Bridging Graph and Kernel Spaces: A Pre-image Perspective

The doctoral defense of

Linlin JIA

LITIS Lab, INSA Rouen Normandie

09/07/2021

Members of jury

Reporters and examiners:

- Mme. Florence D'ALCHÉ-BUC, PR
- M. Donatello CONTE, MCF HDR
- M. Sébastien ADAM, PR
- M. Francesc SERRATOSA, PR
- M. Florian YGER, MCF

Directors of thesis:

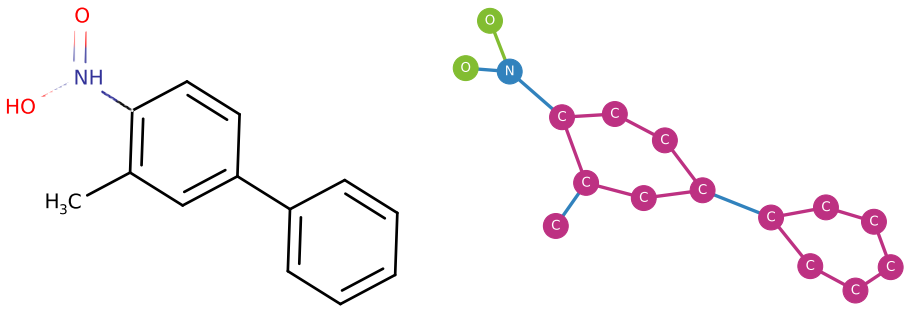
- M. Paul HONEINE, PR
- M. Benoit GAÜZÈRE, MCF

Overview

Overview

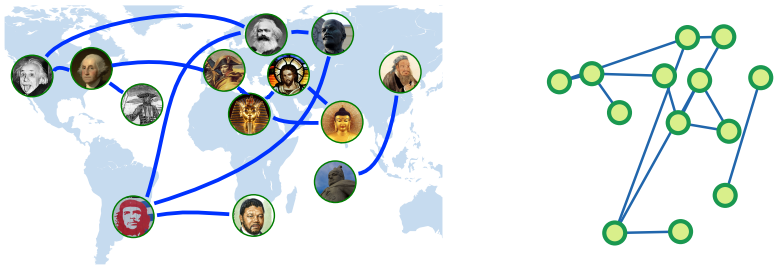
Graph data

chemoinformatics: molecule

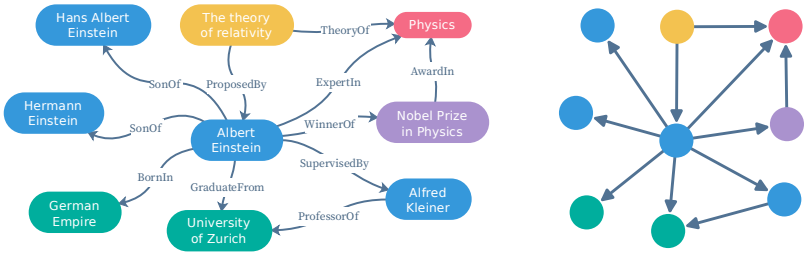


computer vision: 3D point cloud

social media: social network



knowledge graph

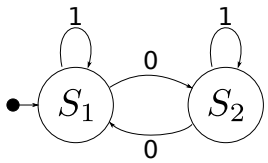


computer vision: handwriting

state transition

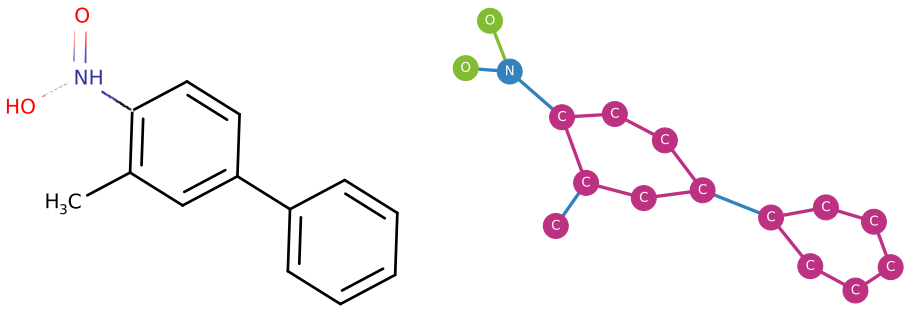
State-transition table

Current state \ Input	0	1
S_1	S_2	S_1
S_2	S_1	S_2



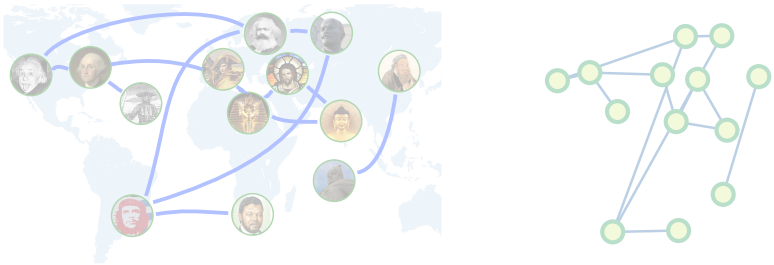
Graph data

chemoinformatics: molecule



computer vision: 3D point cloud

social media: social network



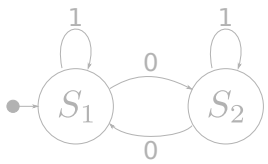
knowledge graph

computer vision: handwriting

state transition

State-transition table

Current state \ Input	0	1
S_1	S_2	S_1
S_2	S_1	S_2



Problematics

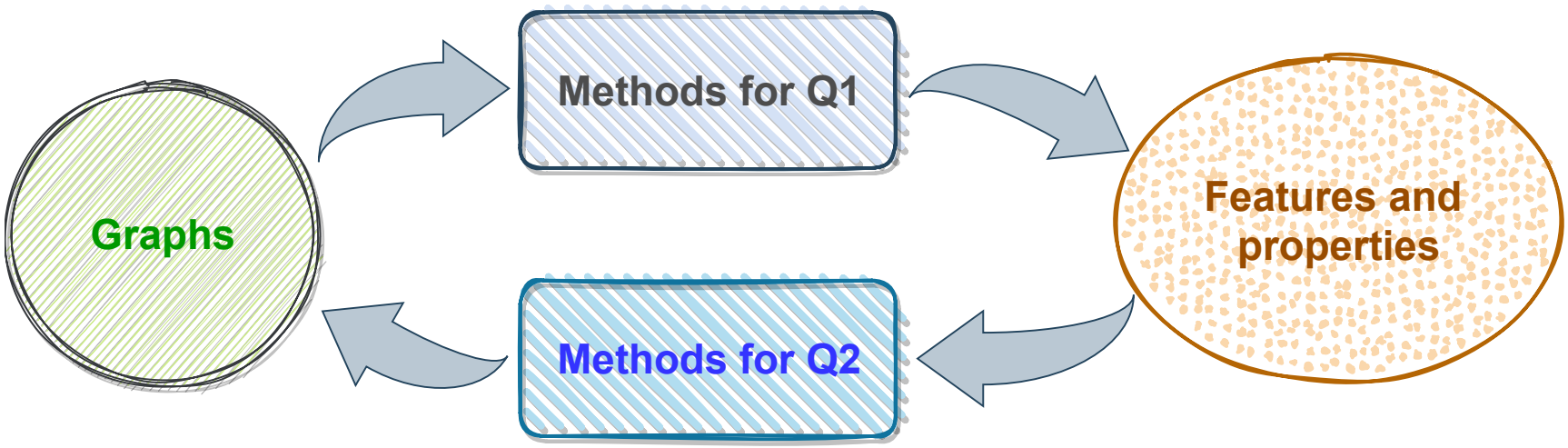
Graph

A graph G is defined by an ordered pair of disjoint sets (V, E), such that V corresponds to a finite set of vertices and $E \subset V \times V$ corresponds to a set of edges.

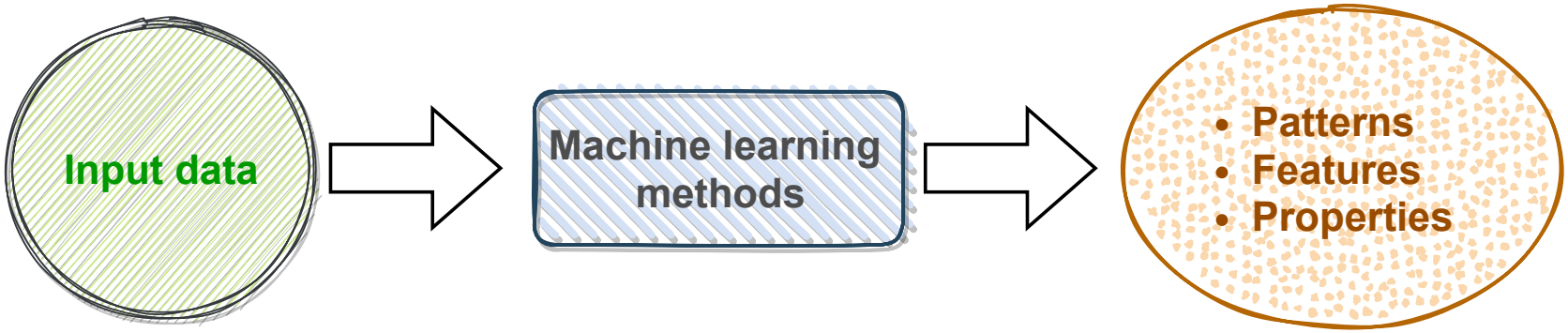
Questions

Q1: How to acquire intrinsic features and properties from graph datasets?

Q2: How to construct graphs endowing desired features and properties?



Machine learning



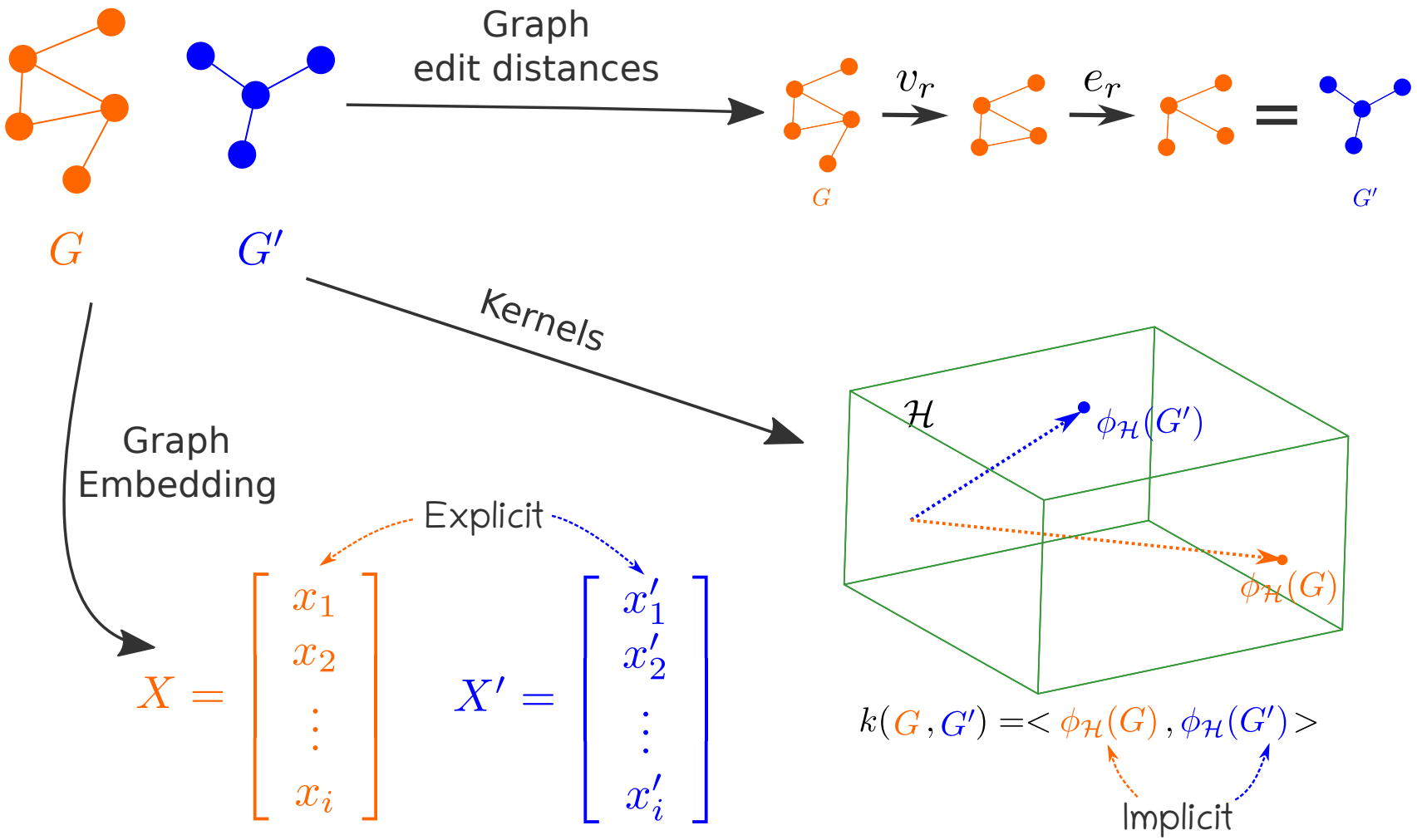
Euclidean space

- It is a continuous space;
- The element dimension is fixed.

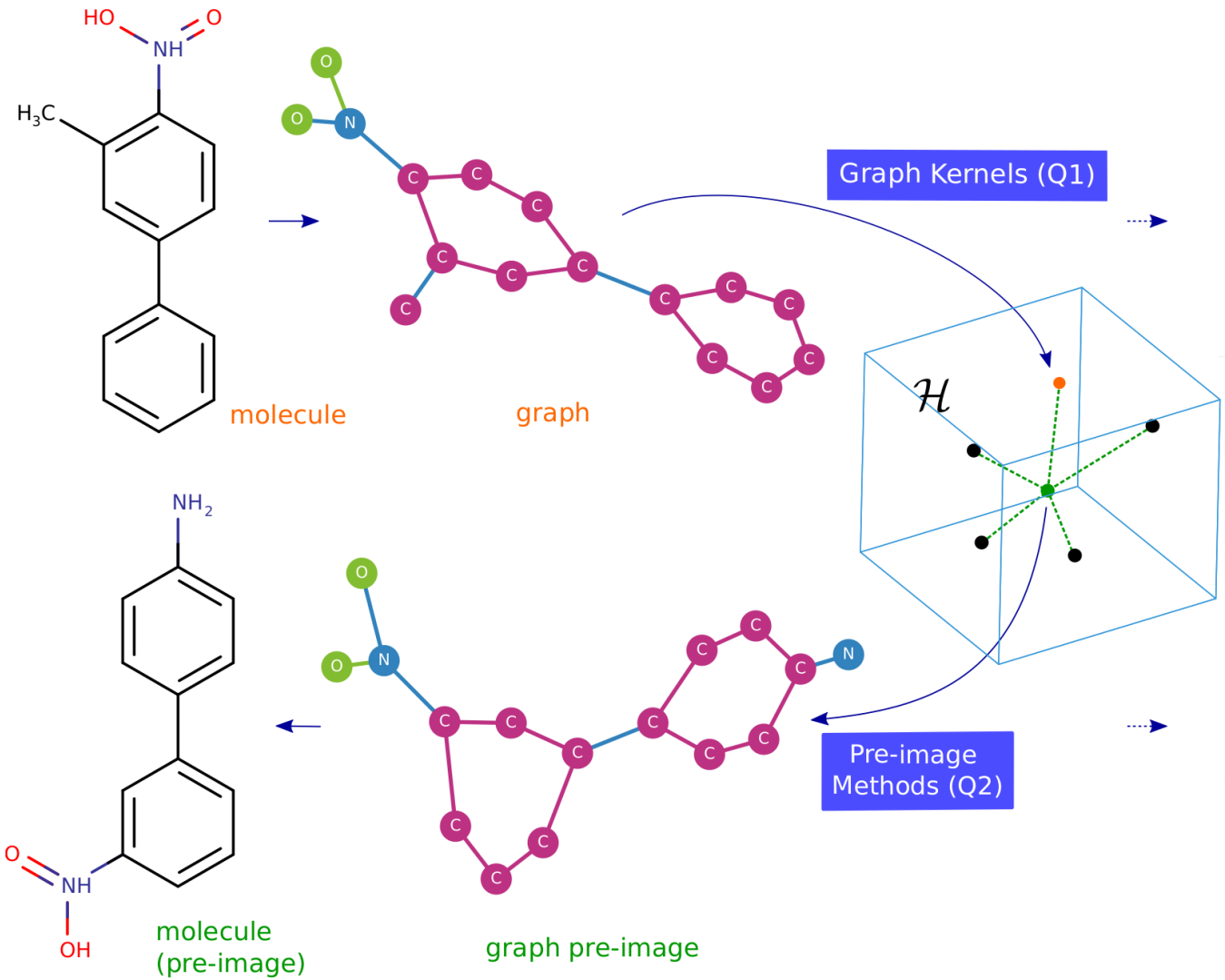
Graph space

- It is a discrete space;
- The numbers of vertices and edges in graphs can be arbitrary;
- Multiple labels and attributes can be plugged into vertices and edges;
- Graph isomorphism.

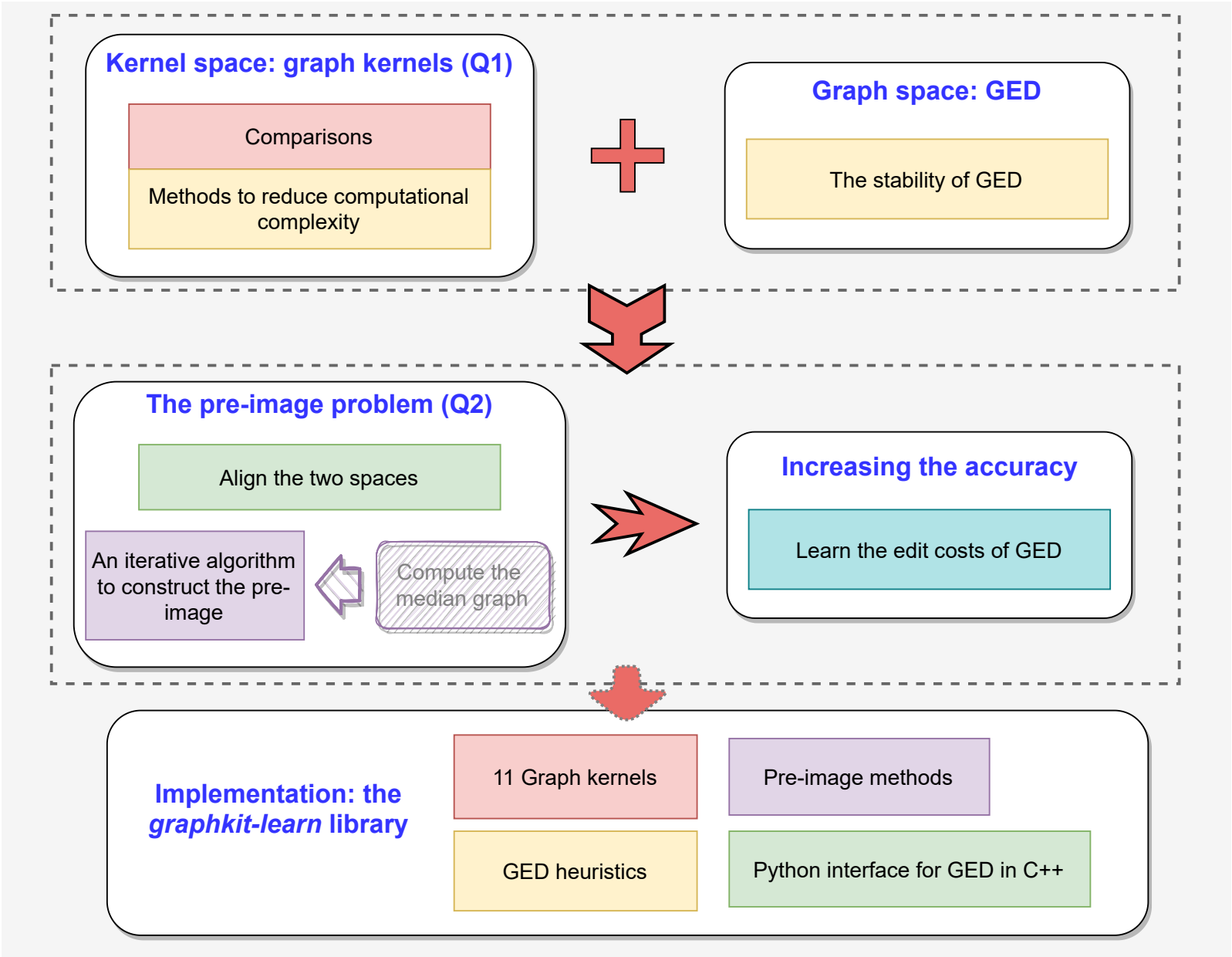
Graph edit distances, graph embedding and graph kernels



Graph kernels and pre-image problem

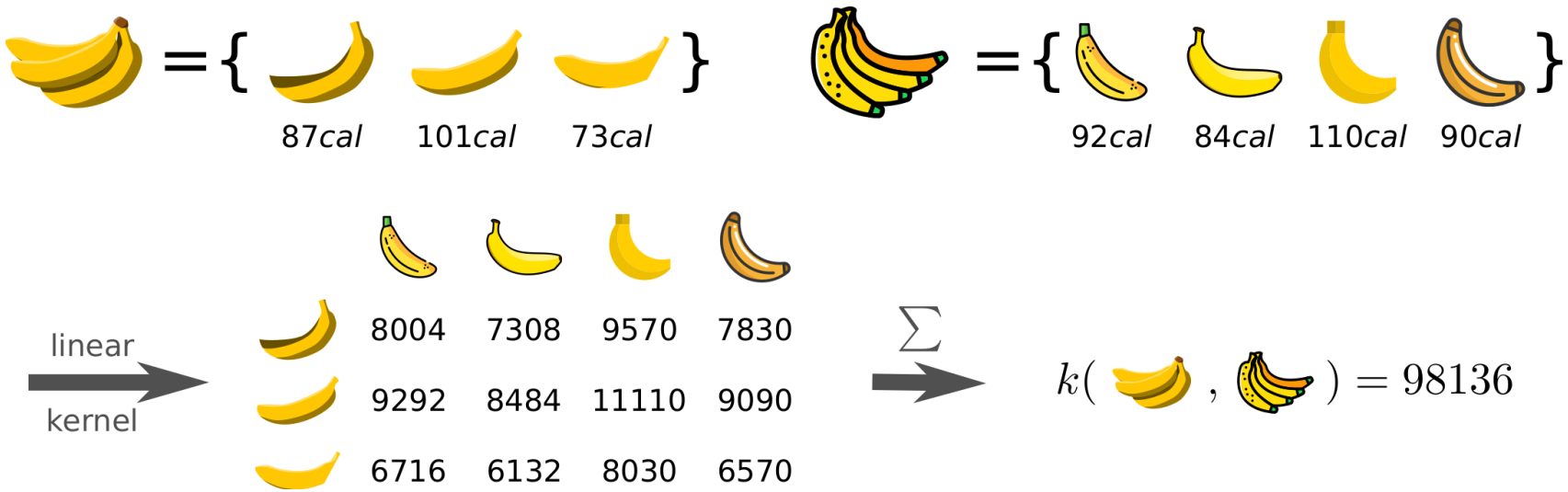


Contributions



Overview

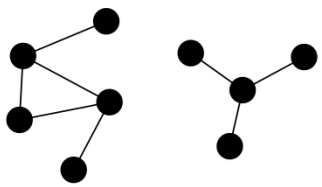
R-convolution kernel[1]



$$k(x_i, x_j) = \sum_{\substack{(x_{i1},\ldots,x_{iD})\in R(x_i)\\(x_{j1},\ldots,x_{jD})\in R(x_j)}} \prod_{d=1}^{D} k_d(x_{id}, x_{jd}).$$

[1] Haussler, Convolution kernels on discrete structures, Tech. report, 1999.

Graph kernels based on sub-patterns



G G'

Kernel

$$k(G, G') = \langle \phi_{\mathcal{H}}(G), \phi_{\mathcal{H}}(G') \rangle$$

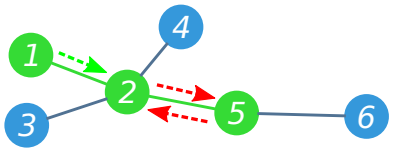
R-convolution kernel

$$k(G, G') = \sum_{\vec{G}, \vec{G'}} \prod_{d=1}^{D} k_d(G_d, G'_d)$$

sub-structures

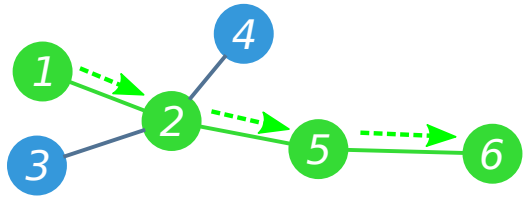
Linear and non-linear graph kernels[1]

Linear patterns based on walks



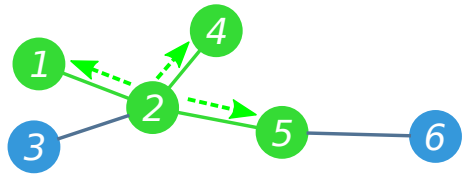
- Common walk kernel
- Marginalized kernel
- Generalized random walk kernel
  - Sylvester equation kernel
  - Conjugate gradient kernel
  - Fixed-point iterations kernel
  - Spectral decomposition kernel

Linear patterns based on paths



- Shortest path kernel
- Structural shortest path kernel
- Path kernel up to length h

Based on non-linear patterns



- Treelet kernel
- Weisfeiler-Lehman subtree kernel

[1] Jia, Gaüzère, and Honeine, Graph Kernels Based on Linear Patterns: Theoretical and Experimental Comparisons. (submitted to *Expert Systems with Applications* in March, 2020)

Graph kernels comparison

Labeling

- **On vertices:** symbolic 9 kernels, non-symbolic 4 kernels;
- **On edges:** symbolic 7 kernels, non-symbolic 3 kernels.

Time complexity to computing Gram matrix

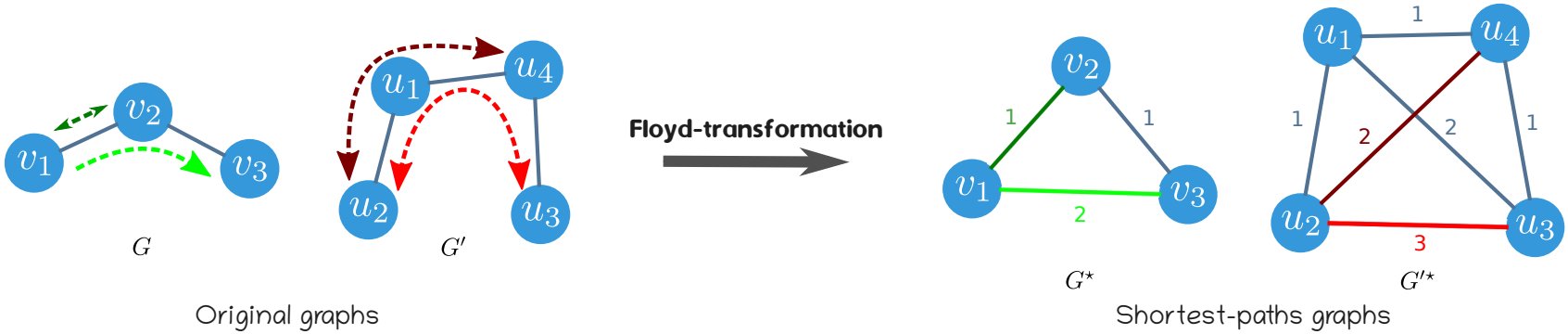
- Worst case: $\mathcal{O}(N^2 n^6)$ (common walk);
- Best case: $\mathcal{O}(N^2 h^2 n^2 d^{2h})$ (path kernel up to length h).

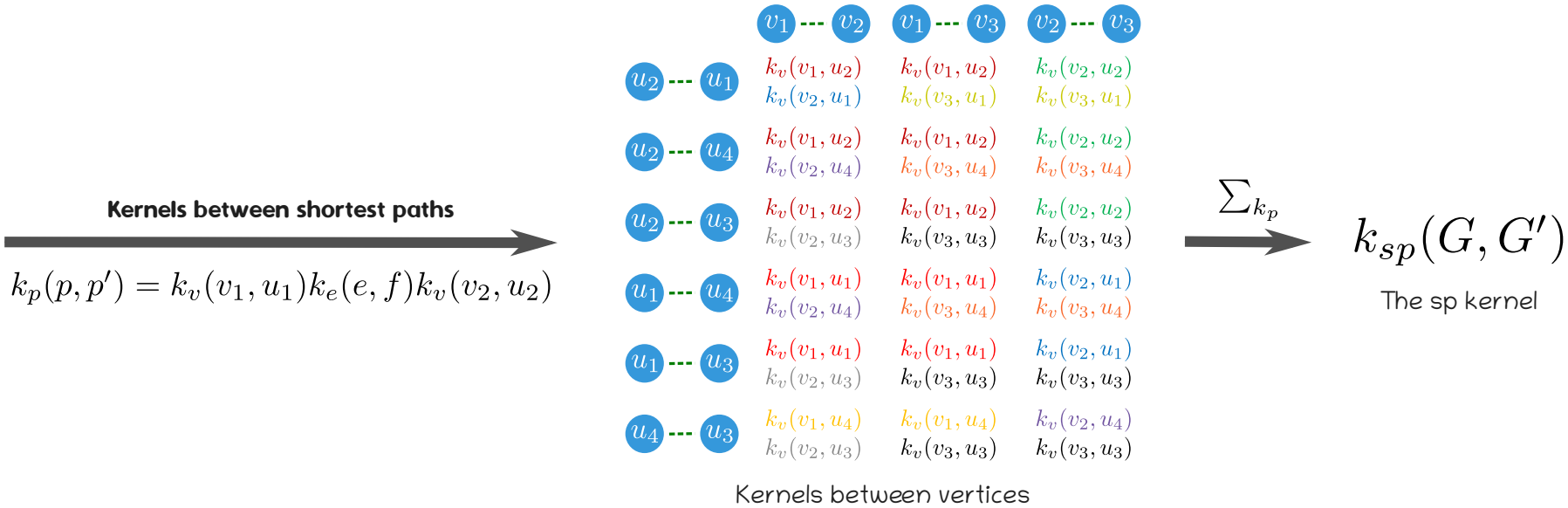
$\longrightarrow$ ***An Achilles' heel!***

Solution: methods to speed up the kernel computation.

The proposed extension of the *Fast Computation of Shortest Path Kernel* (FCSP) method

The shortest-path (sp) kernel

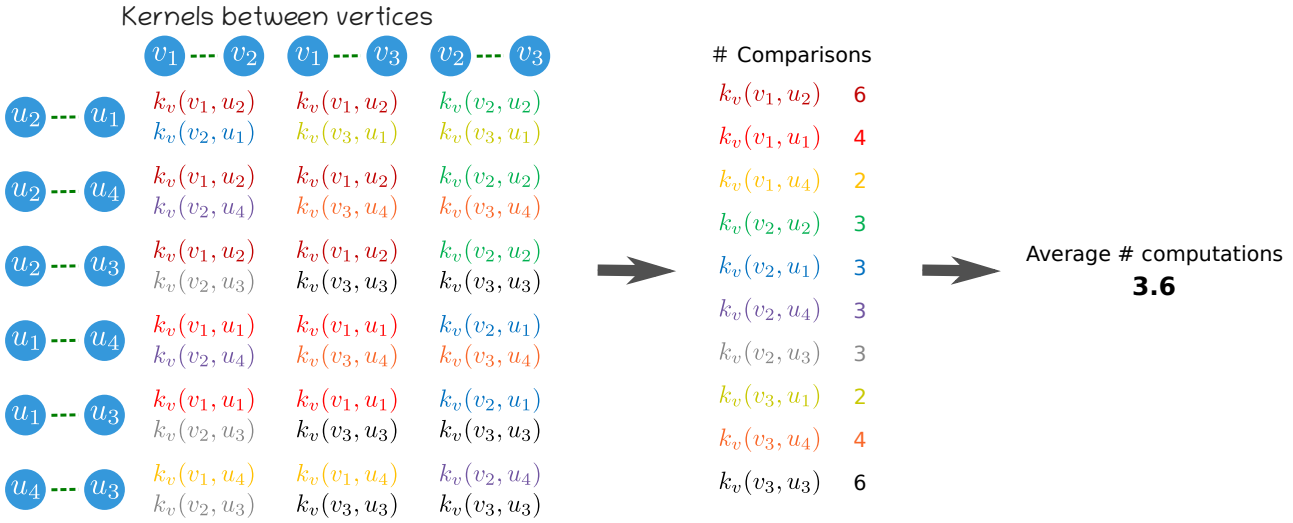




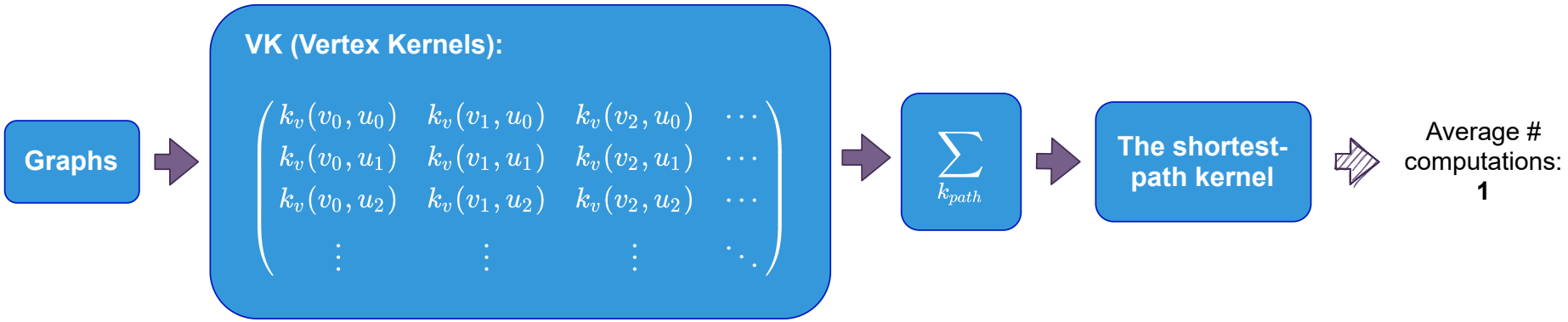
The FCSP method

A new framework

The naive computation:



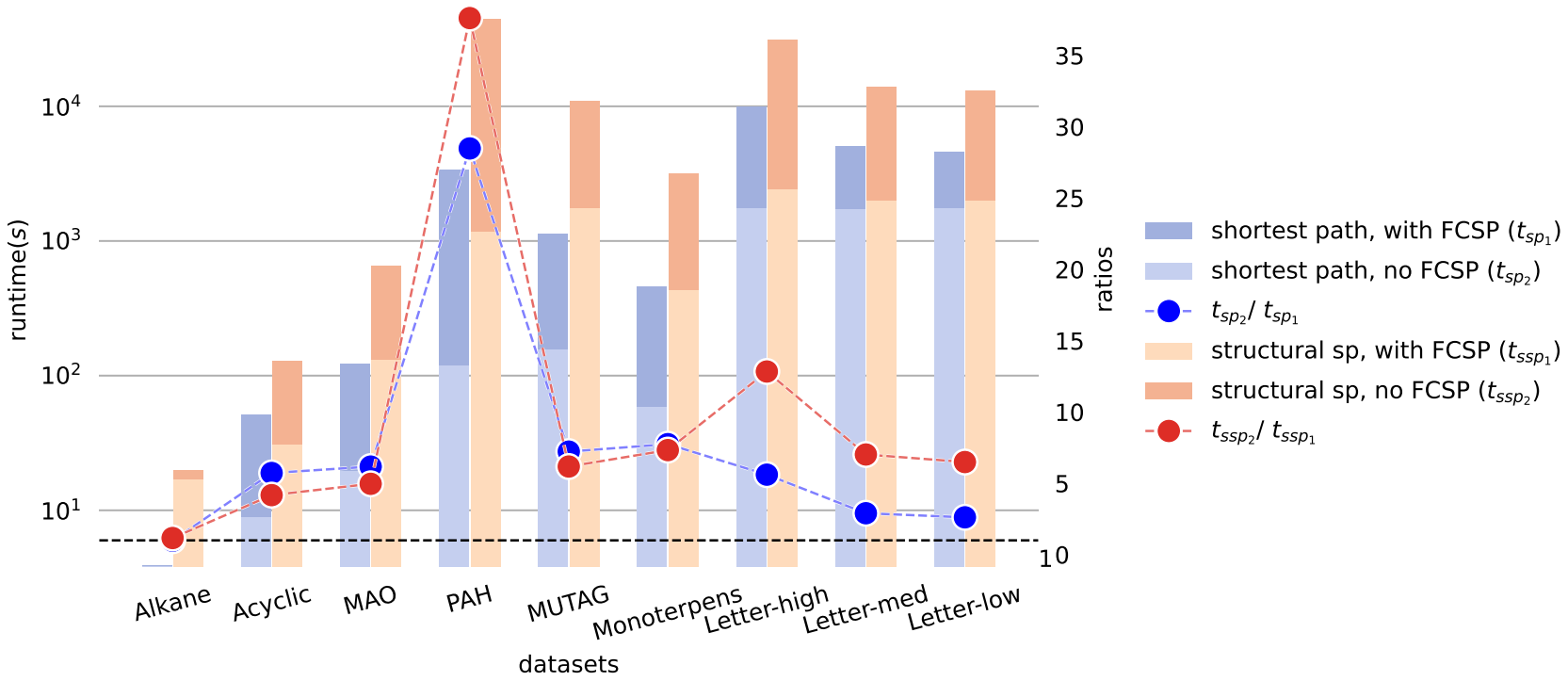
Contribution: computation using FCSP:



Contribution: extend FCSP to edges and structural shortest path kernel.

The FCSP method:

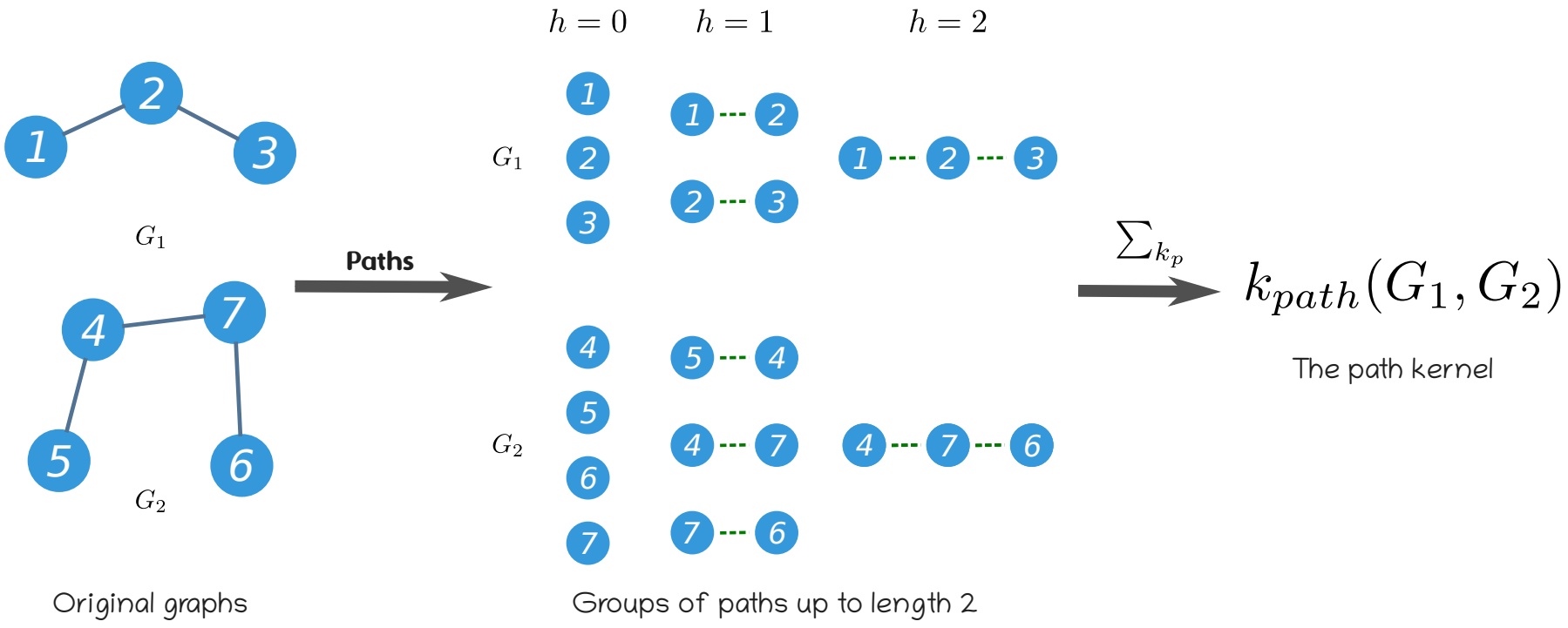
The comparison of runtimes



- The FCSP dominates the performance.
- Best speedups (both on the *PAH* dataset):
  - $\geqslant \mathbf{28}\times$ for the shortest path kernel;
  - $\geqslant \mathbf{37}\times$ for the structural shortest path kernel.

The Trie structure:

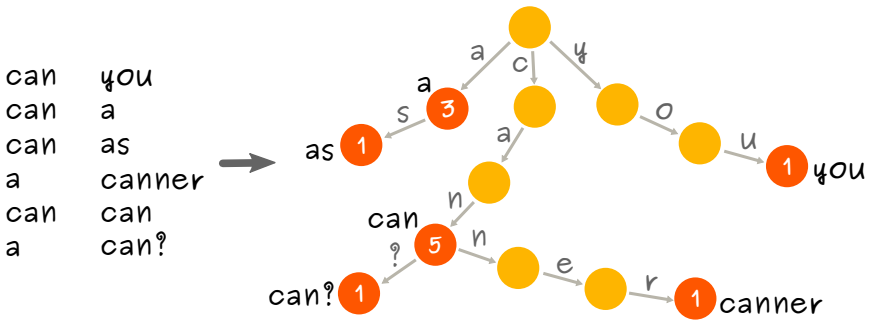
The path kernel up to length h



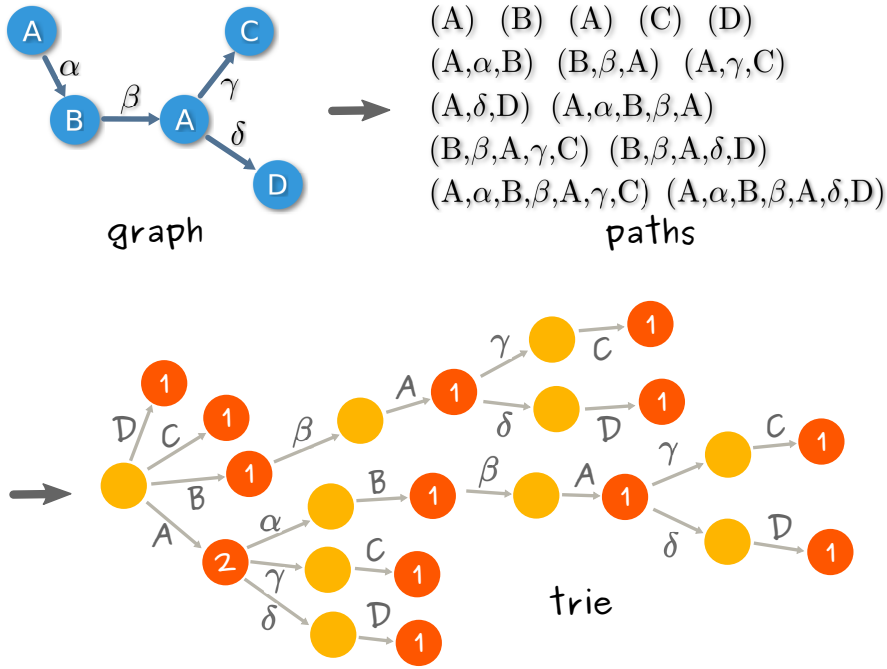
The Trie structure:

The construction of Trie

Store a set of strings in a trie[1]



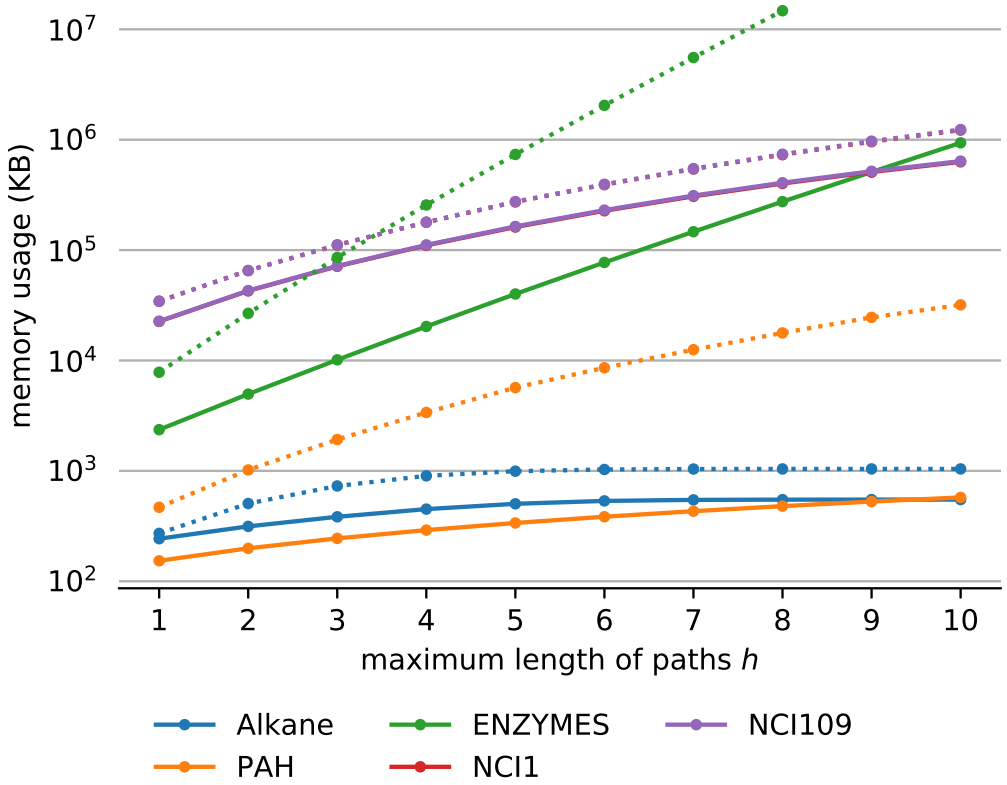
Contribution: construct a trie from paths in a graph



[1] Fredkin, Trie memory, Communications of the ACM 3, 1960.

The Trie structure:

Effect of the trie structure: memory usages



- Dot lines: explicit storing;
- Solid lines: storing using trie.

- Save memory ($\geqslant \mathbf{55}\times$);
- Save time (by avoiding swapping between memory and hard disk).

Performance on real-world datasets

Datasets

Substructures

- **Linear** and **non-linear** for all datasets;
- **Cyclic** for all datasets except *Alkane*, *Acyclic*.

Labeling

- **On vertices:**
  - Symbolic (9): *Alkane*, *Acyclic*, *MAO*, *MUTAG*, *Enzymes*, *AIDS*, *NCI1*, *NCI109*, *DD*;
  - Non-symbolic (3): *Letter-med*, *Enzymes*, *AIDS*.
- **On edges:**
  - Symbolic (3): *MAO*, *MUTAG*, *AIDS*.

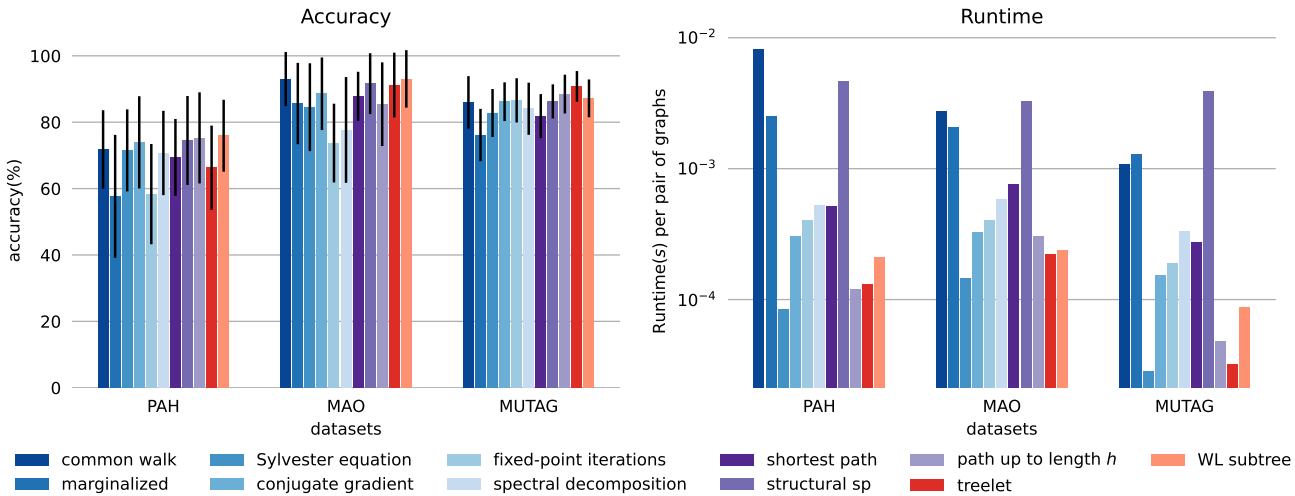
Tasks

- **Regression** (2): *Alkane*, *Acyclic*;
- **Classification** (9): else.

Graph size

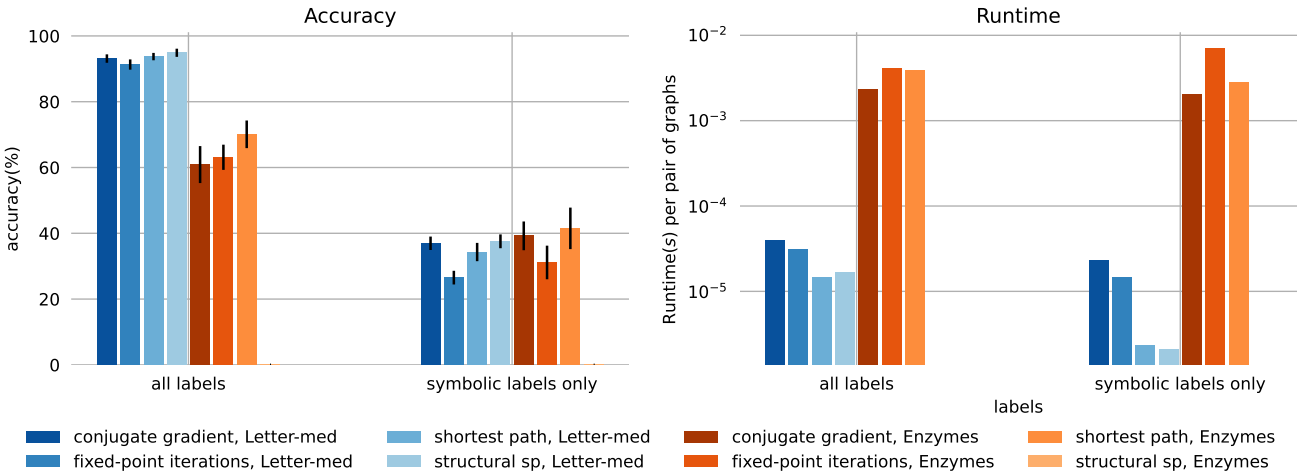
- **Small** (3): *Alkane*, *Acyclic*, *Letter-med*;
- **Medium** (7): *MAO*, *PAH*, *MUTAG*, *Enzymes*, *AIDS*, *NCI1*, *NCI109*;
- **Big** (1): *DD*.

Performance on real-world datasets



Labeled datasets (*MAO*, *MUTAG*) achieve better accuracy than unlabeled ones (*PAH*).

Consider non-symbolic labels when possible leads to one times higher accuracy (*Letter-med*, *Enzymes*).



Performance on real-world datasets

Comparison of kernels based on linear and non-linear patterns

Accuracies achieved by graph kernels

	common walk	marginalized	Sylvester equation	conjugate gradient	fixed-point iterations	spectral decomposition	shortest path	structural SP	path up to length h	treelet	WL subtree
Alkane	15.52	43.75	8.97	11.13	12.78	12.95	7.81	8.65	9.0	2.53	26.42
Acyclic	12.93	18.77	32.5	13.15	14.2	33.05	9.03	13.1	6.66	5.99	19.8
MAO	93.0	85.62	84.52	88.57	73.71	77.67	87.81	91.62	85.43	91.19	93.05
PAH	71.8	57.67	71.5	73.93	58.33	70.73	69.4	74.5	75.27	66.3	75.93
Mutag	85.96	76.11	82.77	86.18	86.58	84.05	81.84	86.26	88.47	90.79	87.18
Letter-med	36.16	5.2	37.27	93.12	91.3	36.38	93.72	94.88	43.83	inf	36.13
Enzymes	42.81	45.92	23.24	60.89	63.11	23.68	70.09	inf	57.49	52.23	50.76
AIDS	94.71	inf	92.42	98.93	98.57	87.21	99.26	98.84	99.65	99.54	98.63
NCI1	inf	inf	59.76	71.34	inf	inf	inf	79.88	84.84	64.84	84.63
NCI109	inf	inf	60.62	67.6	67.25	inf	inf	79.04	83.94	63.46	85.47
D&D	inf	inf	inf	inf	inf	inf	inf	inf	81.4	inf	77.3

In regression error (the upper table) and classification rate (the lower table) estimated on the test set.

Time used to compute Gram matrices of graph kernels

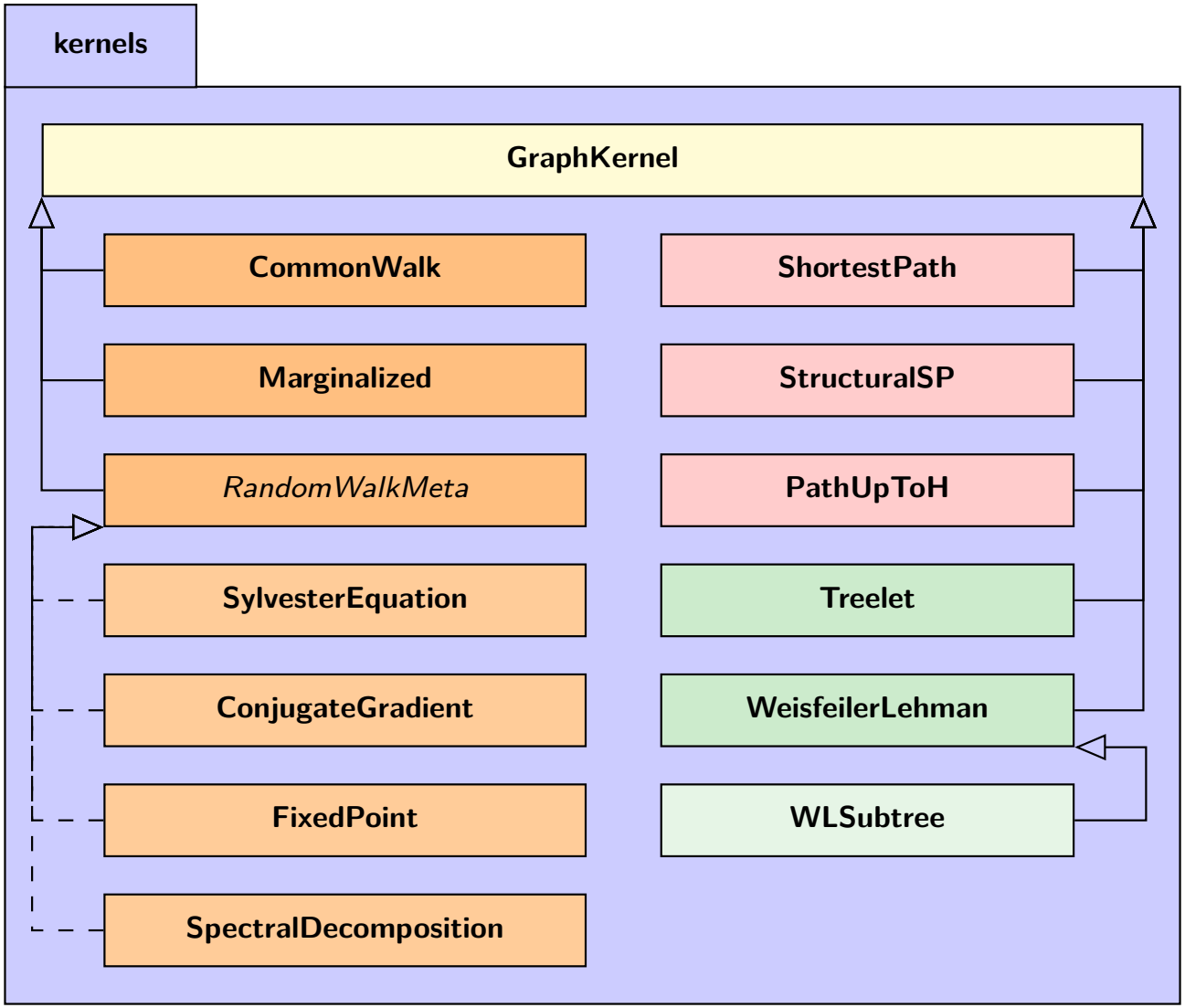
	common walk	marginalized	Sylvester equation	conjugate gradient	fixed-point iterations	spectral decomposition	shortest path	structural SP	path up to length h	treelet	WL subtree
Alkane	0.48	0.51	-0.42	-0.18	-0.22	-0.19	-0.12	0.02	-0.29	-0.3	0.16
Acyclic	0.36	0.62	-0.18	-0.04	-0.11	-0.1	-0.08	0.24	-0.3	-0.31	0.34
MAO	0.81	0.69	-0.47	-0.11	-0.03	0.14	0.25	0.88	-0.14	-0.28	-0.25
PAH	1.56	1.05	-0.42	0.14	0.25	0.37	0.36	1.32	-0.28	-0.24	-0.03
Mutag	1.28	1.36	-0.3	0.44	0.53	0.77	0.69	1.84	-0.07	-0.24	0.19
Letter-med	2.01	2.08	1.13	1.97	1.85	1.78	1.57	1.62	1.08	inf	2.02
Enzymes	3.9	3.18	0.72	2.62	2.79	3.4	2.85	inf	2.16	2.08	1.41
AIDS	2.83	inf	1.37	2.91	3.03	3.74	2.95	3.9	1.59	0.87	2.22
NCI1	inf	inf	2.3	3.96	inf	inf	inf	5.12	2.04	1.48	3.02
NCI109	inf	inf	2.3	4.37	4.38	inf	inf	5.13	2.05	1.48	1.48
D&D	inf	inf	inf	inf	inf	inf	inf	inf	2.67	inf	2.95

In $\log_{10}$ of seconds.

Kernels based on linear patterns achieve success on non-linear structures with competitive accuracy and time complexity to non-linear patterns, especially the kernels based on paths.

Implementations of graph kernels

Implemented graph kernels shown in a UML class diagram



Conclusion

Conclusion on tackling Q1:

- Linear patterns achieve competitive accuracy and time complexity;
- Linear patterns achieve success on non-linear structures;
- Strategies can significantly accelerate the computation;
- Non-symbolic labels matters as well.

And Q2?

Not in graph space, cannot construct graphs directly $\Rightarrow$ use GEDs.

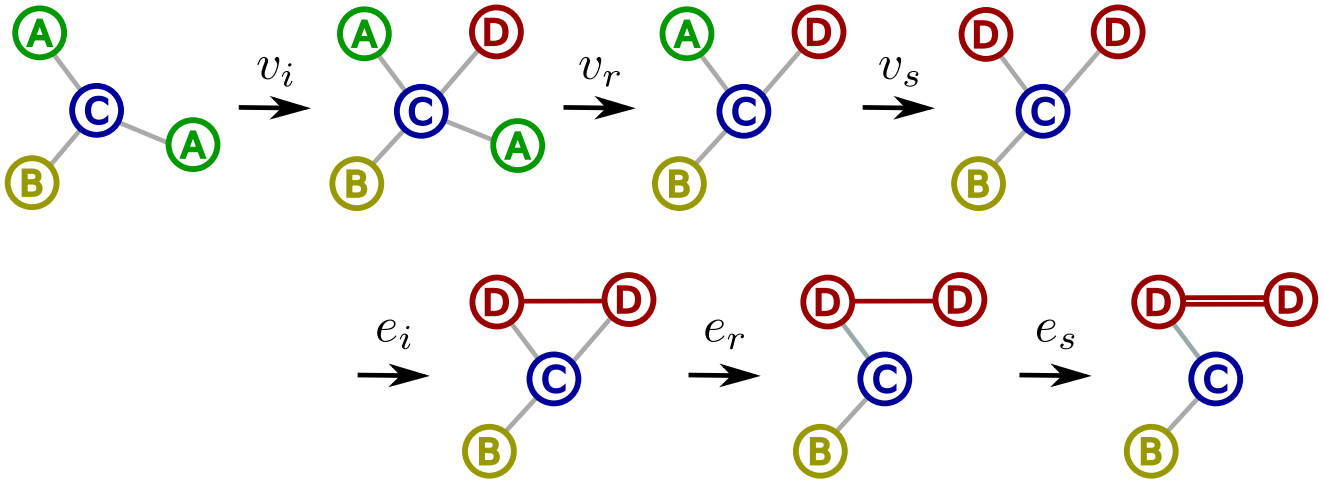
Overview

Graph edit distances

$$\text{edit path } \pi = (v_{i_0}, v_{r_0}, v_{s_0}, v_{s_1}, v_{r_1} \dots)$$

$$\text{ged}(G_1, G_2) = \min_{\pi_1,\dots,\pi_k \in \Pi(G_1,G_2)} \sum_{i=1}^{k} C(\pi_i, G_1, G_2)$$

Graph edit distances



$$\text{edit path } \pi = (v_{i_0}, v_{r_0}, v_{s_0}, v_{s_1}, v_{r_1} \dots)$$

$$\text{ged}(G_1, G_2) = \min_{\pi_1,\dots,\pi_k \in \Pi(G_1,G_2)} \sum_{i=1}^{k} C(\pi_i, G_1, G_2)$$

Graph edit distances:

Edit costs

$$C(\pi, G_1, G_2) = \sum_{\substack{v \in V_2 \\ \pi^{-1}(v) \notin V_1}} c_{vfi}(\varepsilon, v) + \sum_{\substack{u \in V_1 \\ \pi(u) \notin V_2}} c_{vfr}(u, \varepsilon) + \sum_{\substack{u \in V_1 \\ \pi(u) \in V_2}} c_{vfs}(u, \pi(u))$$
$$+ \sum_{e \in \text{inserted edges}} c_{efi}(\varepsilon, e) + \sum_{e \in \text{removed edges}} c_{efr}(e, \varepsilon) + \sum_{e \in \text{substituted edges}} c_{efs}(e, \pi(e))$$

Let edit costs be constants,

then $C(\pi, G_1, G_2) = n_{vr}c_{vr} + n_{vi}c_{vi} + n_{vs}c_{vs} + n_{er}c_{er} + n_{ei}c_{ei} + n_{es}c_{es}$

$$\text{Let} \quad \begin{cases} \boldsymbol{\omega} = [n_{vr}, n_{vi}, n_{vs}, n_{er}, n_{ei}, n_{es}]^\top \\ \boldsymbol{c} = [c_{vr}, c_{vi}, c_{vs}, c_{er}, c_{ei}, c_{es}]^\top \end{cases}$$

$$\Rightarrow \boxed{\text{ged}(G_i, G_j) = \boldsymbol{\omega}^\top \boldsymbol{c}}$$

Graph edit distances

Heuristics to compute GED

Problem: computing GED is NP-hard[1].

Solution: approximate using heuristics:

- Computing GED is a quadratic problem, which can be solved by `IPFP`[2] or m`IPFP`[3] (more accurate).
- Reduce the problem to a linear sum assignment problem with edition or error correction (LSAPE) and then solve it by `bipartite`[4].

New problem induced by heuristics:
Approximations are not optimal, which induce randomness, which causes instability!
Thus the study of the stability is necessary.

[1] Zeng, et al, Comparing stars: On approximating graph edit distance, Proceedings of the VLDB Endowment 2, 2009.

[2] Bougleux, et al, Graph edit distance as a quadratic program, 23rd ICPR, 2016.

[3] Bougleux, et al, Graph edit distance as a quadratic assignment problem, Pattern Recognition Letters, 2017.

[4] Bougleux, et al, Fast linear sum assignment with error-correction and no cost constraints, Pattern Recognition Letters, 2020.

Contribution: the study of stability

The relative error:

$$E_r = \frac{1}{N^2} \sum_{k=1}^{N_t} \frac{\sum_{i,j=1}^{N} \|(\text{ged}^{(k)}(G_i, G_j) - \text{ged}_0(G_i, G_j)\|}{\frac{1}{2}\left(\sum_{i,j=1}^{N} \text{ged}^{(k)}(G_i, G_j) + \text{ged}_0(G_i, G_j)\right)}$$

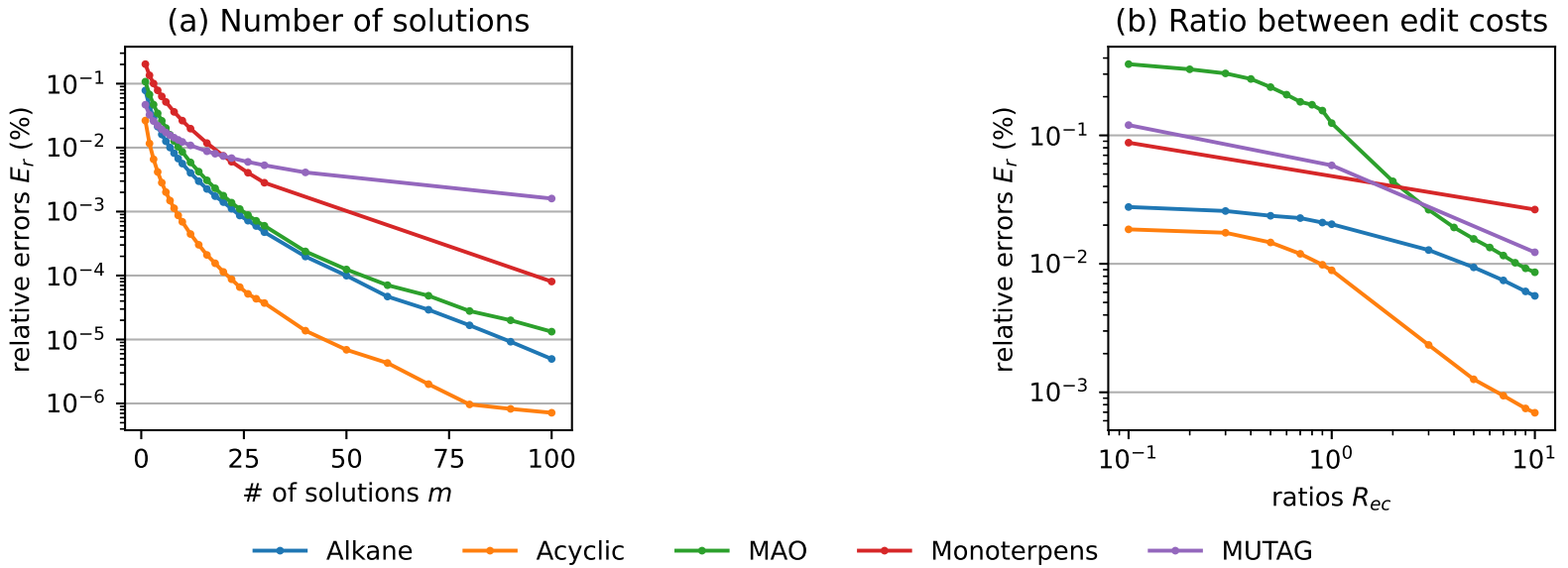
Factors:

1. # of solutions (m):

The number of random initial candidates of the GED heuristic.

2. The ratio between vertex and edge edit costs:

$$R_{ec} = \frac{\texttt{average}(c_{vfi}, c_{vfr}, c_{vfs})}{\texttt{average}(c_{efi}, c_{efr}, c_{efs})}.$$



A metric learning approach to graph edit costs

State-of-the-art methods to learn edit costs

Set manually (expert costs)[1]:

- Domain knowledge is implied a priori.
- Costs are required in datasets.
- It is hard to generalize.
- Data information is not well explored.

Learn costs:

- Edit costs are tuned for each task[2].
- Literature on metric learning for graph data is limited:
  - Some have high computational complexity[3].
  - Some are restricted to strings and trees[4].
  - Some require a ground-truth mapping[5].

Our method:

Learn edit costs by aligning GEDs and distances between targets.

[1] Abu-Aisheh, et al, Graph edit distance contest: Results and future challenges, Pattern Recognition Letters, 2017.

[2] Garcia-Hernandez, et al, Learning the edit costs of graph edit distance applied to ligand-based virtual screening, Current topics in medicinal chemistry, 2020.

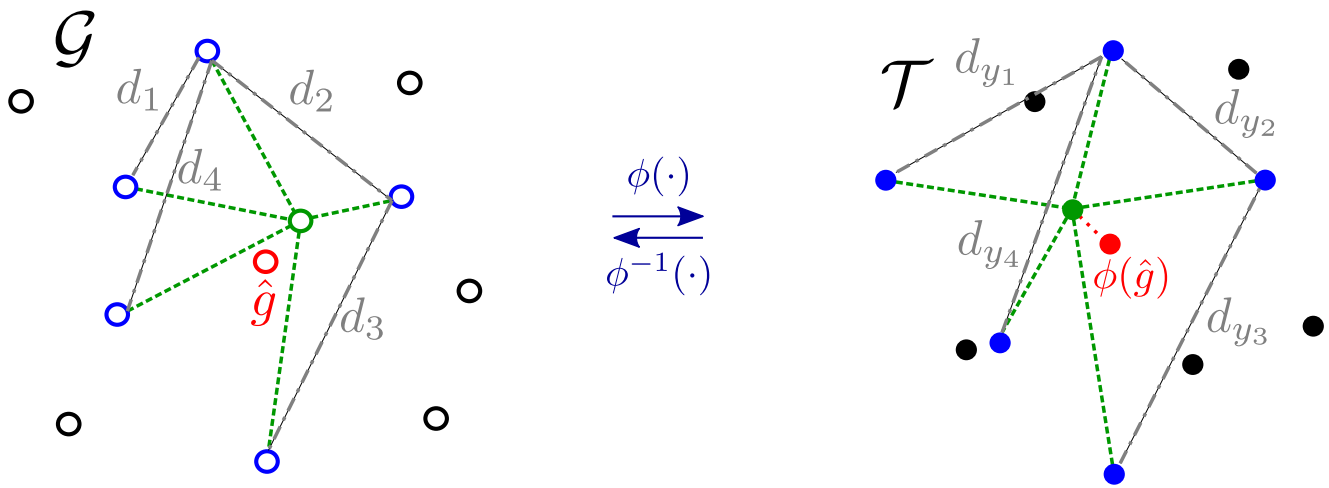
[3] Neuhaus and Bunke, Automatic learning of cost functions for graph edit distance, Information Sciences, 2007.

[4] Bellet, et al, Good edit similarity learning by loss minimization, Machine Learning, 2012.

[5] Cortés, et al, Learning edit cost estimation models for graph edit distance, Pattern Recognition Letters, 2019.

A metric learning approach to graph edit costs

Align graph space and the target space



$$\texttt{ged}_1 = d_{y_1}, \quad \texttt{ged}_2 = d_{y_2}, \quad \texttt{ged}_3 = d_{y_3}, \quad \texttt{ged}_4 = d_{y_4}, \quad \dots$$

$$\begin{cases} \texttt{ged}(G_i, G_j) = \boldsymbol{\omega}^\top \boldsymbol{c} \\ d_y(G_i, G_j) = \|y(G_i) - y(G_j)\|_2 \end{cases}$$

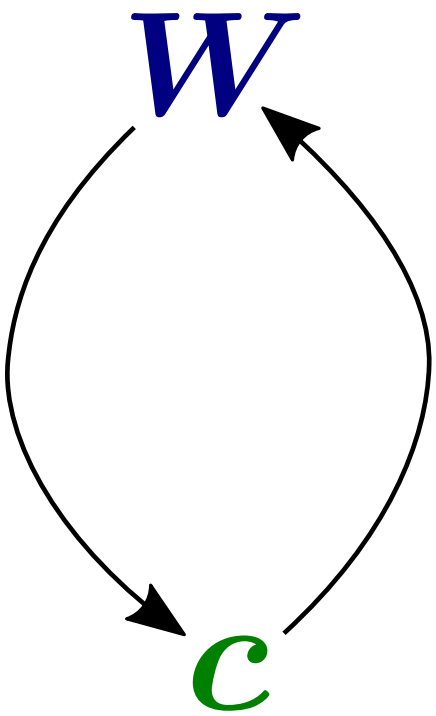
$$\Longrightarrow \quad \boxed{\underset{\boldsymbol{c},\boldsymbol{\omega} \geqslant \mathbf{0}}{\arg\min} \sum_{i,j=1}^{N} \left(\texttt{ged}^{i,j} - d_y^{i,j} \right)^2}$$

A metric learning approach to graph edit costs

An alternated optimization strategy

$$\arg\min_{\boldsymbol{c}\geqslant\mathbf{0}} \quad \|\boldsymbol{W}^\top \boldsymbol{c} - \boldsymbol{d}_y\|^2$$

`CVXPY, scipy`



$$\forall\, G_i, G_j,$$
$$\texttt{ged}(G_i, G_j) = \boldsymbol{\omega}(i,j)^\top \boldsymbol{c}$$

`bipartite, IPFP`

$$\Longrightarrow \quad \boldsymbol{c}_{optimized}$$

- $\boldsymbol{W}^\top$: the N^2-by-6 matrix with rows $\boldsymbol{\omega}(i,j)^\top$;
- $\boldsymbol{d}_y$: the vector of N^2 entries $d_y(G_i, G_j)$, for $i, j = 1, \ldots, N$.

A metric learning approach to graph edit costs

Experiment settings

Datasets:

- Molecules and their boiling points as targets:
  - The `Alkane` dataset: unlabeled;
  - The `Acyclic` dataset: nodes with discrete labels.

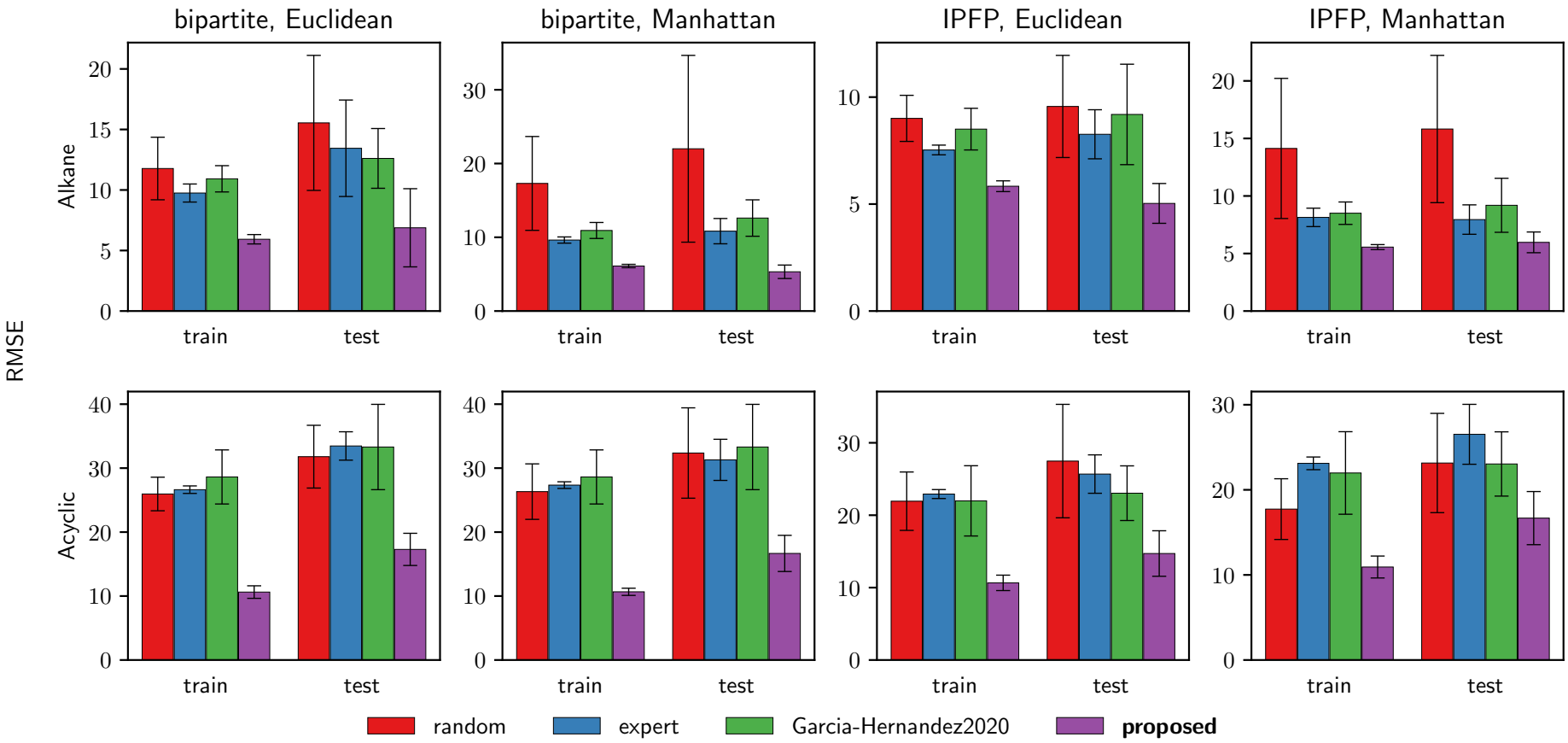
Configurations:

- K-NN regression problem;
- m`bipartite` and m`IPFP` with $m = 40$;
- Outer CV: 90% training + 10% test, inner CV: 5-fold;
- k optimized in inner CV over $\{3, 5, 7, 9, 11\}$;
- Maximum number of iterations: 5.

A metric learning approach to graph edit costs

Performance

Results on each dataset in terms of RMSE for the 10 splits, measured on the training and on the test sets (compared to [1] which uses genetic algorithms to optimize edit costs):



[1] Garcia-Hernandez, et al, Learning the edit costs of graph edit distance applied to ligand-based virtual screening, Current topics in medicinal chemistry, 2020.

A metric learning approach to graph edit costs

Performance

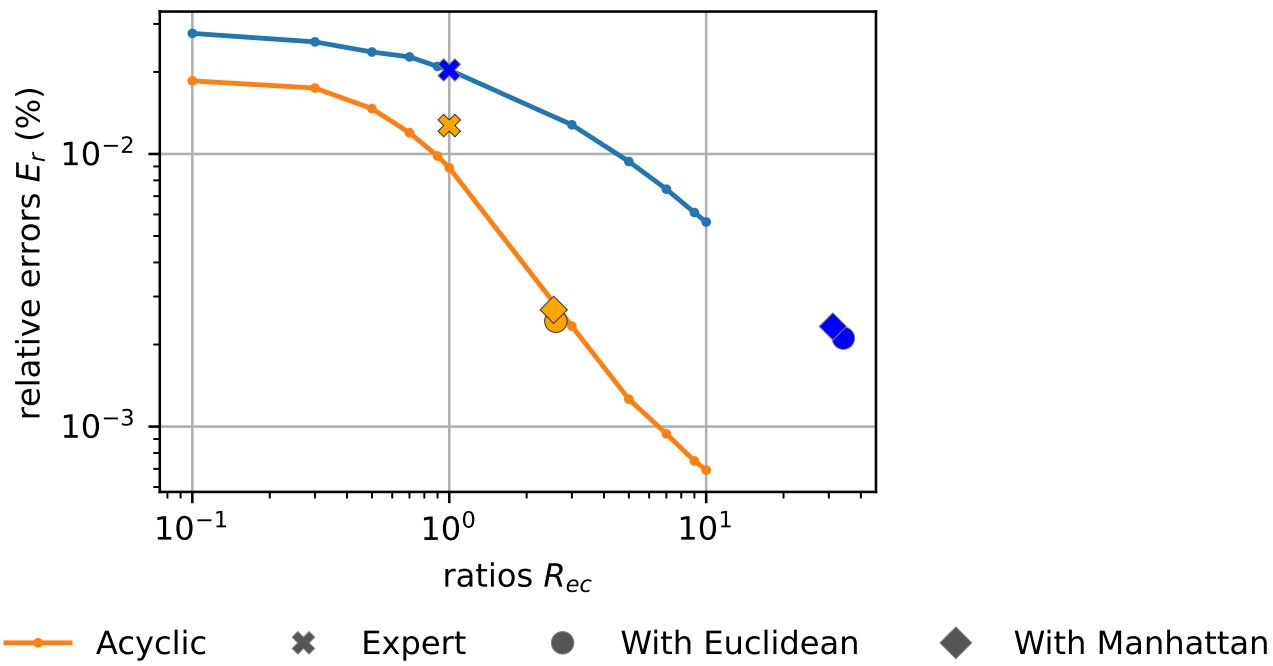
Table: Average and standard deviation of fitted edit costs values

Dataset	Edit cost	Distance	c_{ni}	c_{nr}	c_{ns}	c_{ei}	c_{er}	c_{es}
Alkane	bipartite	Euclidean	26.45±0.48	26.24±0.60	-	0.13±0.06	0.14±0.09	-
		Manhattan	26.67±0.37	26.63±0.58	-	0.11±0.04	0.11±0.06	-
	IPFP	Euclidean	26.12±0.24	25.88±0.25	-	0.74±0.23	0.78±0.23	-
		Manhattan	25.94±0.38	25.71±0.44	-	0.89±0.30	0.77±0.29	-
Acyclic	bipartite	Euclidean	13.81±0.48	13.83±0.80	10.46±0.40	1.37±0.46	1.45±0.46	1.41±0.09
		Manhattan	13.76±0.39	14.14±0.57	10.28±0.44	1.44±0.20	1.45±0.19	1.45±0.07
	IPFP	Euclidean	11.61±0.45	11.68±0.43	11.07±0.53	4.49±0.30	4.46±0.24	4.48±0.18
		Manhattan	11.52±0.40	11.40±0.40	10.61±0.52	4.50±0.31	4.50±0.31	4.50±0.10

- **Insertion and deletion costs are almost similar**, hence showing the symmetry of these operations.
- **Deletion and insertion costs are more important** than substitution costs, which shows that the number of atoms is more important than the atom itself.

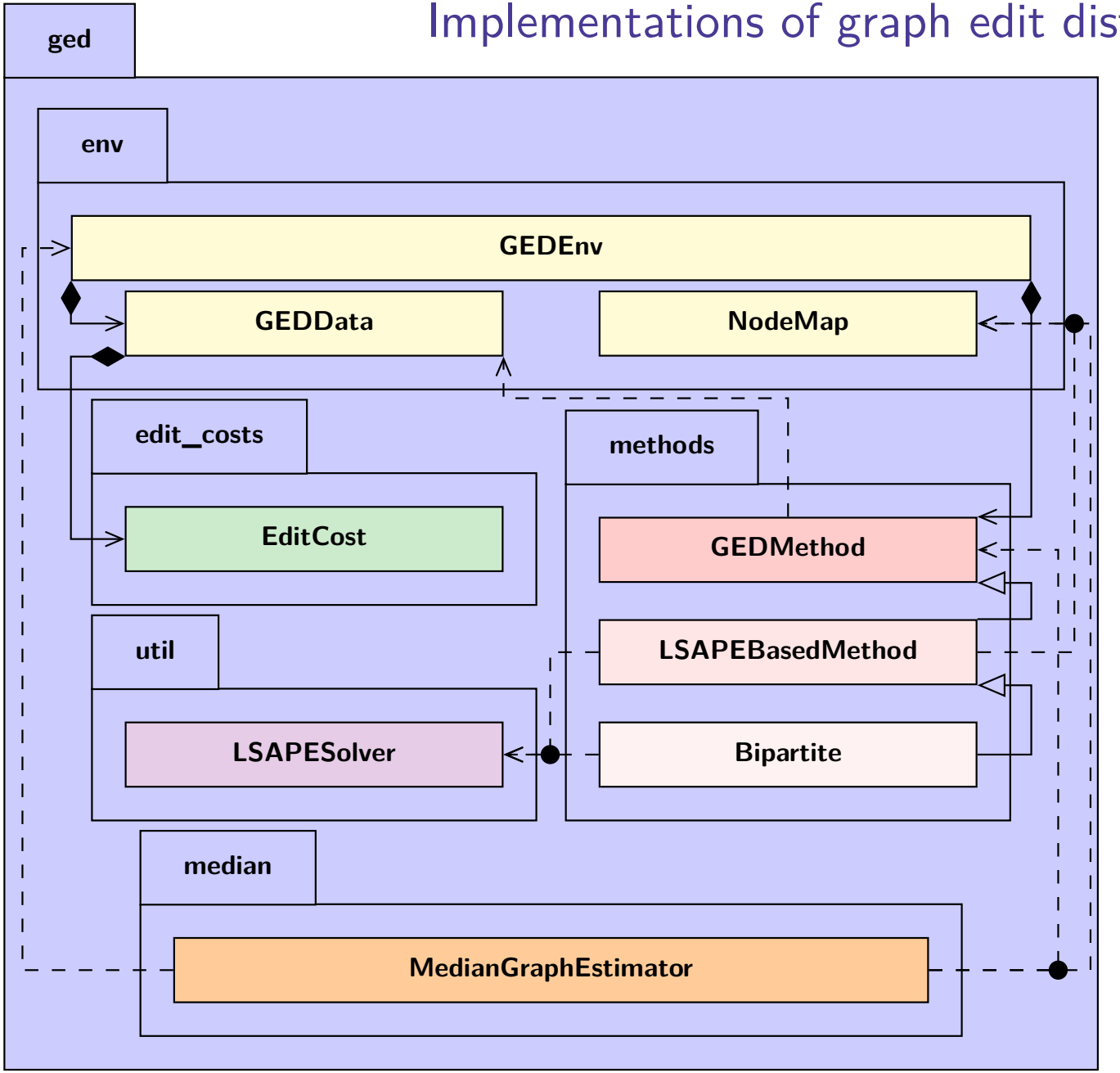
A metric learning approach to graph edit costs

Relation to stability



- The optimized costs *v.s.* the expert costs: higher ratios $\Leftrightarrow$ lower relative errors;
- Optimized costs $\Rightarrow$ higher stability of GEDs.

Implementations of graph edit distance



Conclusion

Conclusion:

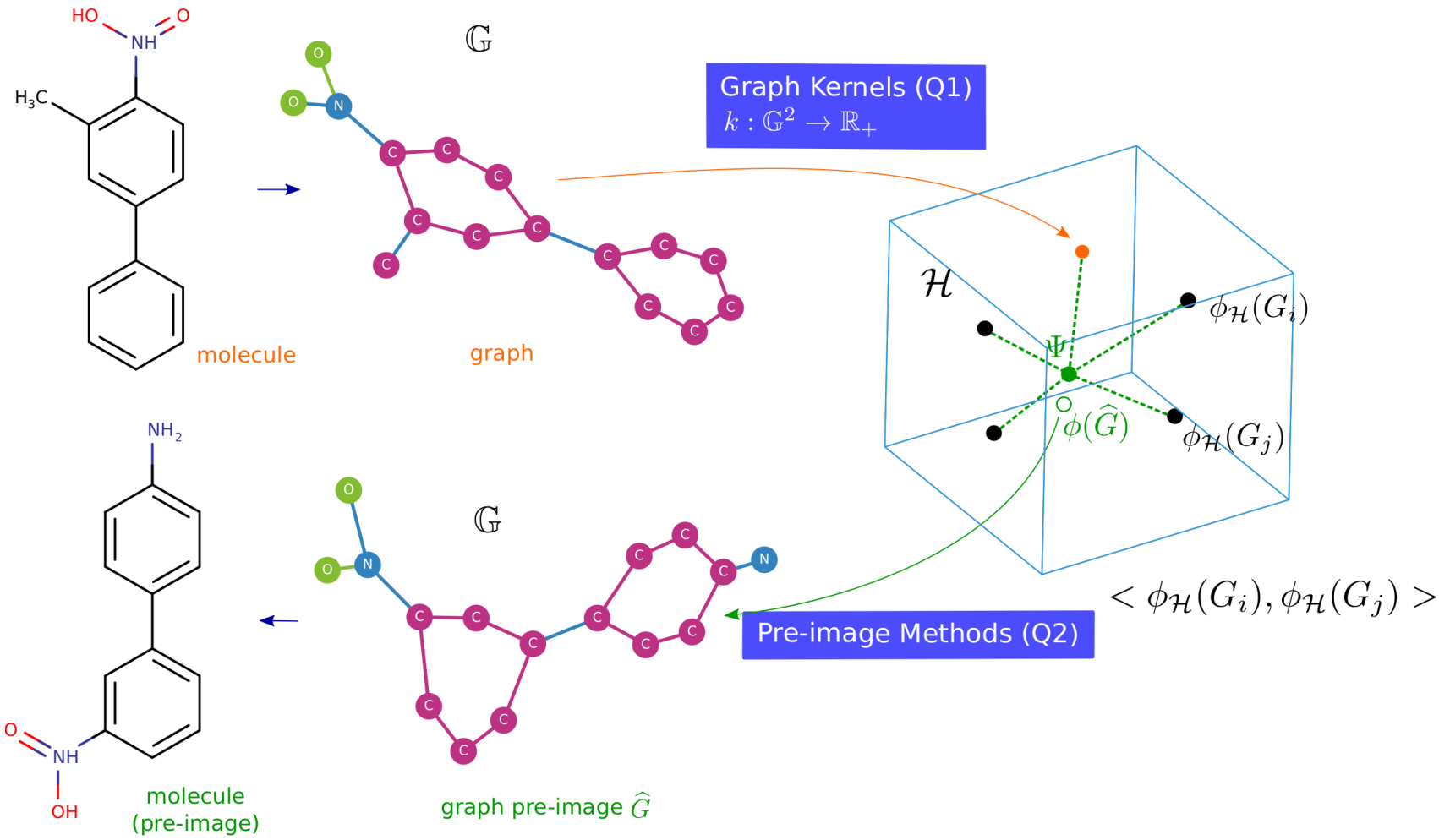
- We studied the stability of the GED heuristics;
- We proposed a method to learn optimal graph edit costs for regression:
  - We apply metric learning for space alignment;
  - An alternated optimization strategy is used to update costs.
- Experiments show promising results.

Constructing graphs using edit operations $\Longrightarrow$ used for graph pre-image methods

Overview

Backgrounds

Graph kernels and graph pre-images



Backgrounds

Current graph pre-image methods

Baseline: based on random iterations[1]:

- It explores the kernel space;
- It is restricted to inserting or removing edges;
- It can only deal with a small number of inserted/removed edges;
- There is no labeling information;
- Random construction leads to quality decrease and is time-consuming.

Contribution: our framework:

GED as a direction + an iterative graph pre-image method:

- Variant 1: with random edit costs;
- Variant 2: with expert edit costs;
- Variant 3: with optimized edit costs.

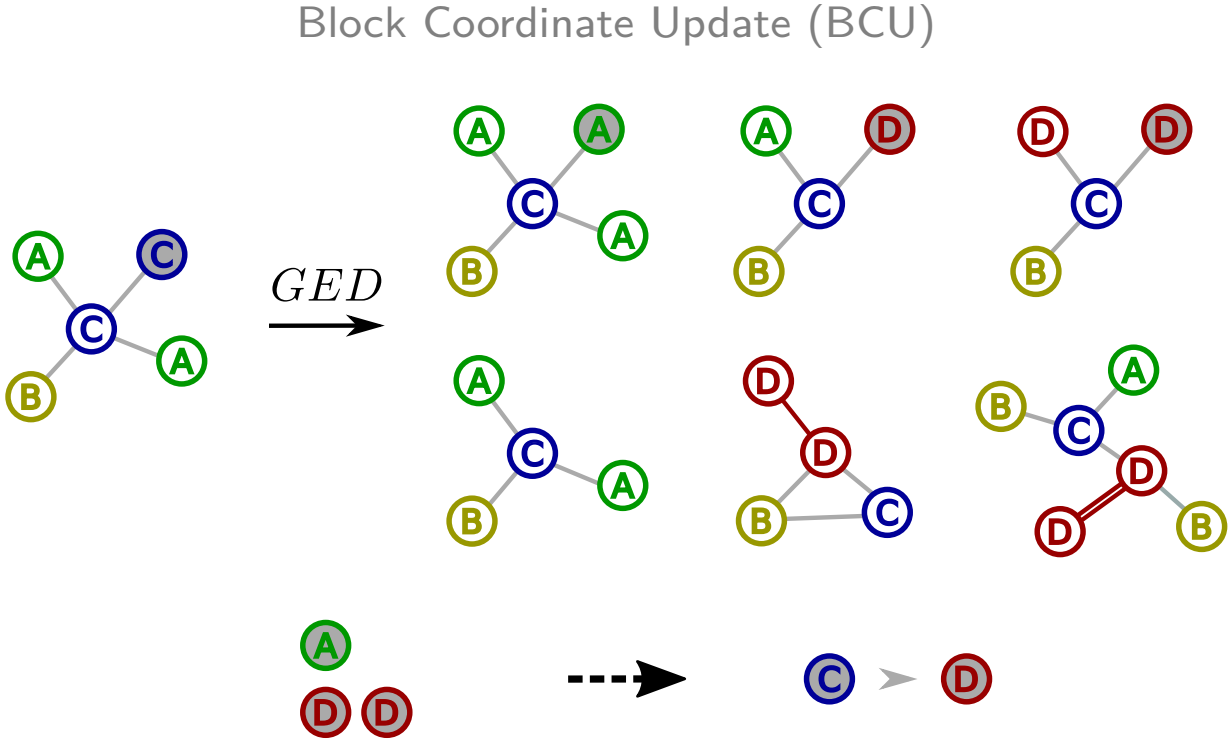
[1] Bakır, et al, Learning to find graph pre-images, Joint Pattern Recognition Symposium, 2004.

Proposed graph pre-image framework

The generalized median graph

Hypothesis

Median graph computed by BCU corresponds to the mean element in the kernel space.

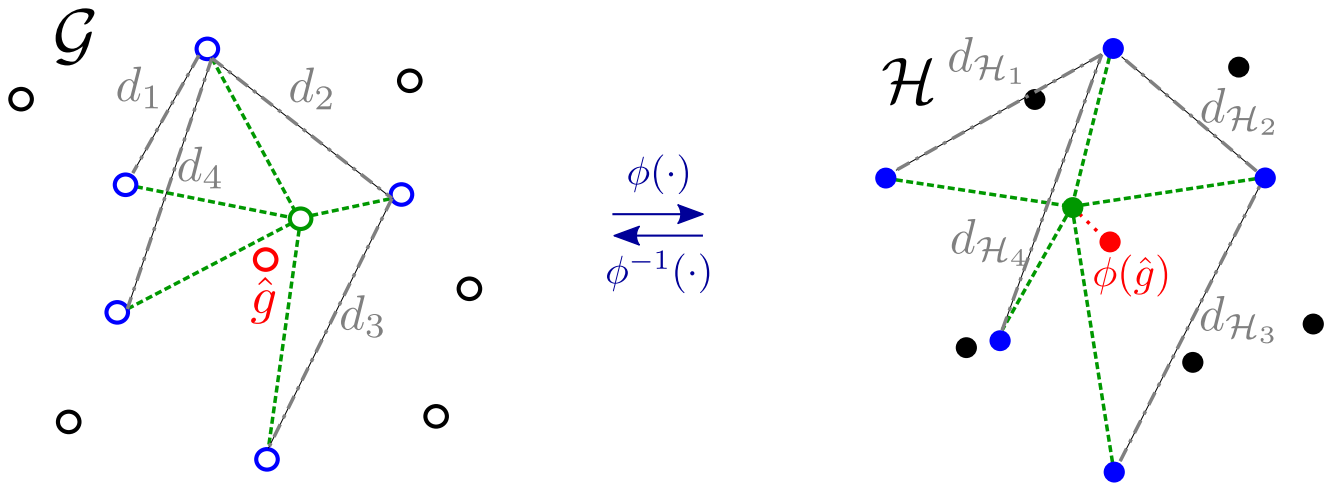


A SOTA method to construct median graph based on GED[1] is used.

[1] Boria, et al, Generalized median graph via iterative alternate minimizations, International Workshop on Graph-Based Representations in Pattern Recognition, 2019.

Proposed graph pre-image framework

A variant: align graph space and kernel space



$$d_1 = d_{\mathcal{H}_1}, \quad d_2 = d_{\mathcal{H}_2}, \quad d_3 = d_{\mathcal{H}_3}, \quad d_4 = d_{\mathcal{H}_4}, \quad \dots$$

$$\begin{cases} \texttt{ged}(G_i, G_j) = \boldsymbol{\omega}^\top \boldsymbol{c} \\ d_{\mathcal{H}}(\phi(G_i), \phi(G_j)) = \sqrt{k(G_i, G_i) + k(G_j, G_j) - 2k(G_i, G_j)} \end{cases}$$

$$\Longrightarrow \quad \boxed{\underset{\boldsymbol{c}, \boldsymbol{\omega} \geqslant \mathbf{0}}{\arg\min} \sum_{i,j=1}^{N} \left(\texttt{ged}^{i,j} - d_{\mathcal{H}}^{i,j} \right)^2}$$

Proposed graph pre-image framework

A variant: alternative iterative optimization strategy

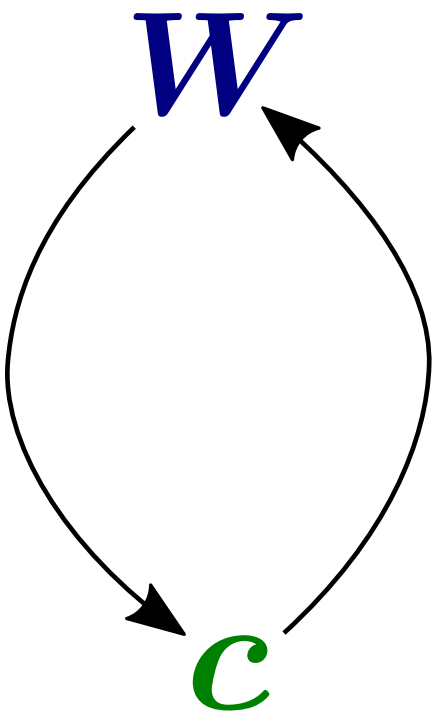
$$\arg\min_{\boldsymbol{c}} \quad \|\boldsymbol{W}^\top \boldsymbol{c} - \boldsymbol{d}_{\mathcal{H}}\|^2$$

$$\text{subject to} \quad \boldsymbol{c} \geqslant \boldsymbol{0}$$

$$c_{vr} + c_{vi} \geqslant c_{vs}$$

$$c_{er} + c_{ei} \geqslant c_{es}$$

`CVXPY, scipy`



$\forall\, G_i, G_j,$

$\text{ged}(G_i, G_j) = \boldsymbol{\omega}(i,j)^\top \boldsymbol{c}$

`bipartite, IPFP`

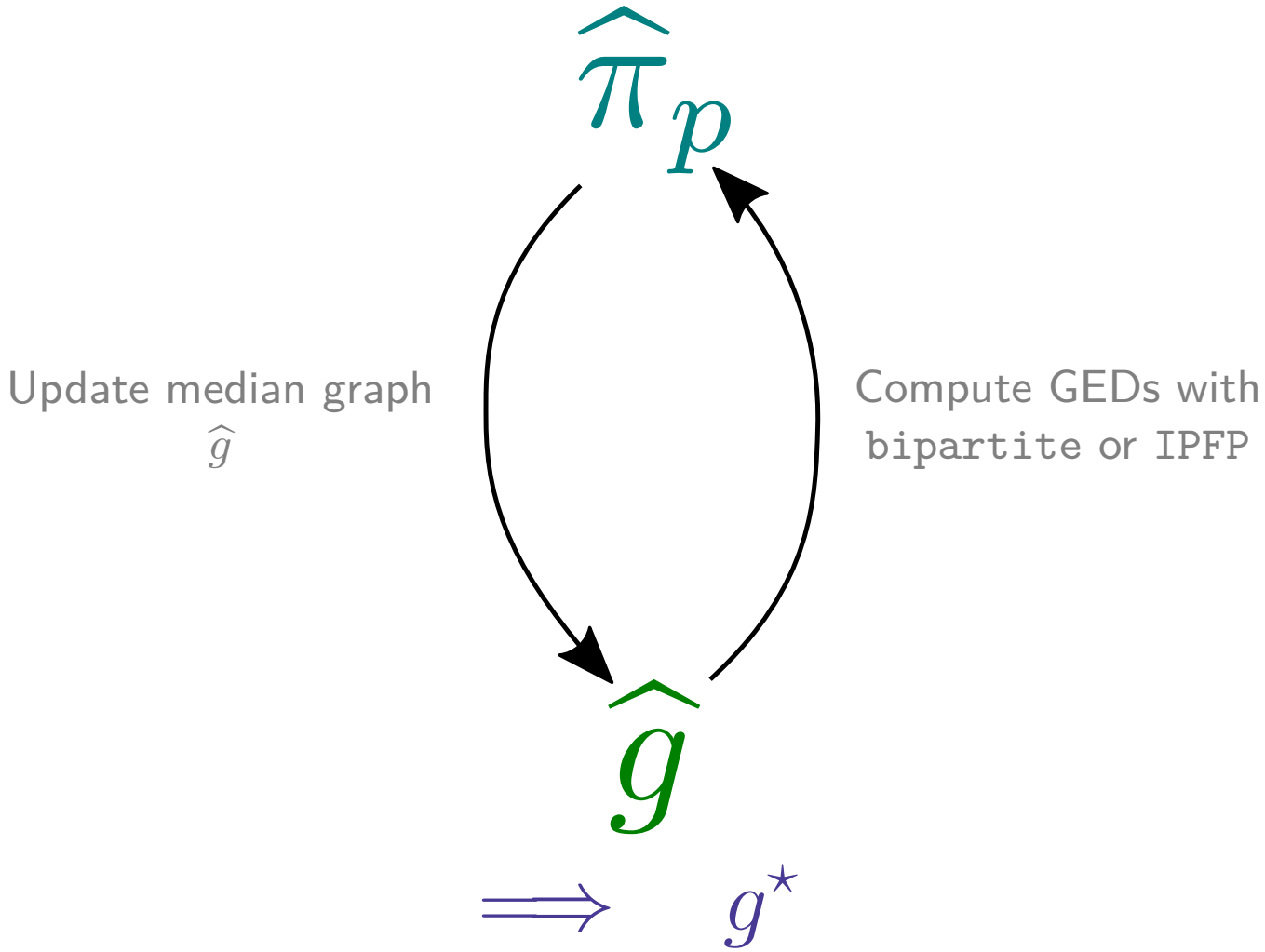
$\Longrightarrow \quad \boldsymbol{c}_{optimized}$

- $\boldsymbol{W}^\top$: the N^2-by-6 matrix with rows $\boldsymbol{\omega}(i,j)^\top$
- $\boldsymbol{d}_{\mathcal{H}}$: the vector of N^2 entries $d_{\mathcal{H}}(\phi(G_i), \phi(G_j))$, for $i, j = 1, \ldots, N$

Proposed graph pre-image framework

A variant: generate graph pre-image

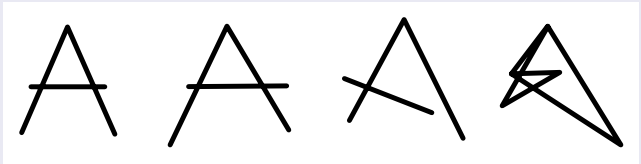
Using $\boldsymbol{c}_{optimized}$, alternately iterate the following procedures:



Experiment settings

Datasets:

- The `Letter` datasets:
  - Handwriting letters with different distortions;
  - Nodes with continuous labels.



Configurations:

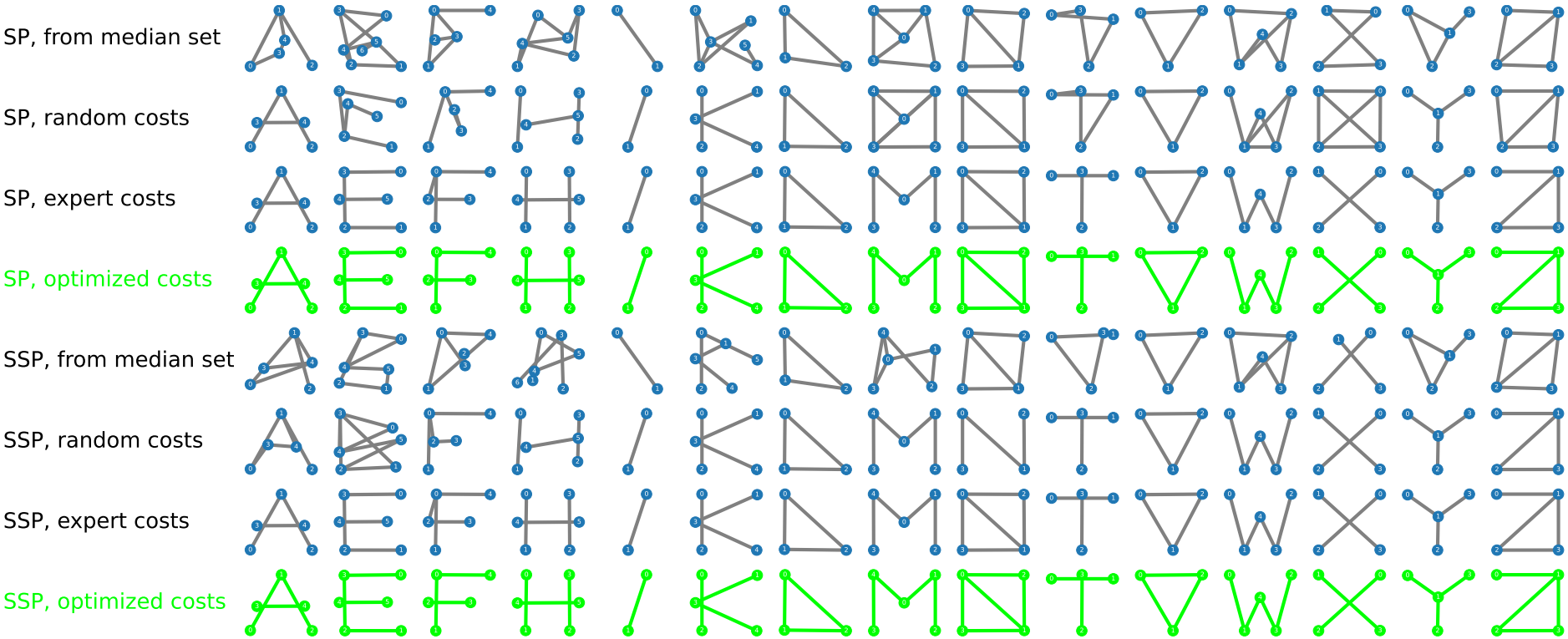
- m`bipartite` and m`IPFP` with # of batches of solutions $m = 40$;
- Size of median set: 150;
- Maximum number of iterations: 6.

Distances in kernel space

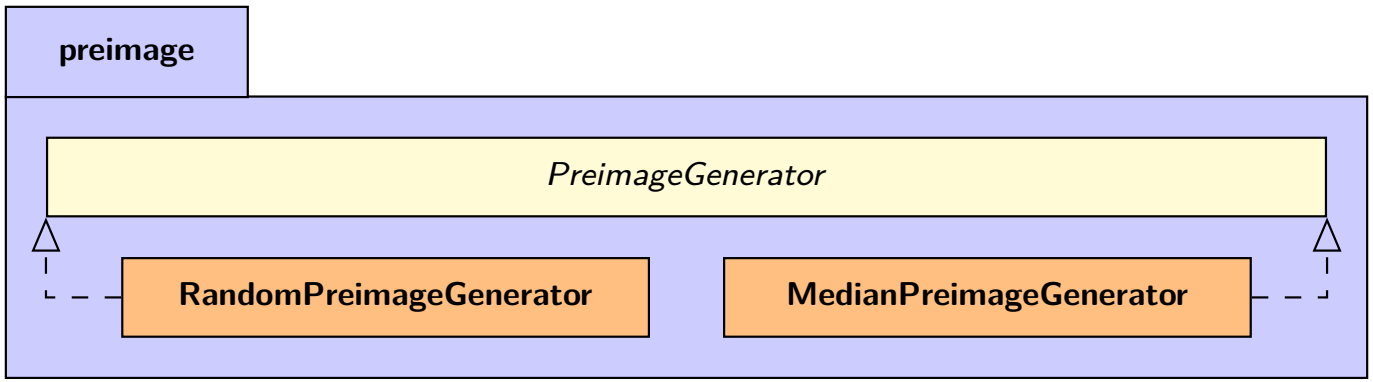
Datasets	Graph Kernels	Algorithms	$d_{\mathcal{H}}$	Running Times (in seconds)		
				Optimization	Generation	Total
Letter-high	shortest path	From median set	**0.406**	-	-	-
		Proposed: random costs	0.467	-	142.59	142.59
		Proposed: expert costs	0.451	-	30.31	30.31
		Proposed: optimized costs	0.460	5968.92	26.55	5995.47
	structural sp	From median set	0.413	-	-	-
		Proposed: random costs	0.435	-	30.22	30.22
		Proposed: expert costs	**0.391**	-	29.71	29.71
		Proposed: optimized costs	**0.394**	24.79	25.60	50.39
Letter-med	shortest path	From median set	0.425	-	-	-
		Proposed: random costs	0.303	-	25.61	25.61
		Proposed: expert costs	**0.288**	-	26.93	26.93
		Proposed: optimized costs	**0.288**	23.72	24.79	48.52
	structural sp	From median set	0.380	-	-	-
		Proposed: random costs	0.286	-	24.77	24.77
		Proposed: expert costs	**0.248**	-	27.51	27.51
		Proposed: optimized costs	**0.248**	27.06	29.24	56.30

Pre-images constructed by different algorithms

Pre-images generated as median graphs for each letter of *Letter-high* dataset using random costs, expert costs and optimized costs:



Implementations of graph pre-image methods



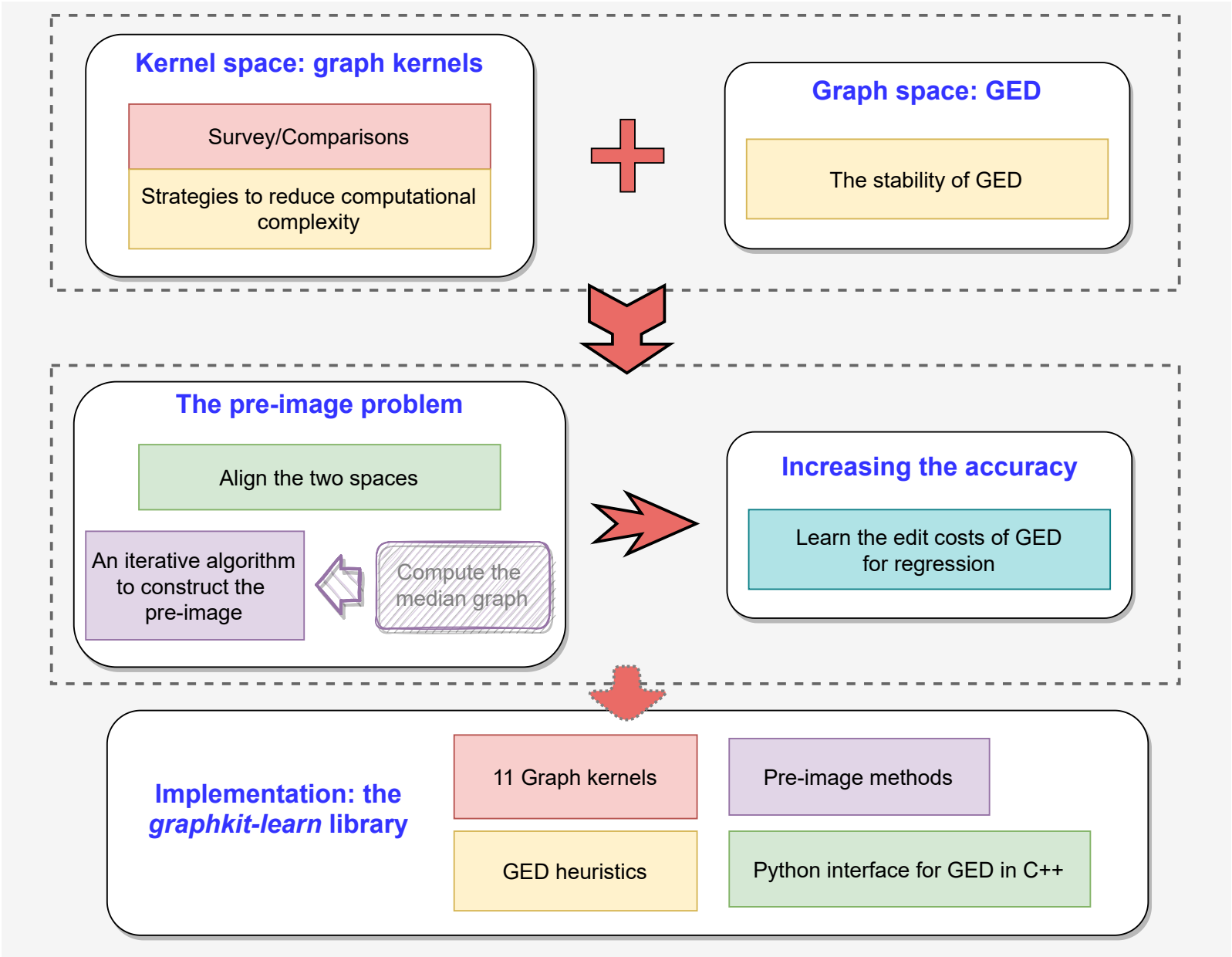
Conclusion

Conclusion:

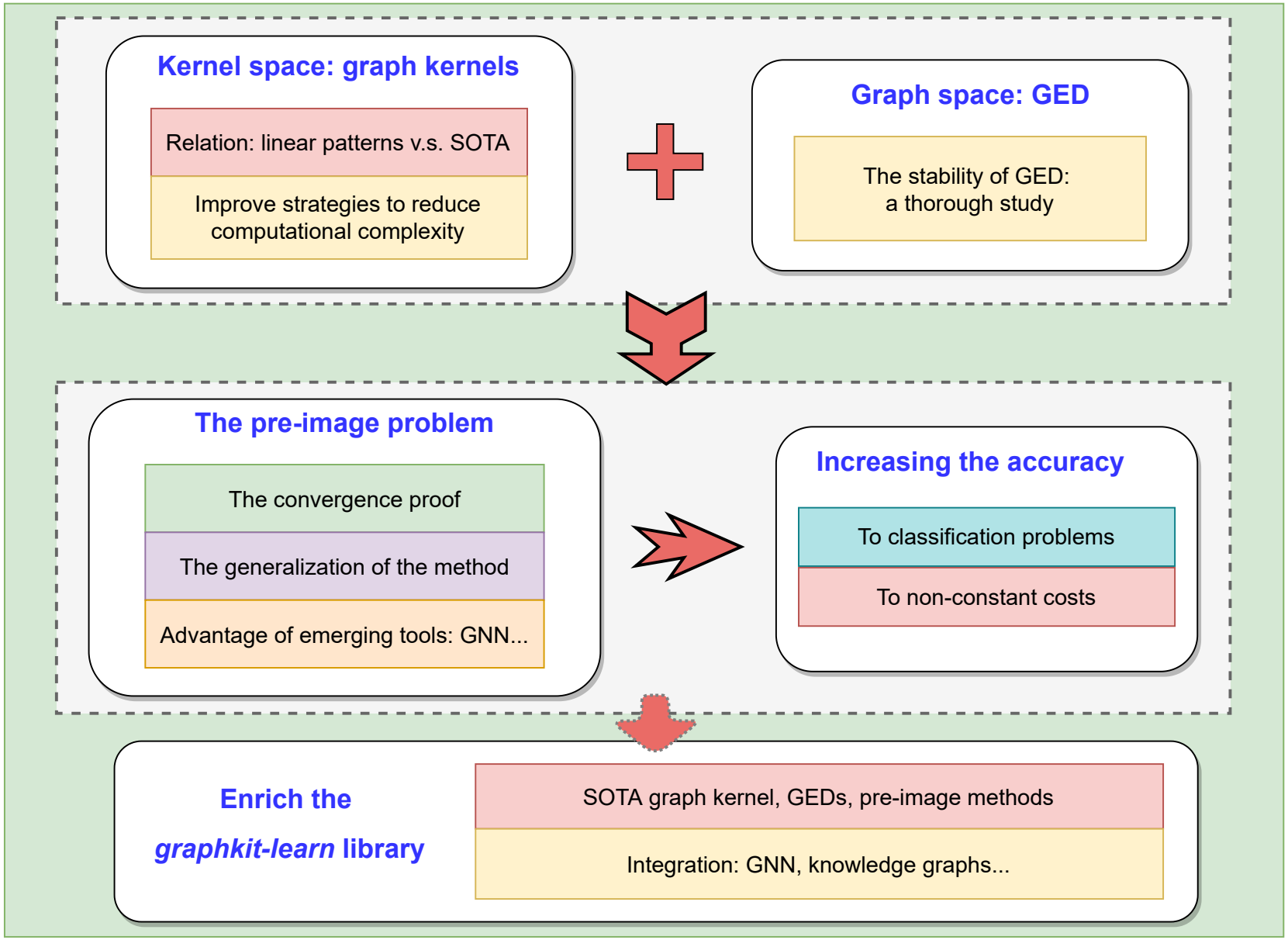
- We proposed a novel framework to estimate graph pre-images:
  - Three variants are proposed;
  - Kernel and graph spaces are aligned using metric learning;
  - Edit costs are optimized during this procedure;
  - The graph pre-image is constructed with the optimized costs using an alternative iteration.
- The proposed framework generates better pre-images than other methods.

Overview

Conclusions



Future work


Kernel space: graph kernels
Relation: linear patterns v.s. SOTA
Improve strategies to reduce computational complexity
Graph space: GED
The stability of GED: a thorough study
The pre-image problem
The convergence proof
The generalization of the method
Advantage of emerging tools: GNN...
Increasing the accuracy
To classification problems
To non-constant costs
Enrich the graphkit-learn library
SOTA graph kernel, GEDs, pre-image methods
Integration: GNN, knowledge graphs...

Publications

Peer-reviewed journal papers

- **Linlin Jia**, Benoit Gaüzère, and Paul Honeine. graphkit-learn: A python library for graph kernels based on linear patterns. Pattern Recognition Letters, 2021.
- **Linlin Jia**, Benoit Gaüzère, and Paul Honeine. Graph Kernels Based on Linear Patterns: Theoretical and Experimental Comparisons. (submitted to *Expert Systems with Applications* in March 2020)

Peer-reviewed international conference papers

- **Linlin Jia**, Benoit Gaüzère, and Paul Honeine. A Graph Pre-image Method Based on Graph Edit Distances. Proceedings of IAPR Joint International Workshops on Statistical Techniques in Pattern Recognition (SPR 2020) and Structural and Syntactic Pattern Recognition (SSPR 2020), 2021.
- **Linlin Jia**, Benoit Gaüzère, Florian Yger and Paul Honeine. A Metric Learning Approach to Graph Edit Costs for Regression. Proceedings of IAPR Joint International Workshops on Statistical Techniques in Pattern Recognition (SPR 2020) and Structural and Syntactic Pattern Recognition (SSPR 2020), 2021.

Libraries

- `graphkit-learn`: A Python package on graph kernels, graph edit distances and graph pre-image problem.

Questions

Thank you.

Any questions?

Graph kernels comparison

Table: Characteristics of graph kernels based on linear patterns and two on non-linear patterns.

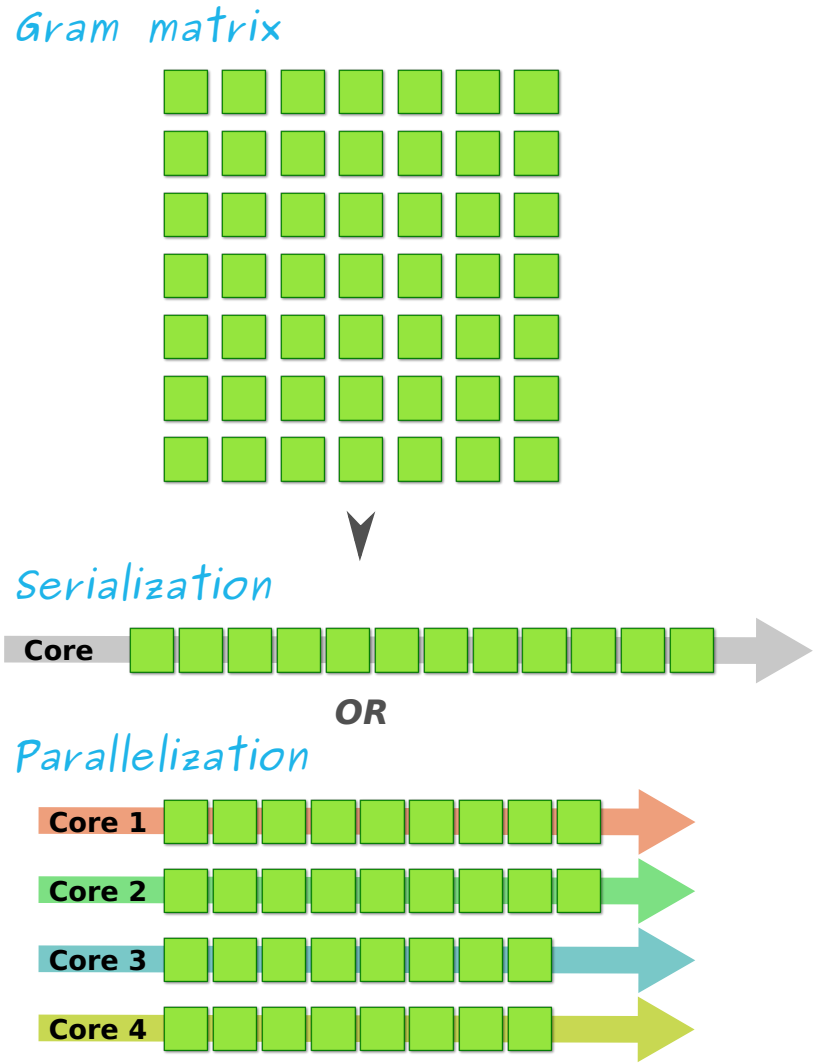
Kernels	Substructures			Labeling				Directed	Edge Weighted	Time Complexity (Gram Matrix)	Explicit Representation	Weighting
	linear	non-linear	cyclic	symbolic		non-symbolic						
				vertices	edges	vertices	edges					
Common walk	✓	✗	✗	✓	✓	✗	✗	✓	✗	$\mathcal{O}(N^2n^6)$	✗	a priori
Marginalized	✓	✗	✗	✓	✓	✗	✗	✓	✗	$\mathcal{O}(N^2rn^4)$	✗	✗
Sylvester equation	✓	✗	✗	✗	✗	✗	✗	✓	✓	$\mathcal{O}(N^2n^3)$	✗	a priori
Conjugate gradient	✓	✗	✗	✓	✓	✓	✓	✓	✓	$\mathcal{O}(N^2rn^4)$	✗	a priori
Fixed-point iterations	✓	✗	✗	✓	✓	✓	✓	✓	✓	$\mathcal{O}(N^2rn^4)$	✗	a priori
Spectral decomposition	✓	✗	✗	✗	✗	✗	✗	✓	✓	$\mathcal{O}(N^2n^2 + Nn^3)$	✗	a priori
Shortest path	✓	✗	✗	✓	✗	✓	✗	✓	✓	$\mathcal{O}(N^2n^4)$	✗	✗
Structural shortest path	✓	✗	✗	✓	✓	✓	✓	✓	✗	$\mathcal{O}(hN^2n^4 + N^2nm))$	✗	✗
Path kernel up to length h	✓	✗	✗	✓	✓	✗	✗	✓	✗	$\mathcal{O}(N^2h^2n^2d^{2h})$	✓	✓
Treelet	✓	✓	✗	✓	✓	✗	✗	✓	✗	$\mathcal{O}(N^2nd^5)$	✓	✓
WL subtree	✓	✓	✗	✓	✗	✗	✗	✓	✗	$\mathcal{O}(Nhm + N^2hn)$	✓	✗

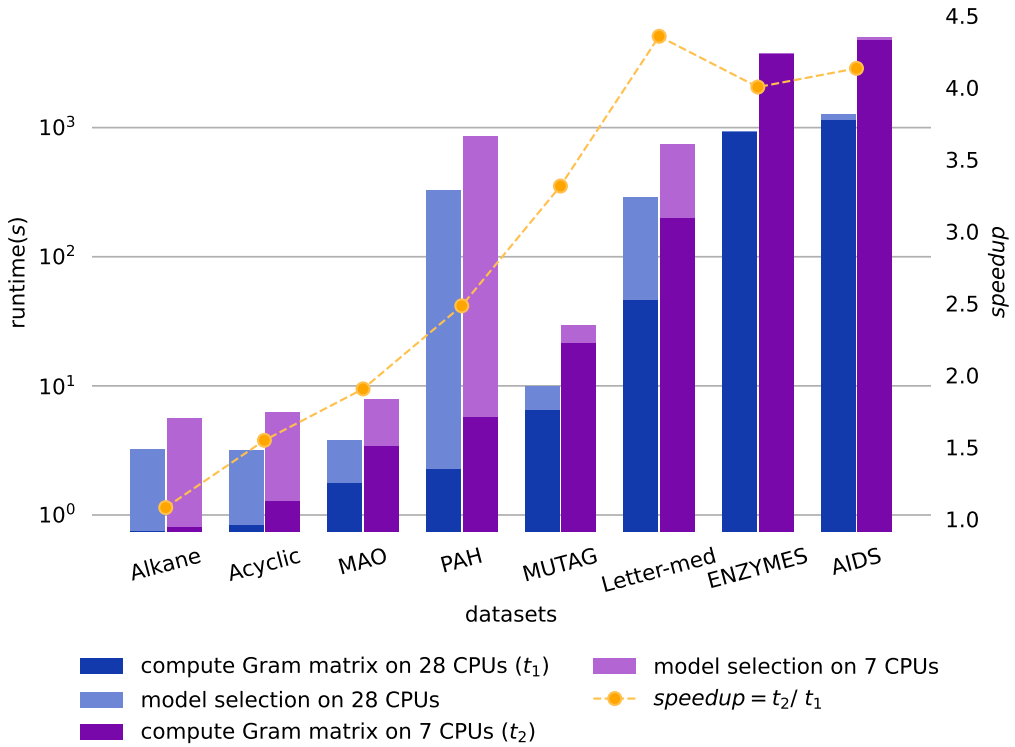
— The "Time complexity" column is a rough estimation for computing the Gram matrix.

— The "Explicit representation" column indicates whether the embedding of graphs in the representation space can be encoded by a vector explicitly; in other words, whether the patterns of graph kernels can be explicitly presented.

— The "Weighting" column indicates whether the substructures can be weighted in order to obtain a similarity measure adapted to the problem at hand, where "a priori" indicates that the weights are set while constructing kernels.

Effect of parallelization: number of computing cores

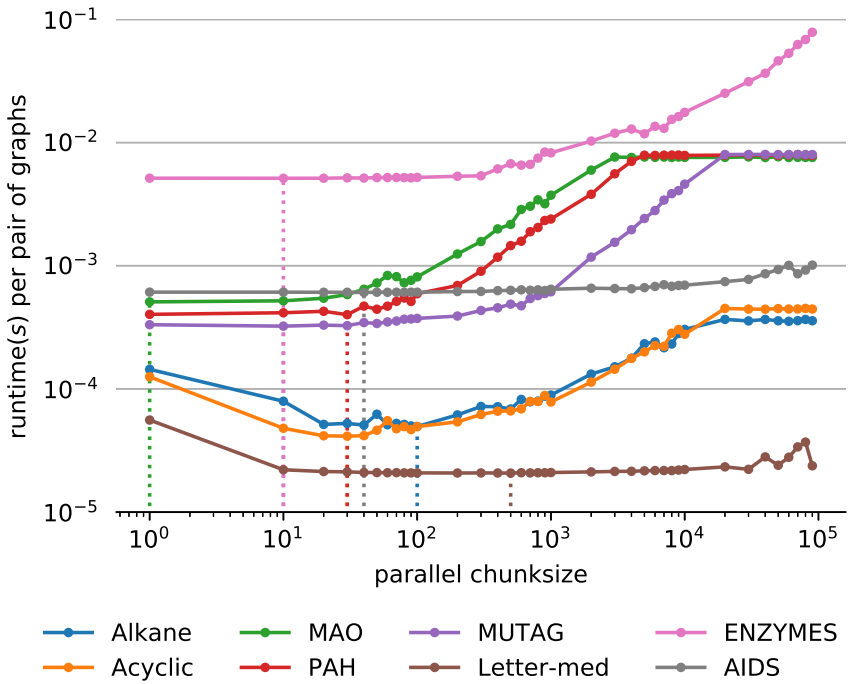




The *speedup* for large-scale datasets are around 4, which turns out to be the inverse ratio of the number of CPU cores (28/7).

Effect of parallelization: `chunksize`

Figure: Runtimes to compute the Gram matrices of the shortest path kernel with different `chunksize` values.



- Wise `chunksize` choices can not be too small or too large;
- It could be more than 20 times faster than the worst choices.

Figure: The ratio between runtimes of the worst and the best `chunksize` settings.

	common walk	marginalized	Sylvester equation	conjugate gradient	fixed-point iterations	spectral decomposition	shortest path	structural SP	path up to length h	treelet	WL subtree
Alkane	1.09	24.94	8.1	23.14	24.22	1.71	20.39	23.29	2.75	2.51	1.02
Acyclic	1.24	25.19	8.9	19.11	18.27	1.51	20.36	22.56	4.89	2.72	1.02
MAO	1.68	24.59	4.32	1.14	1.06	7.25	20.96	24.91	6.46	2.0	1.03
PAH	1.32	26.05	5.17	1.15	1.16	10.19	24.98	26.98	2.06	3.26	1.04
Mutag	1.16	27.23	12.09	1.39	1.57	5.69	25.43	27.47	13.5	6.17	1.01
Letter-med	1.43	1.54	14.83	1.46	1.65	1.61	2.63	2.37	31.54	30.35	1.01
Enzymes	1.43	14.66	1.85	2.03	1.73	inf	14.52	inf	16.23	17.22	1.01
AIDS	1.08	1.59	2.0	1.07	1.08	inf	1.8	2.15	1.8	21.04	1.01
NCI1	inf	1.17	1.94	inf	inf	inf	1.27	inf	4.14	23.94	1.07
NCI109	inf	1.27	1.93	inf	inf	inf	1.3	inf	6.04	32.0	1.07
D&D	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf	1.01

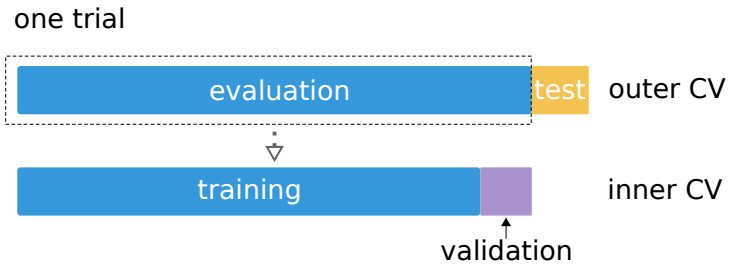
- The ratio is up to 30 times faster.

Experiment settings: environment settings

Environment settings

Environments	Settings
CPU	Intel(R) Xeon(R) E5-2680 v4 @ 2.40GHz
# CPU cores	28
Memory (in total)	252 GB
Operating system	CentOS Linux release 7.3.1611, 64 bit
Python version	3.6.9

Two-layer nested cross validations

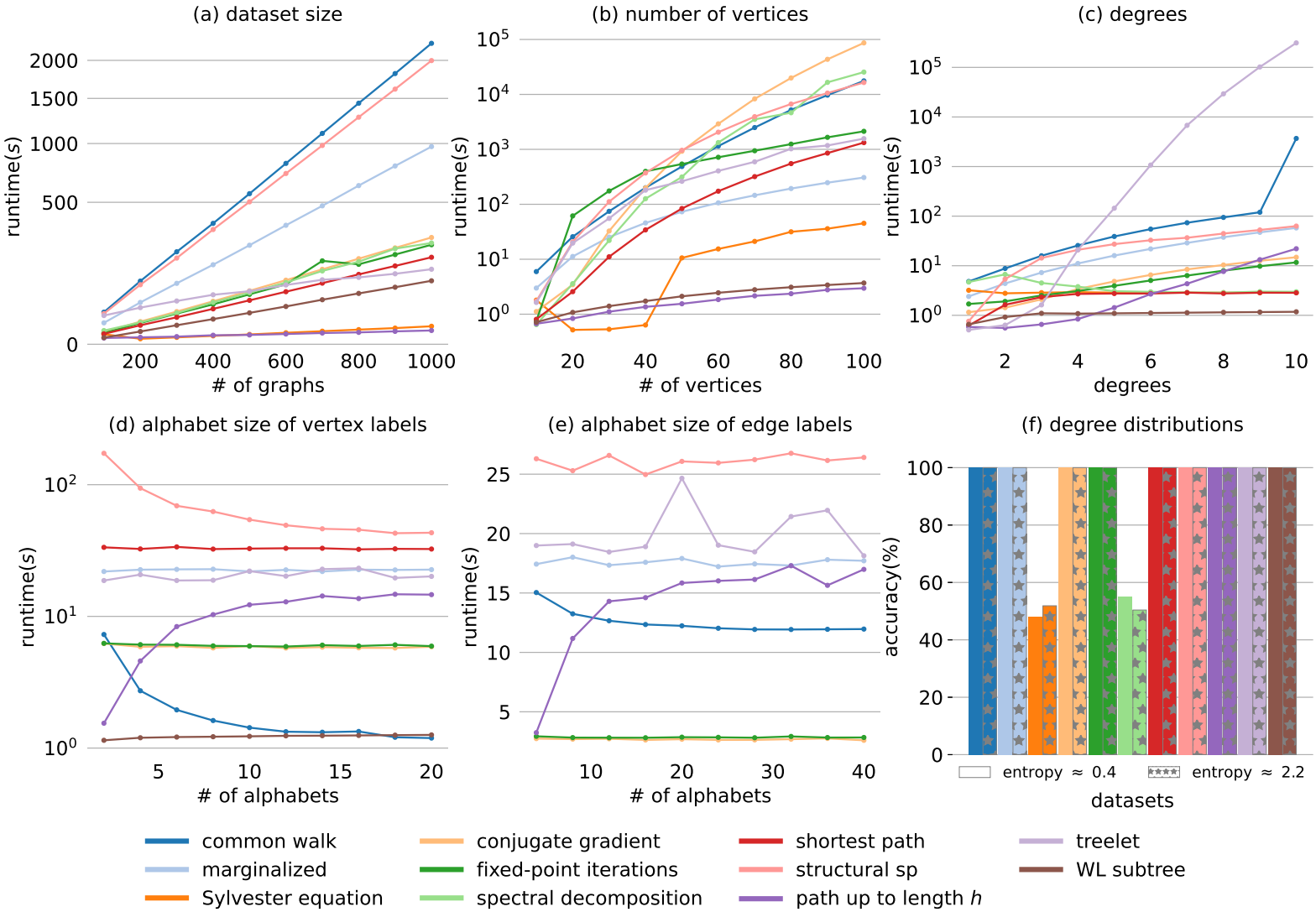


- outer CV: 90% evaluation + 10% test;
- inter CV: 90% training + 10% validation;
- 30 trials.

The ranges of hyper-parameters for each kernel

Kernels	Hyper-parameter ranges
Common walk	method: geo, γ: [0.01, 0.02, . . . , 0.15] method: exp, γ: [0,1, 2, . . . , 15]
Marginalized	iter: $[1, 5, \ldots, 20]$, p_q: [0.1, 0.2, . . . , 1.0]
Sylvester equation	λ: [1-10, 1e-9, . . . , 1e-1]
Conjugate gradient	λ: [1-10, 1e-9, . . . , 1e-1]
Fixed-point iterations	λ: [1-10, 1e-9, . . . , 1e-1]
Spectral decomposition	λ: [1-10, 1e-9, . . . , 1e-1]
Path kernel up to length h	h: [1, 2, . . . , 10], $kfunc$: [MinMax, Tanimoto]
Treelet	kernel: gaussian, γ: [1e-10, 1e-9, . . . , 1] kernel: polynomial, d: [1, 2, . . . , 10], c: [1, 10, . . . , 1e10], γ: [1e-10, 1e-9, . . . , 1]
WL subtree	height: [0, 1, . . . , 10]

Performance on synthesized graphs



Performance on real-world datasets

Datasets

Datasets	Substructures			Numbers of Labels				Directed	N	$\bar{n}$	$\bar{m}$	d	Class Numbers	Tasks
	linear	non-linear	cyclic	symbolic		non-symbolic								
				vertices	edges	vertices	edges							
Acyclic	✓	✓	✗	3	✗	✗	✗	✗	183	8.15	7.15	1.47	-	R
Alkane	✓	✓	✗	2	✗	✗	✗	✗	150	8.87	7.87	1.75	-	R
MAO	✓	✓	✓	3	4	✗	✗	✗	68	18.38	19.63	2.13	2	C
PAH	✓	✓	✓	✗	✗	✗	✗	✗	94	20.70	24.43	2.36	2	C
Mutag	✓	✓	✓	7	11	✗	✗	✗	188	17.93	19.79	2.19	2	C
Letter-med	✓	✓	✓	✗	✗	2	✗	✗	2250	4.67	3.21	1.35	15	C
Enzymes	✓	✓	✓	3	✗	18	✗	✗	600	32.63	62.14	3.86	6	C
AIDS	✓	✓	✓	38	3	4	✗	✗	2000	15.69	16.20	2.01	2	C
NCI1	✓	✓	✓	37	✗	✗	✗	✗	4110	29.87	32.30	2.16	2	C
NCI109	✓	✓	✓	38	✗	✗	✗	✗	4127	29.68	32.13	2.15	2	C
D&D	✓	✓	✓	82	✗	✗	✗	✗	1178	284.32	715.66	4.98	2	C

- "Substructures" are the sub-patterns that graphs contain;
- "Numbers of labels" include numbers of symbolic and non-symbolic vertex and edge labels, with ✗ for no label;
- "Directed" exhibits whether directed graphs are included;
- N is the number of graphs; $\bar{n}$ is the average number of graph vertices;
- $\bar{m}$ is the average number of edges;
- d is the average vertex degree;
- "Tasks" are either regression ("R") or classification ("C").

Performance on synthesized graphs

Observations

- **# of graphs**: runtimes are quadratic in the number of graphs;
- **# of vertices**: number of vertices $\nearrow \Longrightarrow$ runtimes $\nearrow$;
- **Average # vertex degree**: most kernels have good scalability to the degrees.
- **Alphabet size of symbolic vertex labels**: size $\nearrow \Longrightarrow$ runtimes of path kernel up to length h $\nearrow$, common walk kernel and structural shortest path kernel $\searrow$, else the same.
- **Alphabet size of symbolic edge labels**: size $\nearrow \Longrightarrow$ runtimes of path kernel up to length h $\nearrow$, else the same.
- **Distribution of degrees**: has little effect on runtimes.

Performance on real-world datasets

Accuracies achieved by graph kernels

	common walk	marginalized	Sylvester equation	conjugate gradient	fixed-point iterations	spectral decomposition	shortest path	structural SP	path up to length h	treelet	WL subtree
Alkane	15.52	43.75	8.97	11.13	12.78	12.95	7.81	8.65	9.0	2.53	26.42
Acyclic	12.93	18.77	32.5	13.15	14.2	33.05	9.03	13.1	6.66	5.99	19.8
MAO	93.0	85.62	84.52	88.57	73.71	77.67	87.81	91.62	85.43	91.19	93.05
PAH	71.8	57.67	71.5	73.93	58.33	70.73	69.4	74.5	75.27	66.3	75.93
Mutag	85.96	76.11	82.77	86.18	86.58	84.05	81.84	86.26	88.47	90.79	87.18
Letter-med	36.16	5.2	37.27	93.12	91.3	36.38	93.72	94.88	43.83	inf	36.13
Enzymes	42.81	45.92	23.24	60.89	63.11	23.68	70.09	inf	57.49	52.23	50.76
AIDS	94.71	inf	92.42	98.93	98.57	87.21	99.26	98.84	99.65	99.54	98.63
NCI1	inf	inf	59.76	71.34	inf	inf	inf	79.88	84.84	64.84	84.63
NCI109	inf	inf	60.62	67.6	67.25	inf	inf	79.04	83.94	63.46	85.47
D&D	inf	inf	inf	inf	inf	inf	inf	inf	81.4	inf	77.3

In terms of regression error (the upper table) and classification rate (the lower table) estimated on the test set.

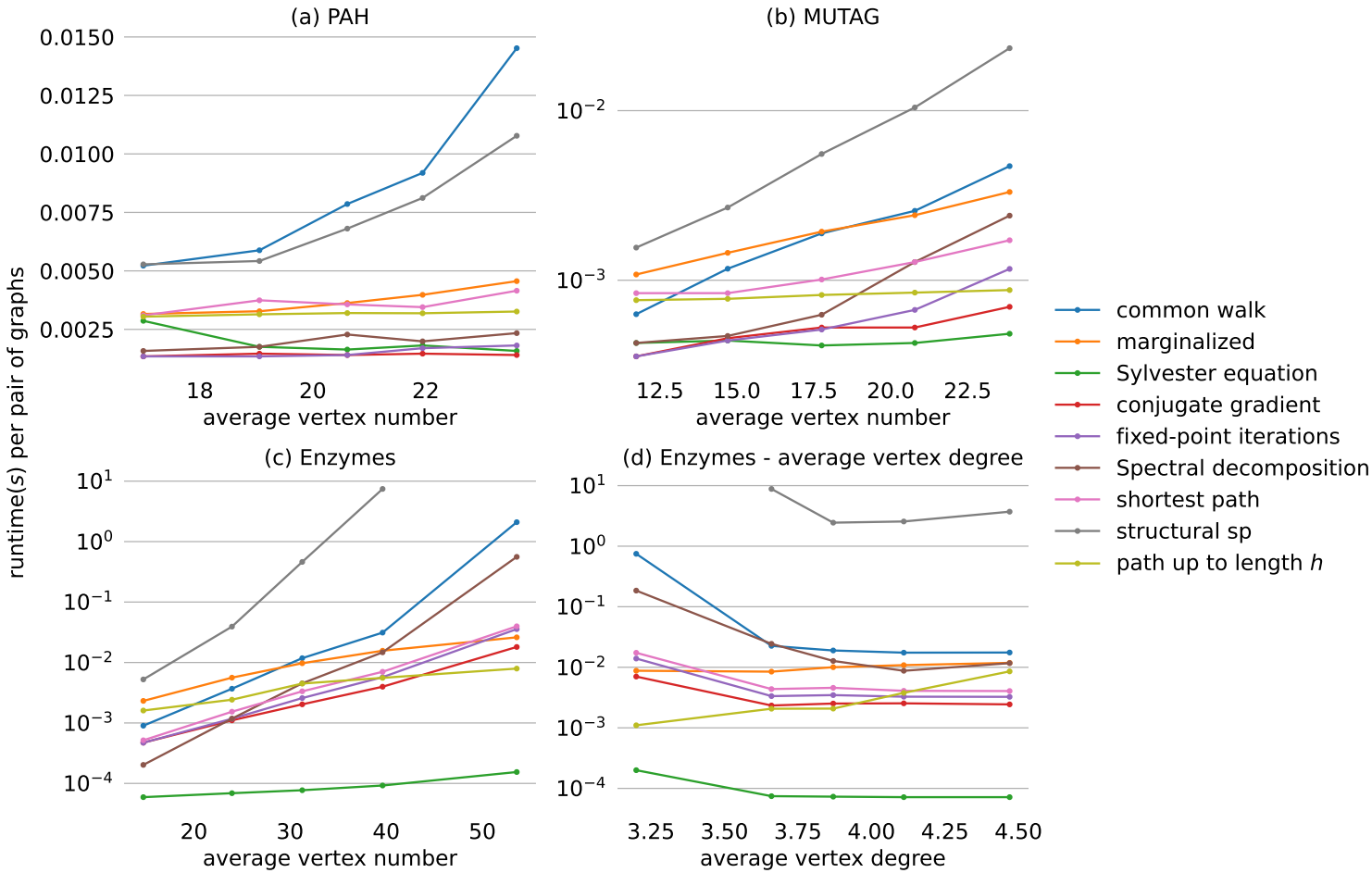
Time used to compute Gram matrices of graph kernels

	common walk	marginalized	Sylvester equation	conjugate gradient	fixed-point iterations	spectral decomposition	shortest path	structural SP	path up to length h	treelet	WL subtree
Alkane	0.48	0.51	-0.42	-0.18	-0.22	-0.19	-0.12	0.02	-0.29	-0.3	0.16
Acyclic	0.36	0.62	-0.18	-0.04	-0.11	-0.1	-0.08	0.24	-0.3	-0.31	0.34
MAO	0.81	0.69	-0.47	-0.11	-0.03	0.14	0.25	0.88	-0.14	-0.28	-0.25
PAH	1.56	1.05	-0.42	0.14	0.25	0.37	0.36	1.32	-0.28	-0.24	-0.03
Mutag	1.28	1.36	-0.3	0.44	0.53	0.77	0.69	1.84	-0.07	-0.24	0.19
Letter-med	2.01	2.08	1.13	1.97	1.85	1.78	1.57	1.62	1.08	inf	2.02
Enzymes	3.9	3.18	0.72	2.62	2.79	3.4	2.85	inf	2.16	2.08	1.41
AIDS	2.83	inf	1.37	2.91	3.03	3.74	2.95	3.9	1.59	0.87	2.22
NCI1	inf	inf	2.3	3.96	inf	inf	inf	5.12	2.04	1.48	3.02
NCI109	inf	inf	2.3	4.37	4.38	inf	inf	5.13	2.05	1.48	1.48
D&D	inf	inf	inf	inf	inf	inf	inf	inf	2.67	inf	2.95

In $\log_{10}$ of seconds.

Performance on real-world datasets

Comparison on graphs with different average vertex numbers/degrees



Bigger # of vertices cause the increase of the computational time (Worst: common walk kernel, the structural shortest path kernel; Best: path kernel up to length h).