the strength of our proposed method is that the property of subtree is conceived. The proposition 2 claims this property for our proposed method.

Definition 4.2.1. The equivalent binary tree with the proposed method can define as :

$$t' = \begin{cases} a, & \text{if } t' = a. \\ f(t'_1, \#(t'_2, \ldots, \#(t'_n - 1, \#(t'_n, \perp)) \ldots)), & \text{otherwise.} \end{cases}$$

Proposition 2. Let t be an k-ary tree, and t' its equivalent binary tree (in terms of the proposed method) , s is a subtree of t and s' is a subtree of t'. We have then $s < t \implies s' < t'$.

Proof. Let $t = f(t_1, t_2, \ldots, t_n)$.
The proof is by induction on the size of t, if $t = a$ and if $s < t$ then $s = a$ □

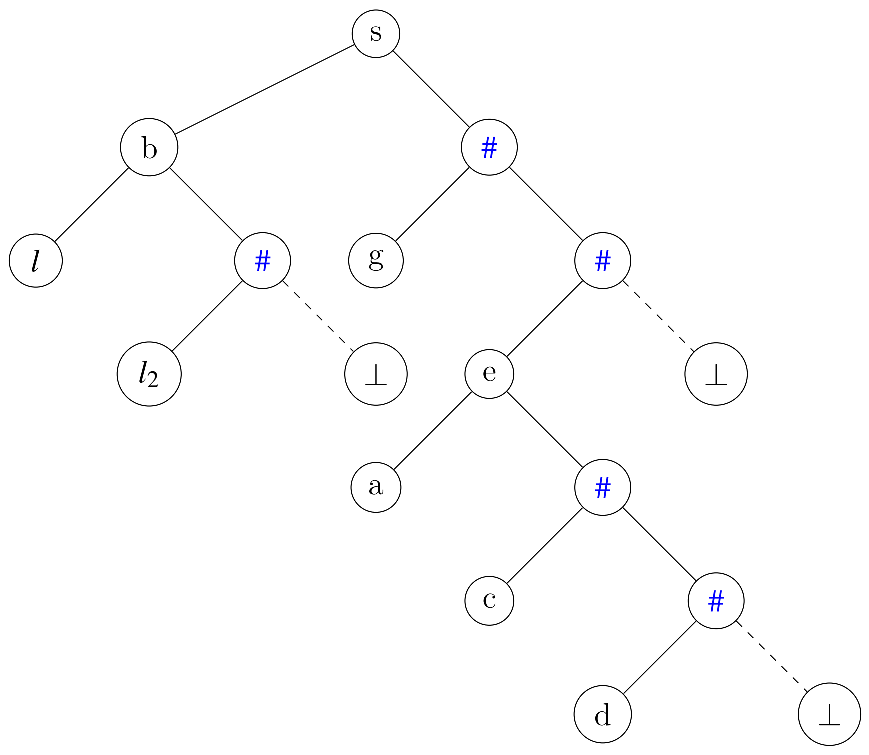


Figure 4.4: the equivalent binary tree to tree shown in 3.2 with the proposed method

To clarify more, the algorithm in 5 explains the proposed method of binarization.

Algorithms 5 The proposed binarization algorithm

```
1: INITILAIZE:
2: Inputs: $t$ a k-ary tree
3: function ToBinaryTree(Node $t$) :An equivalent binary tree to $t$
4:     if $t \neq null$ then
5:         BTreeNode $n, temp$                    ▷ $n$ and $temp$ are binary tree nodes
6:         $n$.setValue($t$.value)
7:         $temp \leftarrow n$
8:         if $t$.hasChildren()  then
9:             Node $c \leftarrow t$.getNext()    ▷ Get the first child of t and move pointer to the next element
10:            $n$.setLeft(ToBinaryTree($c$))
11:        end if
12:        while  $t$.hasChildren()  do
13:            BTreeNode $diaze$
14:            $diaze$.setValue('#')
15:            Node $c \leftarrow t$.getNext()
16:            $temp$.setRight($diaze$)
17:            $temp \leftarrow diaze$
18:            $diaze$.setLeft(ToBinaryTree(c))
19:        end while
20:        $temp$.setRight($\perp$)
21:        return $n$
22:    end if
23:    return $null$
24: end function
```

4.3 The binary tree kernel function

To compute the number of the common substructures between two binary trees n_1 and n_2 or in other context the calculate the similarity in the binary tree, For this purpose, we slightly modied the kernel function proposed in (Moschitti, 2006b) to adapte with the binary tree. We define

$$\Delta(n_1, n_2) = \sigma + \Delta(C^l_{n_1}, C^l_{n_2}) * \sigma + \Delta(C^r_{n_1}, C^r_{n_2}), \tag{4.1}$$

where $C^r_{n_1}$ is the right children of n_1 and $C^l_{n_1}$ is the left children of n_1.

Algorithms 6 Comparing nodes algorithm on binary tree structure

```
INITILAIZE:
Inputs: Binary tree node $t_1$ of tree 1, binary tree node $t_1$ of tree 2
Output: $delta$
function getDelta(Node $t_1$,$t_2$) :$delta$
    $delta \leftarrow 1$
    if $t_1$.getProduction() $\neq$ $t_2$.getProduction() then
        $delta \leftarrow 0$
    else if $t_1$.getProduction() = $t_2$.getProduction() and $t_1$.isPreterminal() and $t_2$.isPreterminal() then
        $delta \leftarrow 1$
    else
        $delta \leftarrow delta$ * (0 + getDelta($t_1$.getLeft(),$t_2$.getLeft())
        $delta \leftarrow delta$ * (0 + getDelta($t_1$.getRight(),$t_2$.getRight())
    end if
    return $delta$
end function
```

4.4 Experiments and discussion

The purpose of these experiments is to show the impact of tree binarization with Knuth method and the proposed method on kernel computation in terms of similarity and running time.

The tests are run on an Intel Atom processor at 1,80 GHZ with 2 GB of RAM under Windows 10, 32 bit. All the algorithms tested are implemented in Java.
In order to conduct the experiment, we generated trees of different size of nodes $(20, 50, \ldots, 1000)$ on very short alphabet size(2), on short alphabets sizes (32) and on large alphabets size (1024). The Algorithm 7 show how we build a tree. All generated k-ary trees are converted into equivalent binary trees using both, Knuth binarization and the proposed method. For each node size, we generate 20 trees. For the purpose to obtain accurate running time, we repeat the experiments 5 times.

Algorithms 7 Tree generation algorithm

```
1: Initialize:
2: $alphabet$: $[a,b,c,...,x,y,z,aa,bb,cc,dd]$        ▷ Short alphabet size
3: $t$ : empty root tree node
4: $n$ : random number                                  ▷ $n$ is number of $t$ children
5: $counter \leftarrow 0$
6: $m \leftarrow 200$                 ▷ $m$ is the number of nodes in $t$ , in this case 200
7: $counter \leftarrow counter + 1$
8: local $i \leftarrow 0$
9: while $i < n$ do
10:     $node \leftarrow$ creatSubElement($t$)
11:     $node.value = alphabet$[random(0..size($alphabet$))]
12:     if $counter < m$ then
13:         $r \leftarrow Random(0..5)$               ▷ 5 means max number of children
14:         GenerateTree($node, r$)
15:     else
16:         break
17:     end if
18:     $i \leftarrow i + 1$
19: end while
20:
```

As we shown in section 3.3, (Moschitti, 2006a) presented an algorithm (Fast tree kernel) for the evaluation of *ST* and *SST* kernels that operate with a linear average time and calculates the kernels between two parse trees in average time "$O(m+n)$", where m and n are the number of nodes in both trees.
To check the claims of (Moschitti, 2006a), we measure the running time of this implementation on *SST* kernel. Using the constructed k-ary tree dataset. Figure 4.5 shows the obtained results.

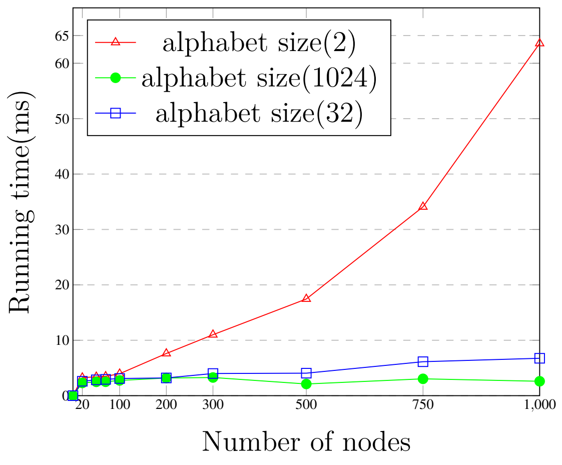


Figure 4.5: Running time of SST kernel in different sizes runs on k-ary tree structure

Running time Has a relationship with the size of alphabet, When the size of alphabet is short compared with the number of nodes the running time became quadratic.

Moschitti, 2006a studied the tree kernel in natural language processing where the size of alphabet almost equal to size of tree, this explains why he reached this average running time.

IN order to study the impact of tree binarization on the tree kernel (*ST* and *SST*), in terms of running time, we consider the three different structures (*k*-ary, binary tree of knuth and binary tree of our proposed approach).
We consider also different alphabet sizes (2,32,1024) and the number of nodes (20,50,...,1000). Figures 4.6, 4.7, 4.8 shows the obtained results.

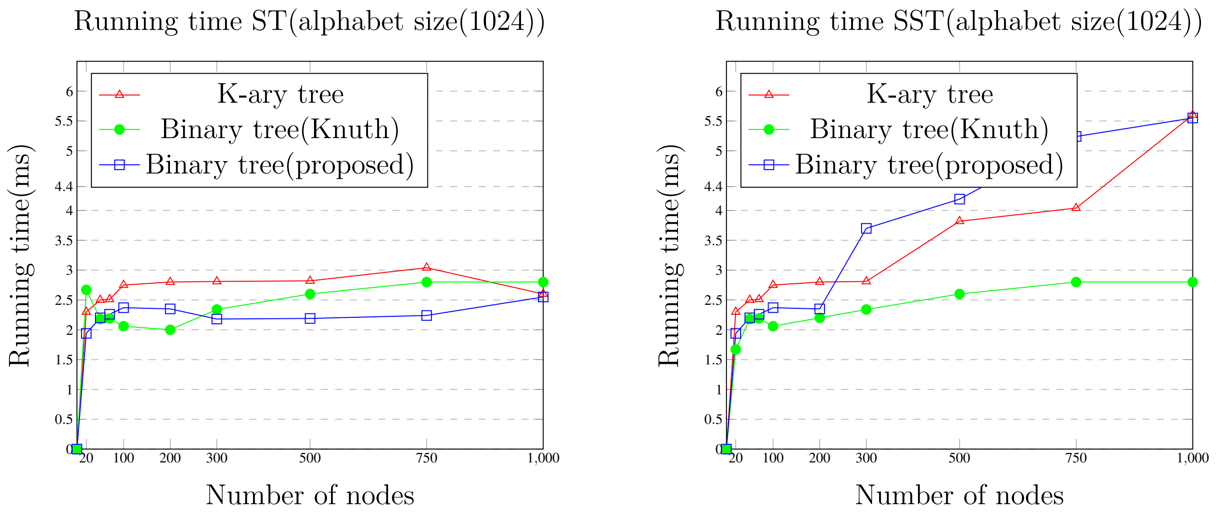


Figure 4.6: Average time in millis seconds for the ST and SST evaluations

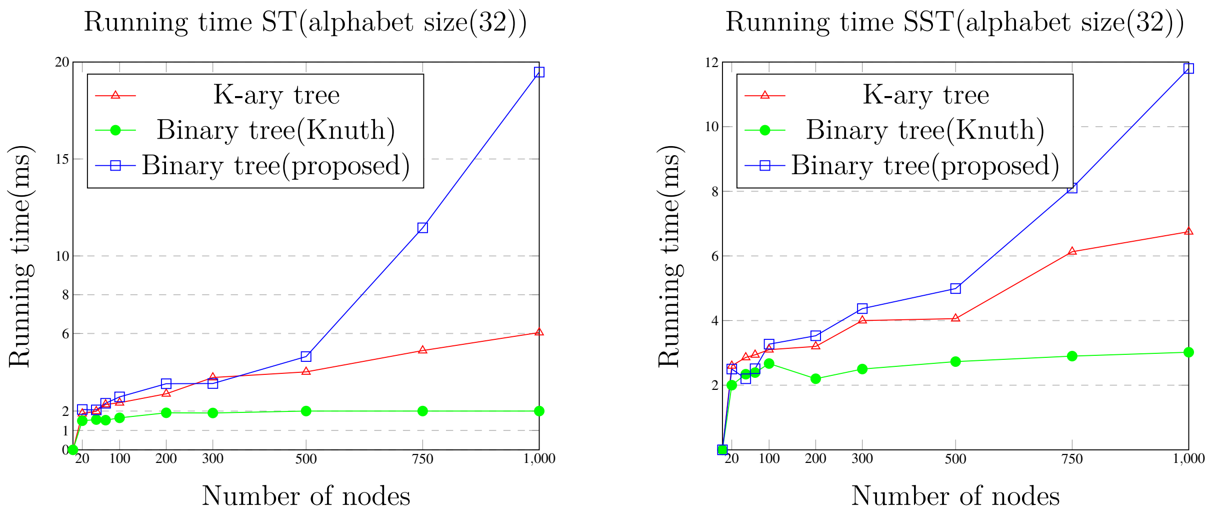


Figure 4.7: Average time in millis seconds for the ST and SST evaluations

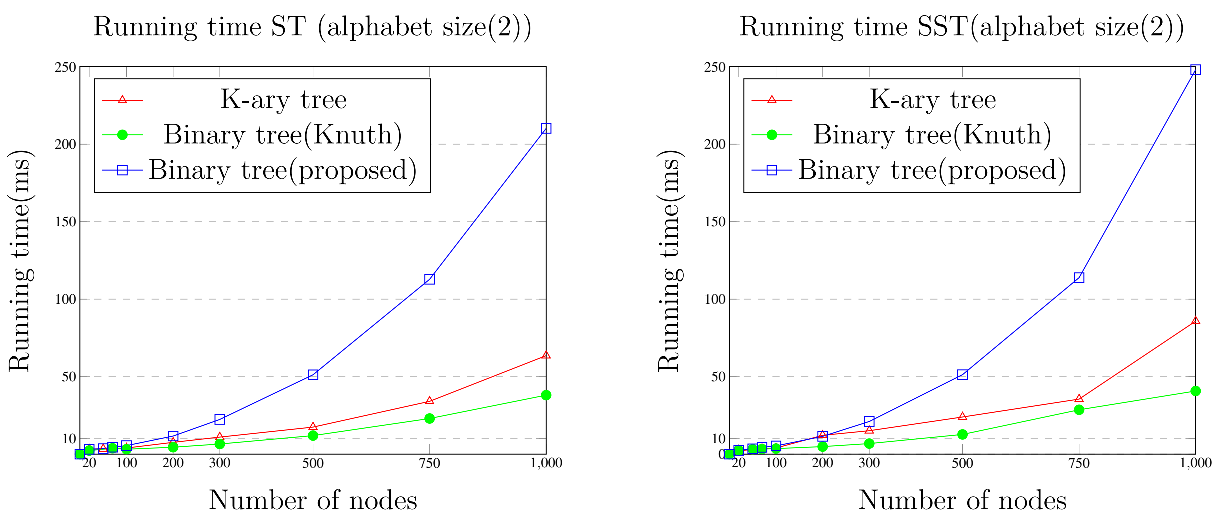
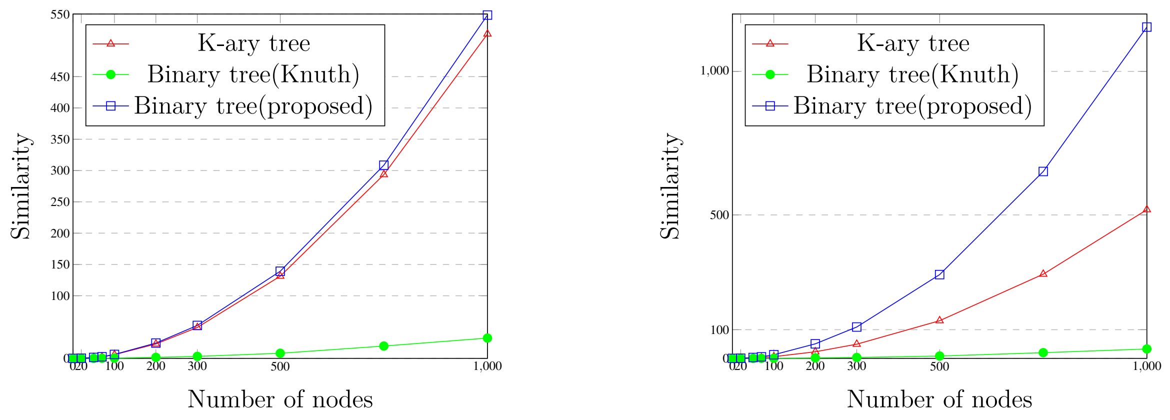


Figure 4.8: Average time in millis seconds for the ST and SST evaluations

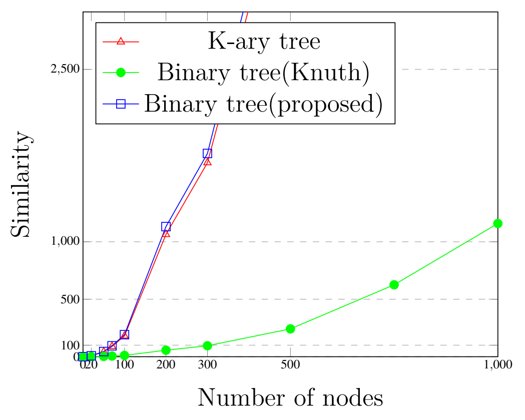
IN order to study the impact of tree binarization on the tree kernel (ST and SST) in terms of similarity, we consider the three different structures (k-ary, binary tree of knuth and binary tree of our proposed approach).

We consider also different alphabet sizes (2,32,1024) and the number of nodes (20,50,70,. . . ,1000). Figures 4.9, 4.11, 4.10 shows the obtained results.

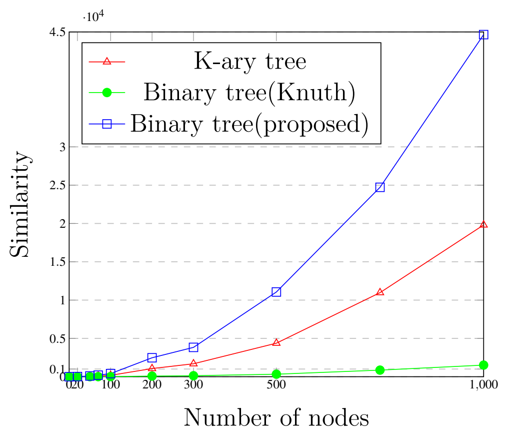


(a) Similarity of ST kernel(alphabet size(1024)) (b) Similarity of SST kernel(alphabet size(1024))

Figure 4.9: Similarity obtained for ST and SST kernel according to large alphabet (1024) and different sizes of trees.

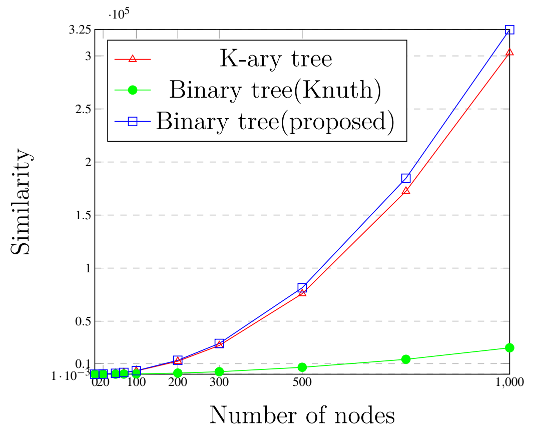


(a) Similarity of ST kernel(alphabet size(32))

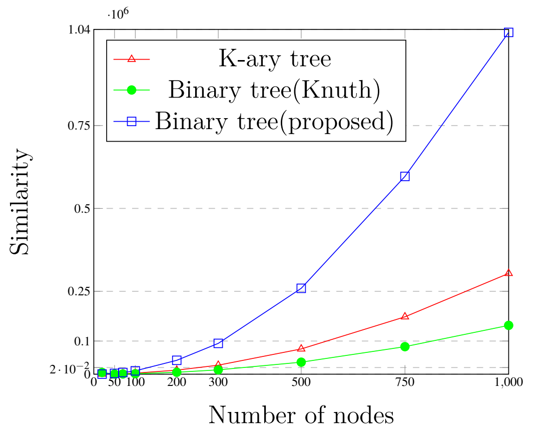


(b) Similarity of SST kernel(alphabet size(32))

Figure 4.10: Similarity obtained for ST and SST kernel according to short alphabet size (32) and different sizes of trees



(a) Similarity of ST kernel(alphabet size(2))

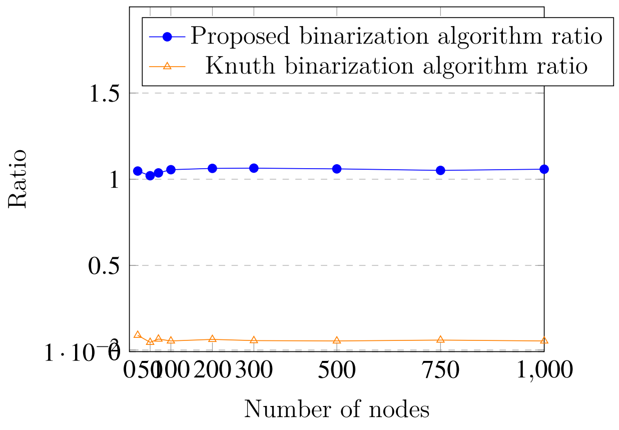


(b) Similarity of SsT kernel(alphabet size(2))

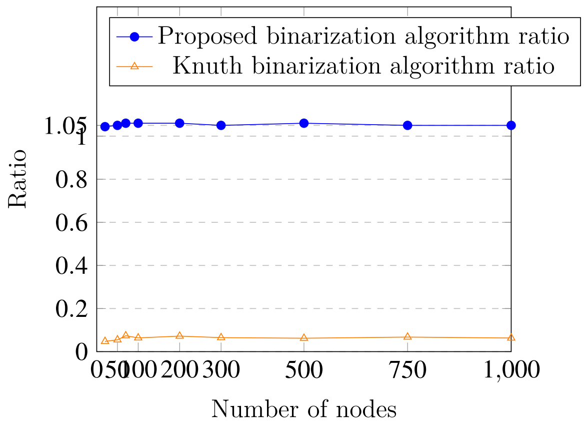
Figure 4.11: Similarity obtained for ST and SST kernel according to very short alphabet (2) and different sizes of trees.

First of all, we can easily observe that *k*-ary tree and its equivalent binary tree with the proposed algorithm is hardly identical in similarity in ***ST***, and this proves to us that the proposed method of binarization keep the property of subtree and the information does not disturb in which leaves in *k*-ary tree remains leaves in the proposed method, also internal nodes.
In general, we can see that the obtained curves have the same order of growth. To validate this observation, we pass to another visualization by considering the ration of similarities of the different structures. Figures illustrates the results of the ratio between different similarities.

We can easily see that the ratio in the k-ary tree and after converted into a equivalent binary tree both, Knuth and proposed method is stable. However, this ratio is different the first (proposed method) which is always greater than 1, but the second (Knuth binarization) does not exceed **0.2**. These results reveal the efficiency of the proposed method, that clearly keep the property of substructures of a tree. Figures 4.12, 4.13, 4.14 shows the obtained results.

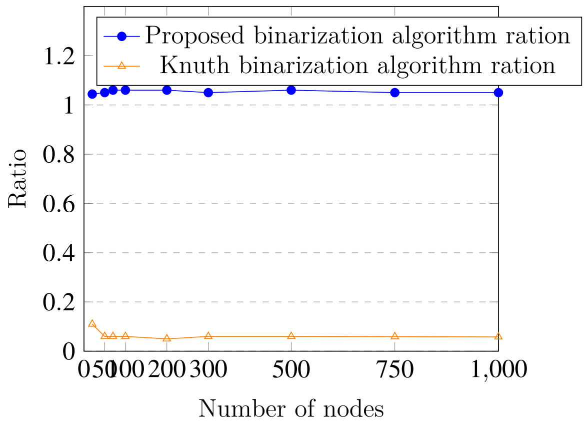


(a) Ratio ST kernel

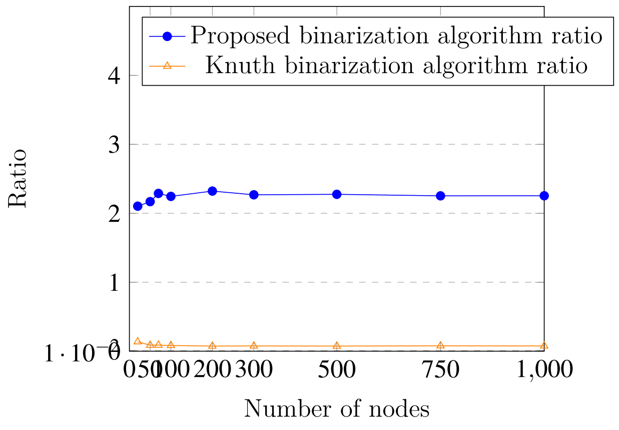


(b) Ratio SST kernel

Figure 4.12: Ratio of similarity obtained for ST and SST kernel according to large alphabet size (1024)

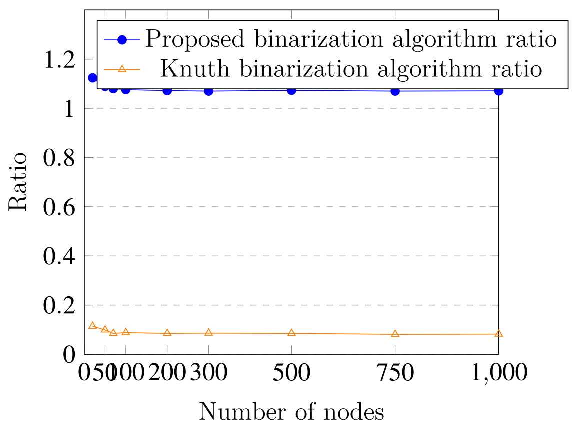


(a) Ratio ST kernel(32)

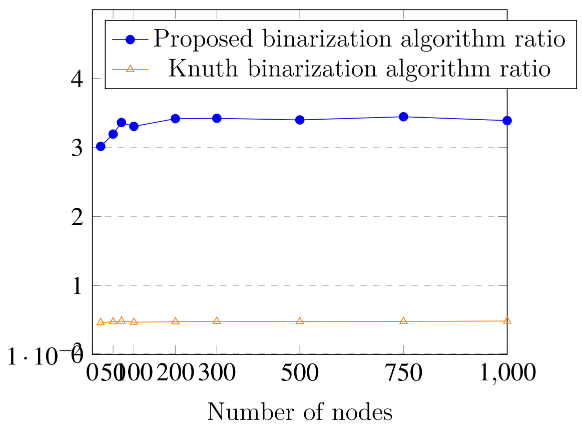


(b) Ratio SST kernel

Figure 4.13: Ratio of similarity obtained for ST and SST kernel according to short alphabet size(32)



(a) Ratio ST kernel(2)



(b) Ratio SST kernel(2)

Figure 4.14: Ratio of similarity obtained for ST and SST kernel according to very short alphabet size(2)

4.5 Conclusion

In this chapter, we discuss the impact of tree binarization on tree kernels, we proposed a new binarization approach (Algorithm 5) which always keep the property of subtrees We conduct experiments involving running time and similarity of the *ST* and *SST* kernel by considering the *k*-ary, Knuth binary and the proposed-binary structures.
We concluded that the proposed method are mush better than Knuth method in similarity but worse than it in running time.

5

Conclusion

"It is hard to fail, but it is worse never to have tried to succeed."
– *Theodore Roosevelt*

THE purpose of this thesis is to evaluate the effect of tree binarization on the kernel. We start with an overview on structured data and some preliminaries and introduced the concept of kernel methods. Thereafter, we investigated the tree kernel, we mentioned some related work in this area and we gave an example of a similarity computation between trees. Subsequently we conducted a comparative study of tree kernel between k-ary tree and after binarization with two method: Knuth method binarization and a proposed method. The comparison was based on similarity and running time measures. We used an XML dataset that we generated.

The result of experiments reveal that the proposed method for binarization keeps the property of subtree and the information does not disturb.

we also plan to focus on the random generation of trees, a task that we consider very useful in this area of research

Bibliography

M. A. Aizerman, È. M. Braverman, and L. I. Rozonoèr. Theoretical foundation of potential functions method in pattern recognition. *Avtomat. i Telemekh.*, 25, 1964.

Hiroki Arimura. Efficient algorithms for mining frequent and closed patterns from semi-structured data. Springer, 2008.

Sujeevan Aseervatham. *Machine Learning with Semantic Kernels for Textual Data.* PhD thesis, Université Paris-Nord - Paris XIII, 2007.

Michael Collins and Nigel Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002a.

Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *Advances in neural information processing systems*, pages 625–632, 2002b.

Antoine Cornuéjols and Laurent Miclet. *Apprentissage artificiel: concepts et algorithmes.* Editions Eyrolles, 2011.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20, 1995.

Corinna Cortes, Patrick Haffner, Mehryar Mohri, Kristin Bennett, and Nicolò Cesa-bianchi. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, pages 1035–1062, 2004.

Aron Culotta and Jeffrey S. Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, 21-26 July, 2004, Barcelona, Spain.*, 2004.

Chad M. Cumby and Dan Roth. On kernel methods for relational learning. In *Machine Learning, Proceedings of the Twentieth International Conference*, 2003.

Haussler. Convolution kernels on discrete structures, 1999.

Ellis Horowitz, Sartaj Sahni, and Sanguthevar Rajasckaran. *Computer Algorithms: C++.* W. H. Freeman & Co., New York, NY, USA, 1996.

Charles J Fillmore. Frame semantics, 1982.

Paul Kingsbury and Martha Palmer. From treebank to propbank. European Language Resources Association (ELRA), May 2002.

Knuth. The art of computer programmingvolume 1: Fundamental algorithms (donald e. knuth). *SIAM Review*, 1969.

Sumit Kumar Ghosh, Joydeb Ghosh, and Rajat Pal. A new algorithm to represent a given k-ary tree into its equivalent binary tree structure. 01 2008.

Nadia Marref. *Apprentissage Incrémental & Machines à Vecteurs Supports*. PhD thesis, Université de Batna 2, 2013.

at SAS Matthew Magne. Global product marketing for data management. 2017.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X, 9780262018258.

Alessandro Moschitti. A study on convolution kernels for shallow statistic parsing. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, 2004.

Alessandro Moschitti. Making tree kernels practical for natural language learning. In *EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics*, 2006a.

Alessandro Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin*, 2006b.

Sameer Pradhan, Kadri Hacioglu, Valerie Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky. Support vector learning for semantic argument classification. *Machine Learning*, 2005.

Bernhard Schölkopf, Alex J. Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *Artificial Neural Networks - ICANN '97, 7th International Conference, Lausanne, Switzerland, October 8-10, 1997, Proceedings*, 1997.

Aliaksei Severyn and Alessandro Moschitti. Fast support vector machines for convolution tree kernels. *Data Min. Knowl. Discov.*, 2012.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, UK, 2004.

Kilho Shin and Taro Niiyama. The mapping distance - a generalization of the edit distance - and its application to trees. In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence*.

Christine Solnon. Théorie des graphes et optimisation dans les graphes. *INSA de Lyon*, 2008.

G. Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, 2009.

S. V. N. Vishwanathan and Alexander J. Smola. Fast kernels for string and tree matching. In *Advances in Neural Information Processing Systems 15 — Proceedings of the 2002 Neural Information Processing Systems Conference*, 2003.

Min Wang. The recursive algorithm of converting the forest into the corresponding binary tree. Springer Berlin Heidelberg, 2011.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. *Journal of Machine Learning Research*, 3, 2003.