A Subpath Kernel for Rooted Unordered Trees

Daisuke Kimura[1] , Tetsuji Kuboyama[2] , Tetsuo Shibuya[1] , Hisashi Kashima[1]

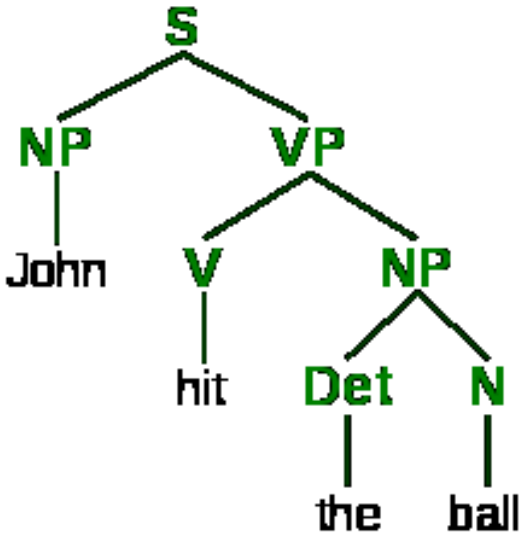
[1] The University of Tokyo

[2] Gakushuin University

2011-06-11 @ERATO WS

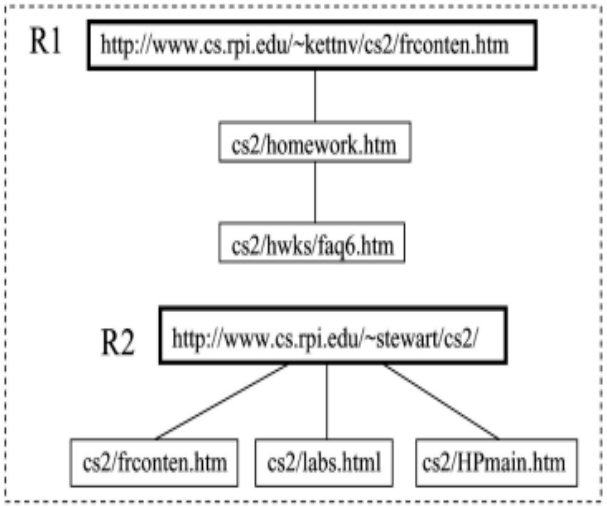
Motivation

- There exist a lot of data represented as **trees**



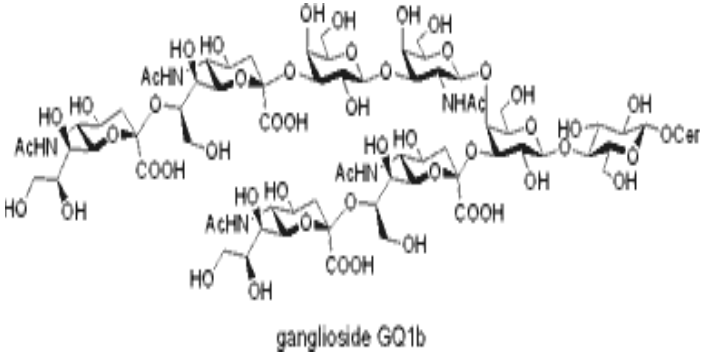
Parse tree

(in natural language processing)



XML

(in Web mining)



Glycan

(in bioinformatics)

Our contribution

- Application of kernel method to classification of **unordered trees**
  - Existing tree kernels are for **ordered trees**
- A new kernel function based on vertical substructures in trees
- An efficient algorithm for kernel computation by extending **Multikey Quicksort**
- Good performance both in time and accuracy

Kernel methods

- Kernel methods are one of the promising approaches to learning with tree-structured data

- Kernel methods for binary classification

$$f(x) = \mathrm{sign}[\ \sum_{i=1}^{N} \alpha_i y^{(i)} k(x, x^{(i)}) + b\ ]$$

$$k(x, x^{(i)}) = \mathbf{\Phi}(x)^\top \mathbf{\Phi}(x^{(i)})$$

  - $k(x, x^{(i)})$ is a kernel function, which is a similarity between two feature vectors of data

  - No need to construct feature vectors explicitly, **only need to calculate an inner product efficiently**

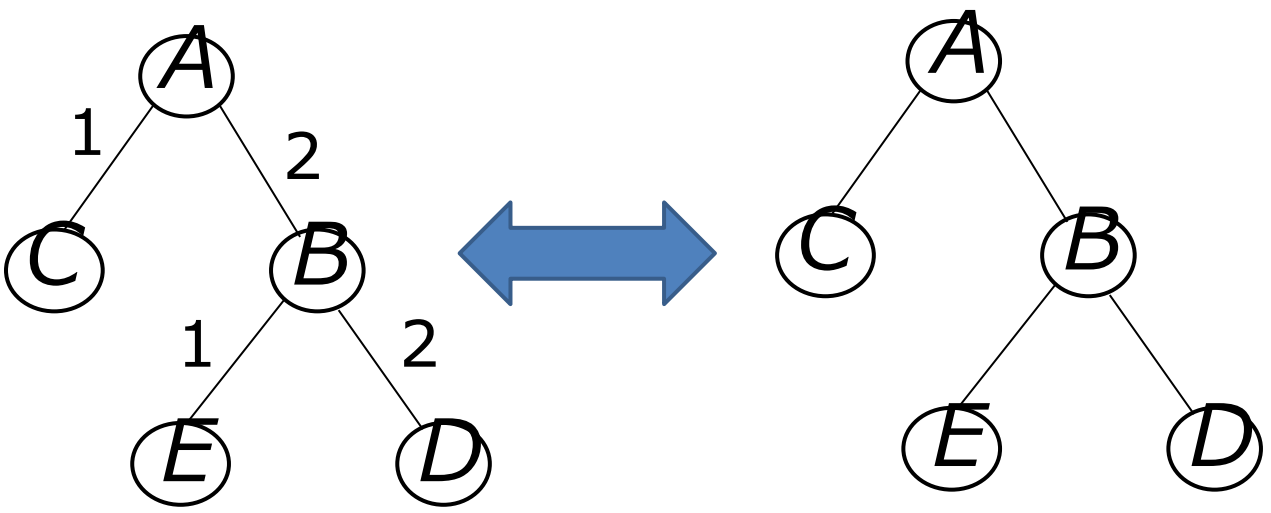
Existing tree kernels

- Existing tree kernels
  - [Collins& Duffy, ’ 02]: The first tree kernel (using subset trees)
  - [Kashima&koyanagi, ’ 02] : More general form of the tree kernel by Collins
  - [Kuboyama et al., ’ 07] : More general form of the convolution kernel
  - [Aiolli et al., ’ 09] : A Route kernel (using vertical structure)

e.t.c.

- Most of the existing tree kernels are for **ordered** trees

The children of each node are ordered



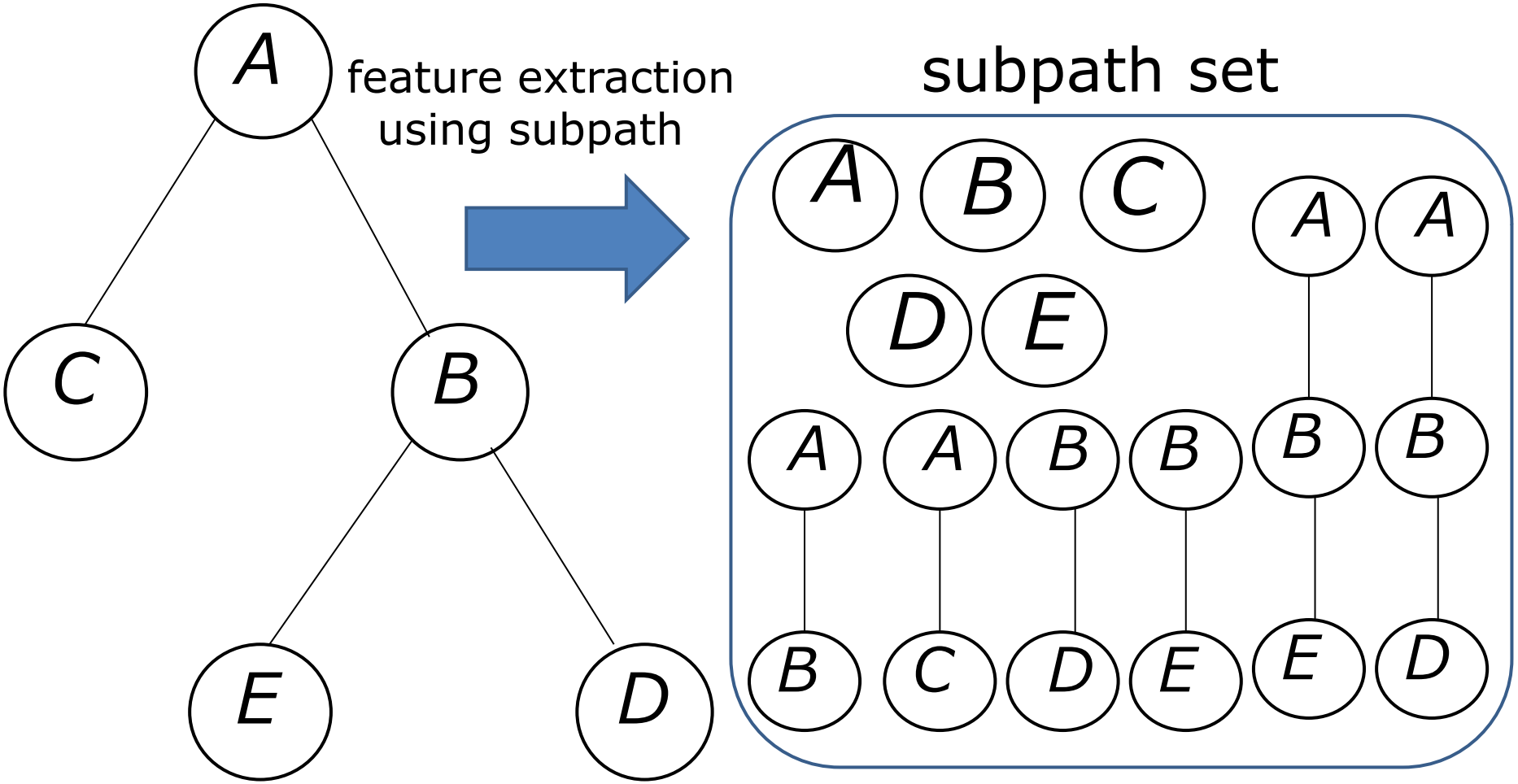
Proposed tree kernel

- We propose a tree kernel for rooted **unordered** trees
  - Unordered trees are an important data type,
    but there are few learning methods for them
    - **Vishwanathan**: using subtrees as features
    - **Kailing**: using heights, degrees or node labels as features

- The Proposed kernel uses subpaths as features
  (=substrings of paths from root to leaves)
  - Tree-structured data often represents hierarchical structures, so vertical structures are important

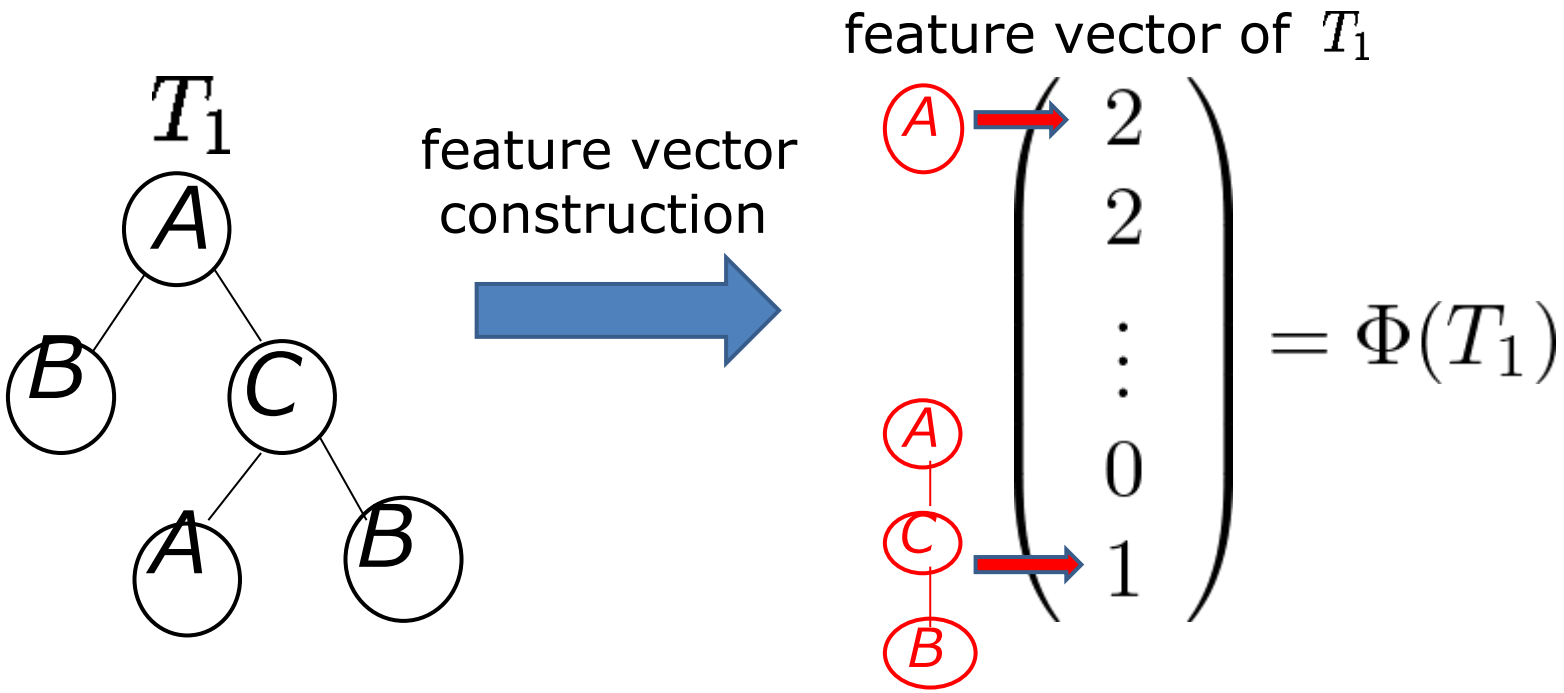
Example of subpath set

subpaths = substrings of paths from root to leaves



Definition of feature vector

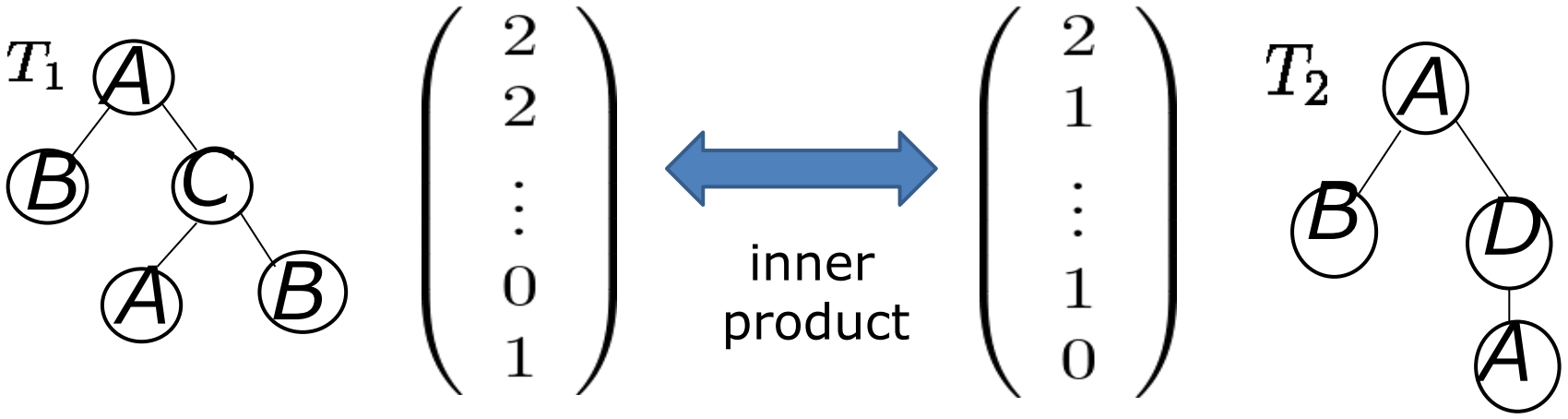
- Each dimension of a feature vector corresponds to the number of subpaths appearing in a tree



Proposed subpath kernel

$$K(T_1, T_2) = \Phi(T_1)^\top \Phi(T_2) = \sum_{p \in S} num(T_{1p}) num(T_{2p})$$

- T_1, T_2 : Input two trees
- S : Subpath set of T_1, T_2
- $num(T_{ip})$: Number of subpath p appearing in T_i

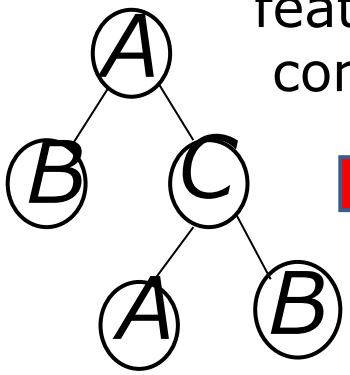


Hardness of computing the kernel

- We need to enumerate all of the common subpaths in T_1, T_2 for computing $K(T_1, T_2)$
- Naive method : enumerating all the subpaths and constructing feature vectors explicitly

⇒ Generally the number of subpaths (= dimensions of feature vector) is $\mathrm{O}(|T|^2)$

Not efficient!



feature vector construction ⇒

$$\begin{pmatrix} 2 \\ 2 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

$\mathrm{O}(|T|^2)$ High dimensions!

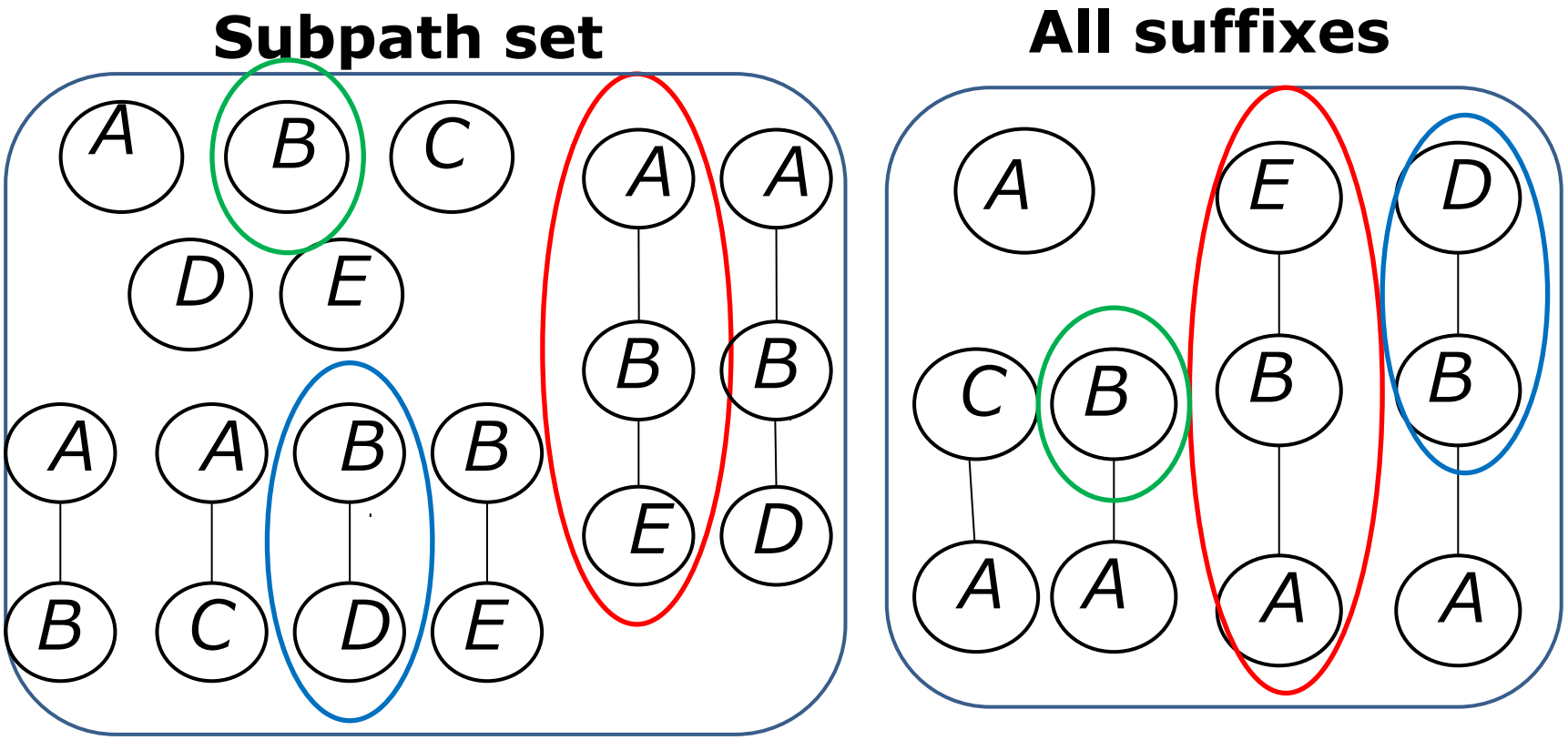
Two ideas for efficient computation

- **1. subpaths = prefixes of all suffixes**
  - suffix of a tree: string starting from one node and ending at the root
  - prefix: substring starting from the beginning and ending at some position

    (ex. prefixes of "ABC" = "A", "AB", "ABC")

- **2. efficient enumeration of prefixes of suffixes by Multikey Quicksort**

A (reversed) supath = prefix of suffix



one-to-one correspondence between a subpath and a prefix of a suffix

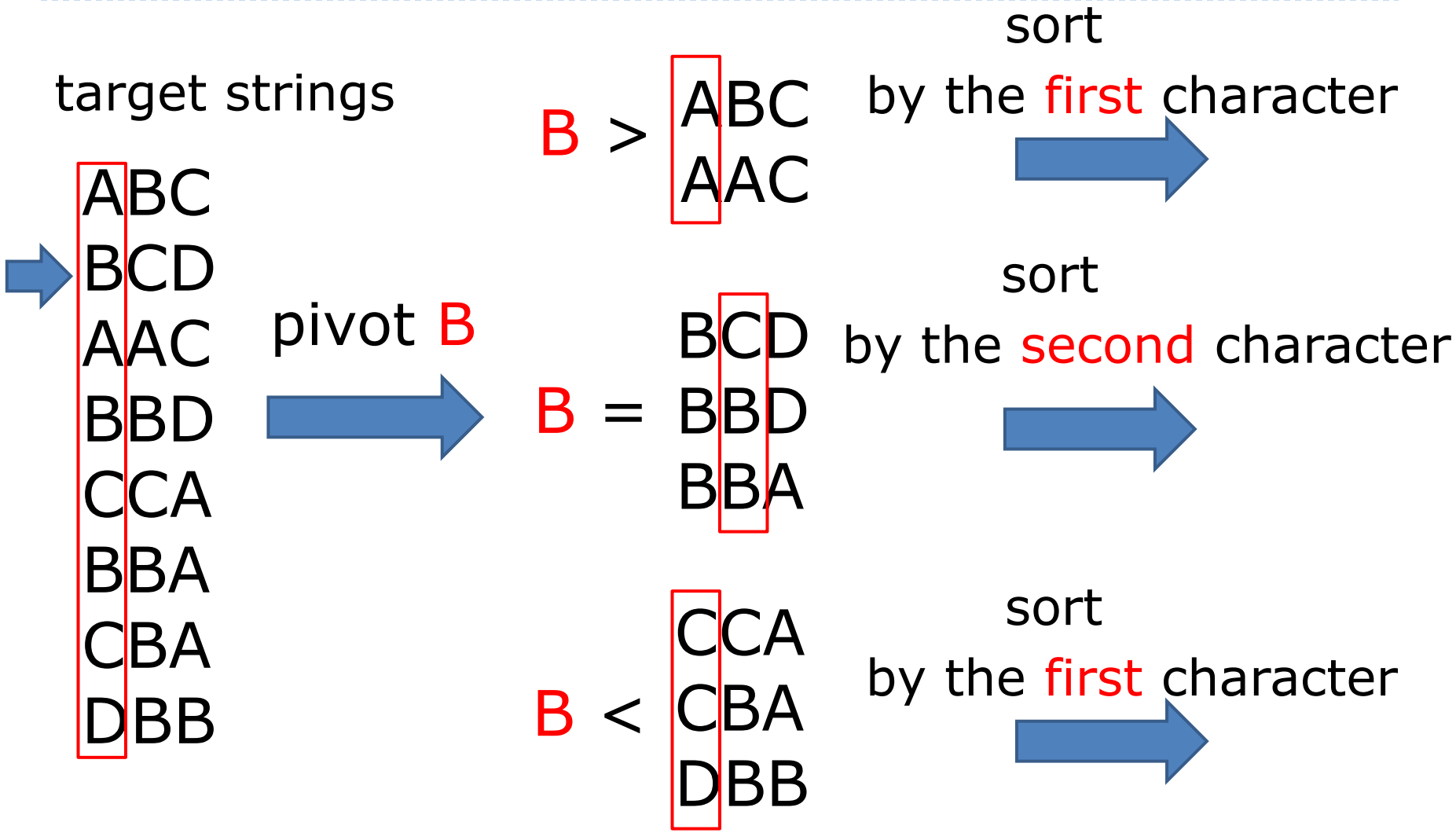
Algorithm of computing the kernel

- We can enumerate **prefixes of suffixes** of two trees efficiently by extending **Multikey Quicksort**

Multikey Quicksort for strings

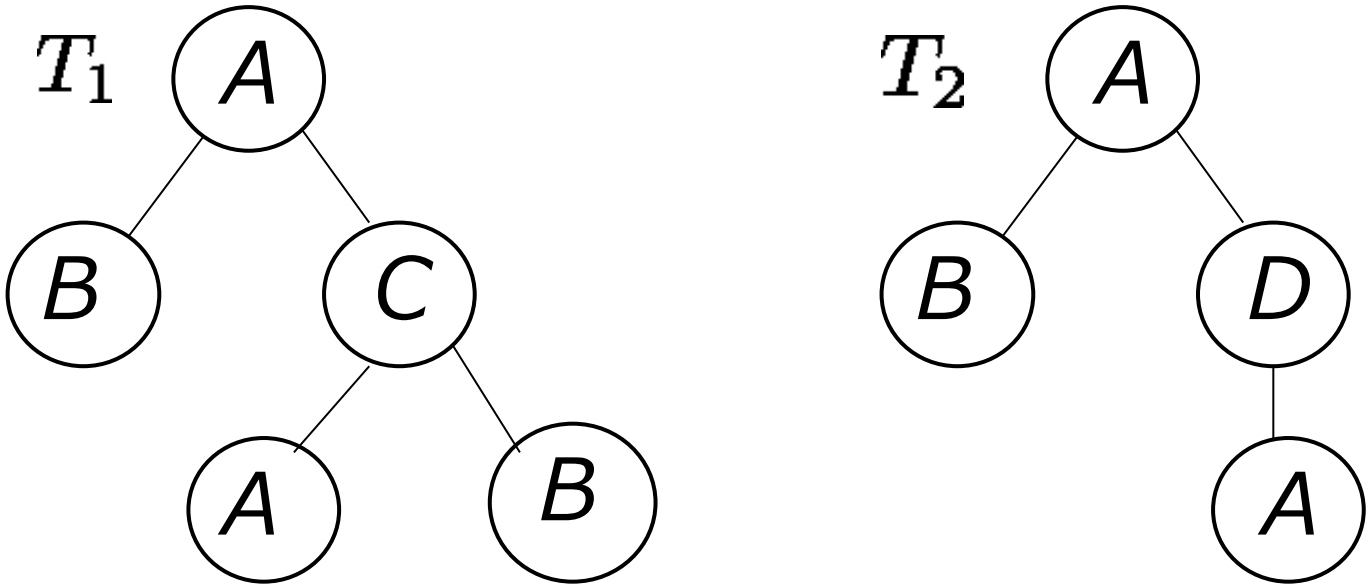
1. Use each character of string as a pivot (from the beginning)
2. Divide the current set into three sets recursively (alphabetically larger, smaller than the pivot and the same with the pivot)
3. For the set with the same character with the pivot, sort them by the next character

Running example of Multikey Quicksort for strings



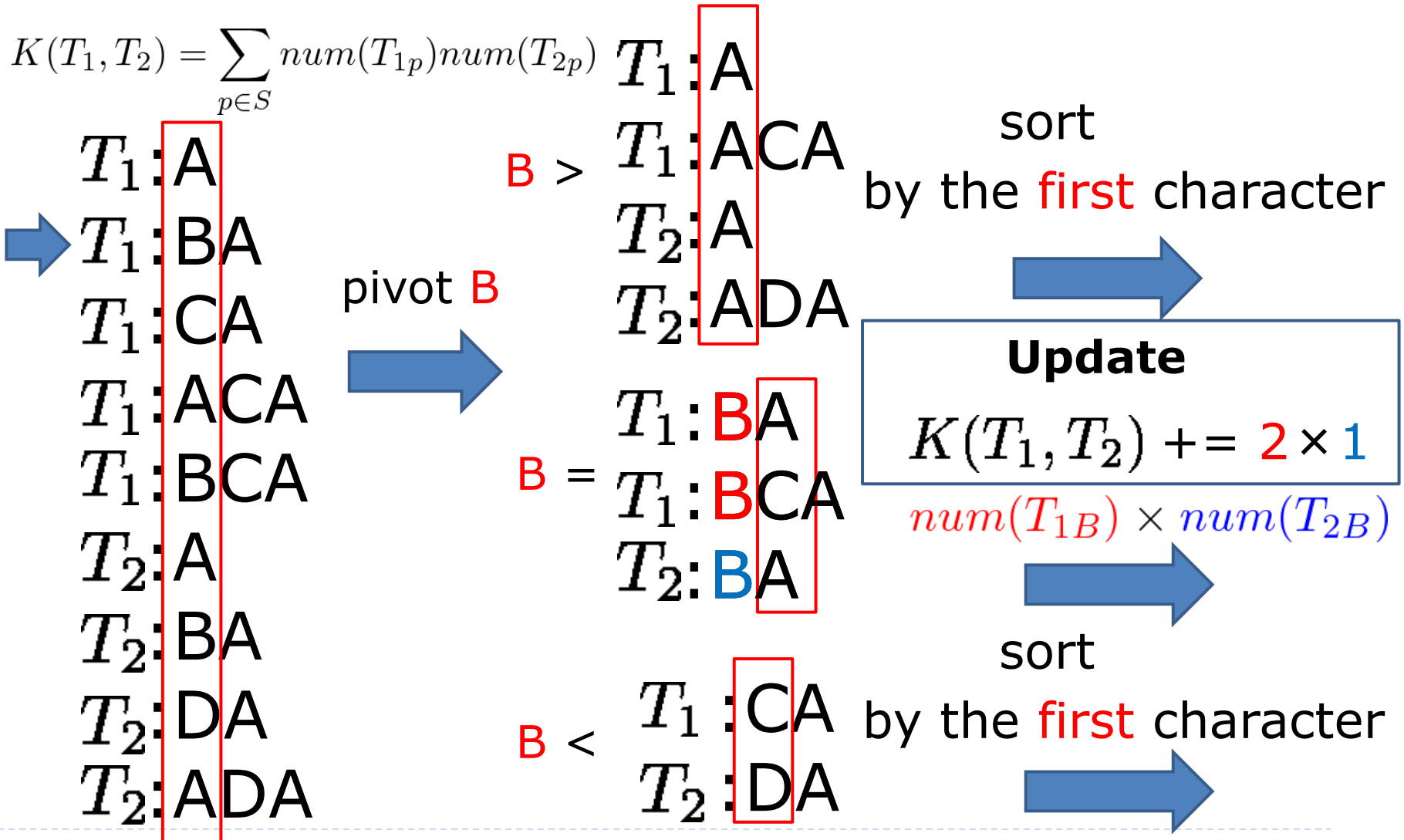
Proposed Algorithm

- Apply Multikey Quicksort to all suffixes of two trees (divide suffixes into three sets recursively)
  - Update the kernel function value **when the current set label is the same with the pivot label**



Running example of the proposed algorithm for two trees

$$K(T_1, T_2) = \sum_{p \in S} num(T_{1p}) num(T_{2p})$$



Time and space complexity of the algorithm

- No need to explicitly store all of the suffixes in memory (only each node label and pointer to parent node)

⇒ space complexity $\mathrm{O}(|T_1| + |T_2|)$

- The time complexity of the algorithm is equal to that of Multikey Quicksort

⇒ average $\mathrm{O}((|T_1| + |T_2|)\log(|T_1| + |T_2|))$

$|T_1|, |T_2|$: the numbers of nodes in T_1 T_2

Experiments

- We compare the predictive performance and running time of the proposed tree kernel with those of the existing tree kernels for binary classfication tasks(two datasets: XML and Glycan)
- The predictive performance is evaluated in Accuracy measured by 10-fold cross-validation
- The running time is compared by the average computation times needed for a single evaluation of a kernel function

Tree kernels in experiments

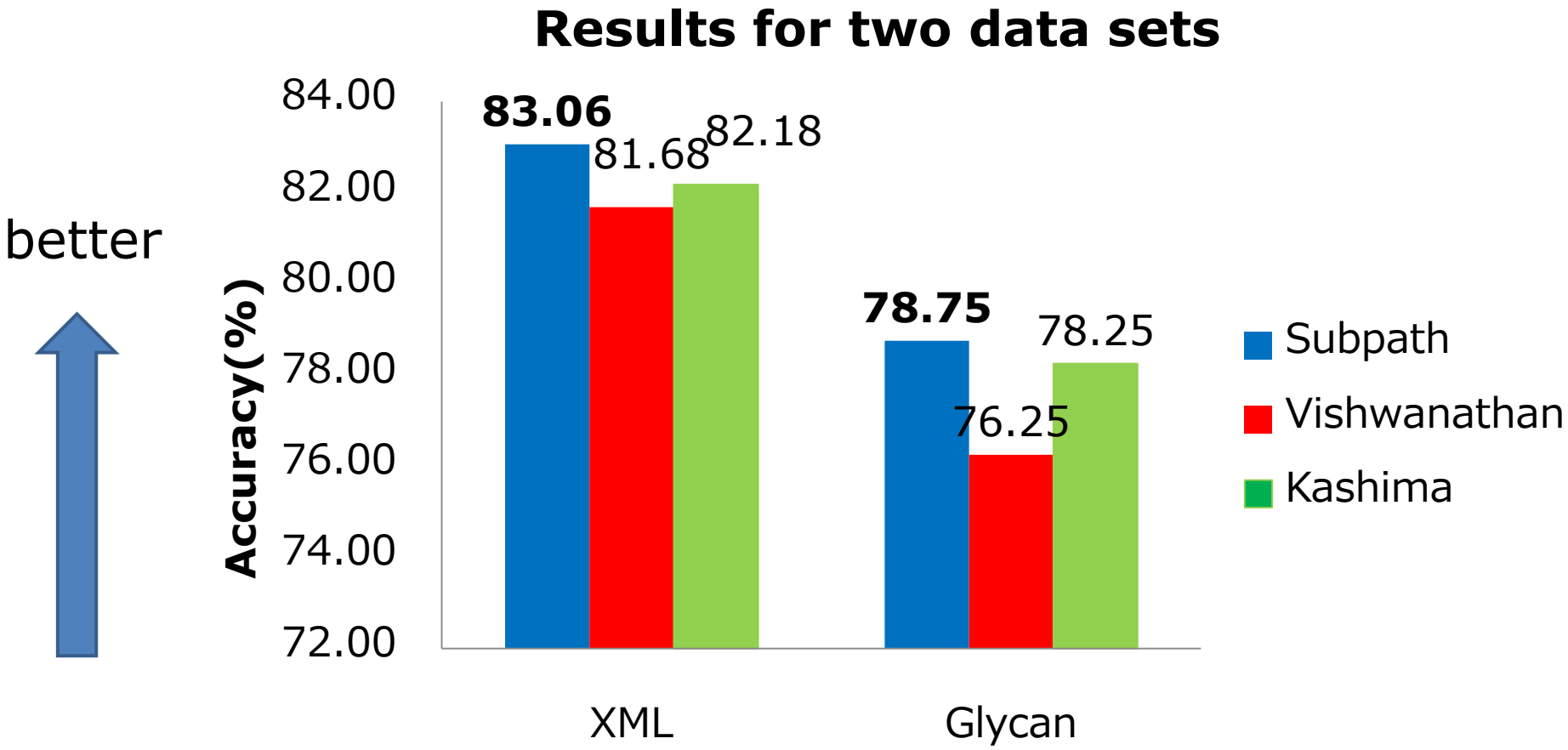
- We compare the following three tree kernels
  - Subpath: our proposed kernel
  - Vishwanathan: a linear-time kernel for unordered trees
  - Kashsima: a standard kernel for ordered trees

	feature	time complexity	for unordered trees?
Subpath	subpath	$\mathrm{O}(\lvert T\rvert \log \lvert T\rvert)$	○
Vishwanathan	subtree	$\mathrm{O}(\lvert T\rvert)^*$	○
Kashima	subset tree	$\mathrm{O}(\lvert T\rvert^2)$	×

- *The $\mathrm{O}(\lvert T\rvert)$ algorithm used in vishwanathan kernel is not faster than the other $\mathrm{O}(\lvert T\rvert \log \lvert T\rvert)$ algorithms in practice.

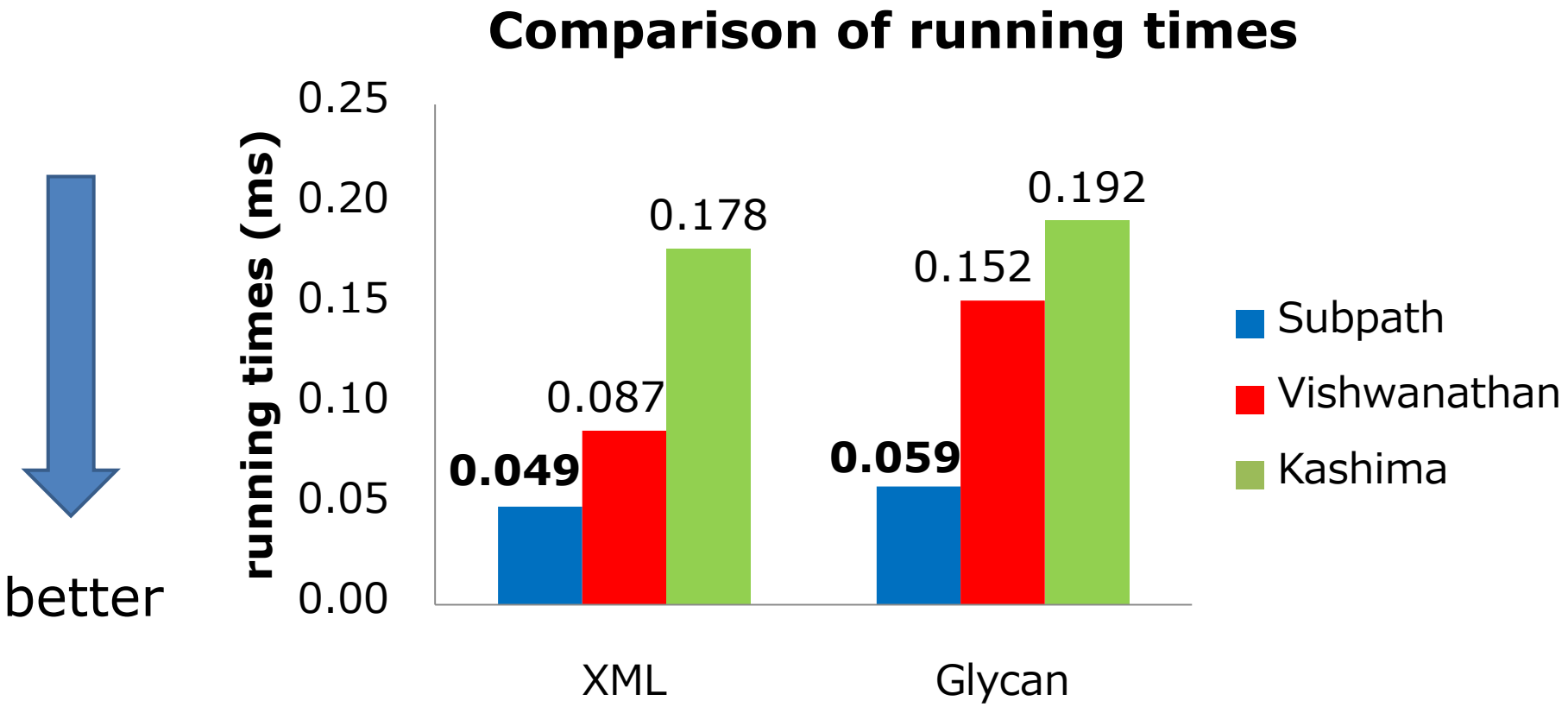
Result (The predictive performance)

- Subpaths are effective as features



Result (The running time)

- Our proposed kernel is faster than the existing linear-time tree kernel



Summary

- We proposed a new tree kernel for **unordered** trees based on **subpaths** and an efficient algorithm for computing the kernel

- The predicting performance of the proposed kernel is **competitive** with the existing tree kernels and **faster** than the linear-time tree kernel in practice

- **Feature Work**
  - Linear-time algorithms (A suffix array for trees)
  - Allowing mismatches of subpaths