Universität Rostock Traditio et Innovatio

WIRTSCHAFTSINFORMATIK
UNIVERSITÄT ROSTOCK

PaSiGraM – Parallel Frequent Subgraph Mining in a Single Large Graph

Ole Fenske
ole.fenske@uni-rostock.de

Agenda

1. Grundlagen des Frequent-Subgraph-Mining
   a. Vorgehensweise
   b. Kandidatengenerierung
   c. Isomorphie und Teilgraph-Isomorphie
   d. Signifikanzberechnung
2. PaSiGraM
   a. Allgemeiner Ansatz
   b. Datenstruktur
   c. Kandidatengenerierung
   d. Signifikanzberechnung
   e. Weitere Optimierungen
   f. Parallelisierung
3. Zusammenfassung
4. Ausblick

1. Frequent-Subgraph-Mining

a) Vorgehensweise:

1. Erzeugen von Kandidaten (Mustern)
2. Überprüfen, ob diese Kandidaten in dem Datensatz auftreten
3. Berechnung der Signifikanz (Häufigkeit) der Kandidaten in dem Datensatz
4. Überprüfen, ob die Signifikanz eines Kandidaten über der Mindesthäufigkeit liegt

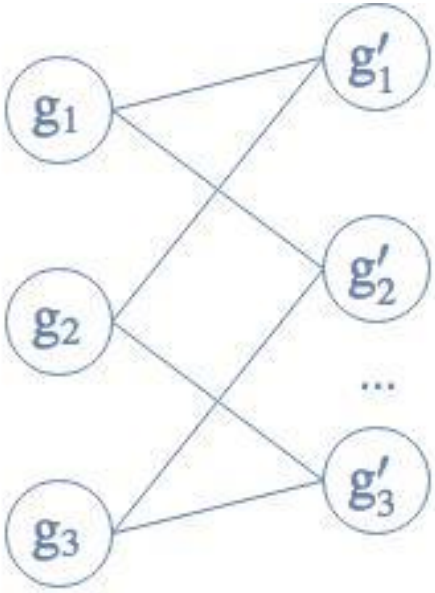
Fragen/Probleme:

- Nach welcher Strategie werden die Kandidaten generiert?
- Wie wird ermittelt ob ein Kandidat in einem Graphen auftritt (Teilgraph-Isomorphie)?
- Welches Maß für die Signifikanz eines Kandidaten kann benutzt werden?

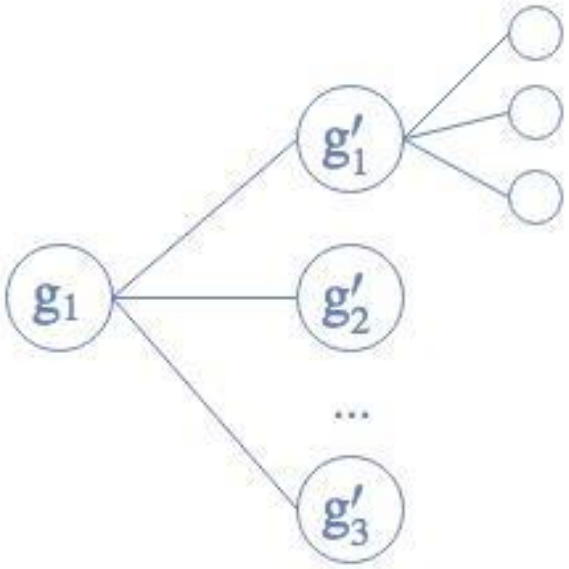
1. Frequent-Subgraph-Mining

b) Kandidatengenerierung

Definition 3.4 (Anti-Monotonie). *Ein Teilgraph G_s ist nur dann häufig, wenn all seine Teilgraphen häufig sind.*



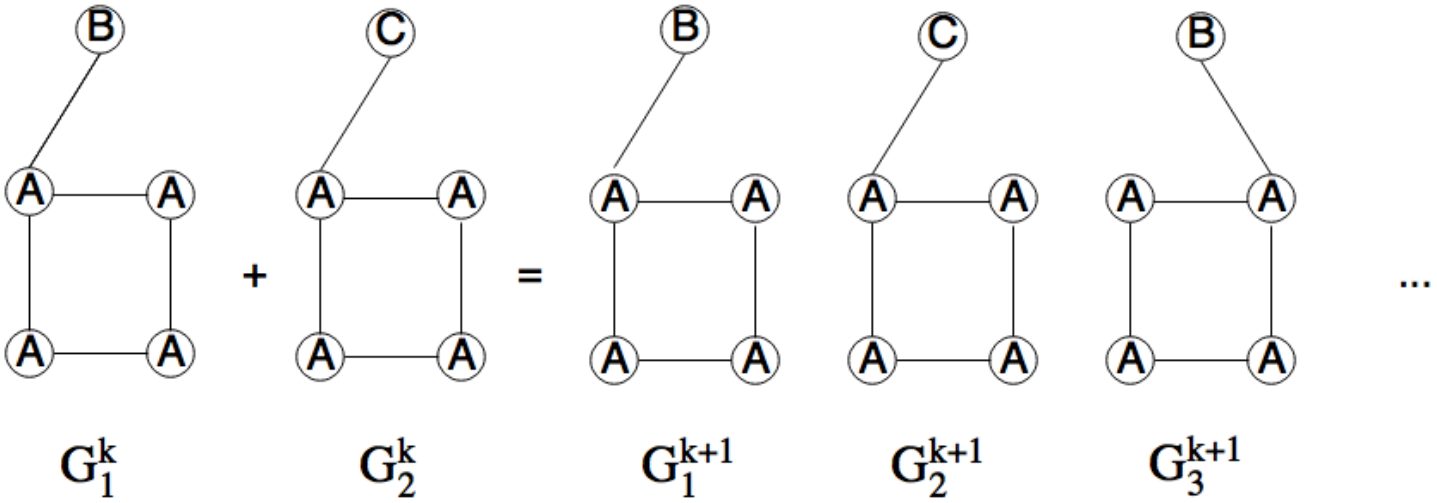
Join-basierter Ansatz



Pattern-Growth- basierter Ansatz

1. Frequent-Subgraph-Mining

b) Kandidatengenerierung



Probleme:

- Ähnlichkeit zwischen zwei Kandidaten ermitteln (Isomorphie-Problem)
- Join von zwei Kandiaten ist eine sehr teure Operation und kann somit einen Overhead verursachen

1. Frequent-Subgraph-Mining

b) Kandidatengenerierung

Pattern-Growth-basierter Ansatz:

- Zu jedem Kandidaten werden eine Kante (und ein Knoten) hinzugefügt von denen man weiß, dass diese häufig in dem zu untersuchenden Graphen auftreten

Problem:

- Kandidaten könnten mehrfach generiert werden

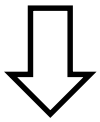
Lösung:

- Nutzen der „*right-most-extension*“ bei der Generierung von neuen Kandidaten
- *right-most-extension* = ganz rechter Pfad in einem Graphen

1. Frequent-Subgraph-Mining

c) Isomorphie und Teilgraph-Isomorphie

Definition 3.7 (Isomorphie). *Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ sind isomorph, wenn sie die gleiche Topologie besitzen. Das heißt, es gibt eine Zuordnung von V_1 zu V_2, sodass jede Kante in E_1 einer Kante in E_2 zugeordnet werden kann. Im Fall von gelabelten Graphen muss die Zuordnung ebenfalls die Labels von Knoten und Kanten beachten.*



Definition 3.8 (Teilgraph-Isomorphie). *Für zwei gelabelte Graphen $G = (V, E, l)$ und $G_s = (V_s, E_s, l_s)$ ist eine Teilgraph-Isomorphie eine injektive Funktion $f : V(G_s) \to V(G)$ unter der Bedingung von (s.t.):*

1. $\forall v \in V(G_s) : l_s(v) = l(f(v))$
2. $\forall (u, v) \in E(G_s) : (f(u), f(v)) \in E(G) \wedge l_s(u, v) = l(f(u), f(v))$

wobei l und l_s die Labelingfunktion von G und G_s ist. f wird dann eine Einbettung von G_s in G genannt.

1. Frequent-Subgraph-Mining

d) Signifikanzberechnung

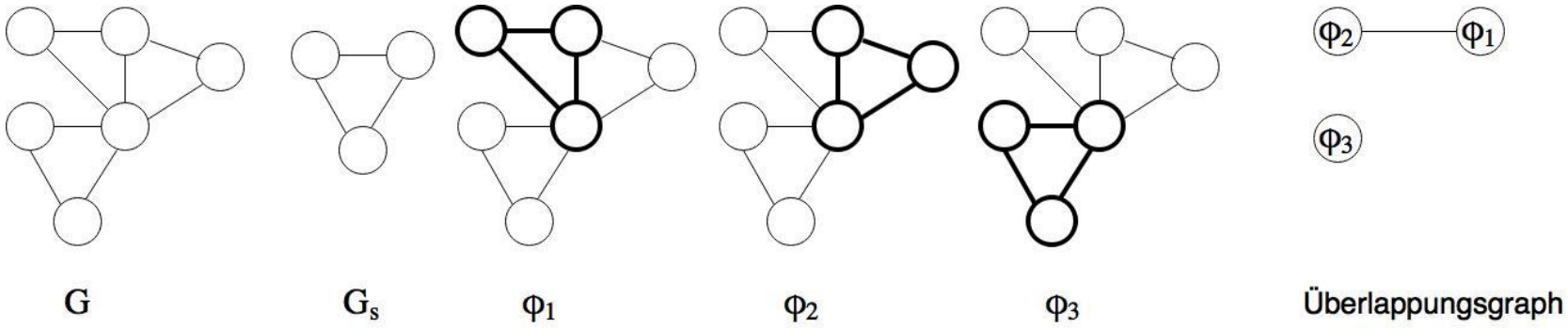
Drei Maße für die Signifikanz:

- Maximal unabhängige Menge (MIS)
- Schädliche Überlappung (HO)
- Minimum-Image-Based-Support (MNI)

1. Frequent-Subgraph-Mining

d) Signifikanzberechnung

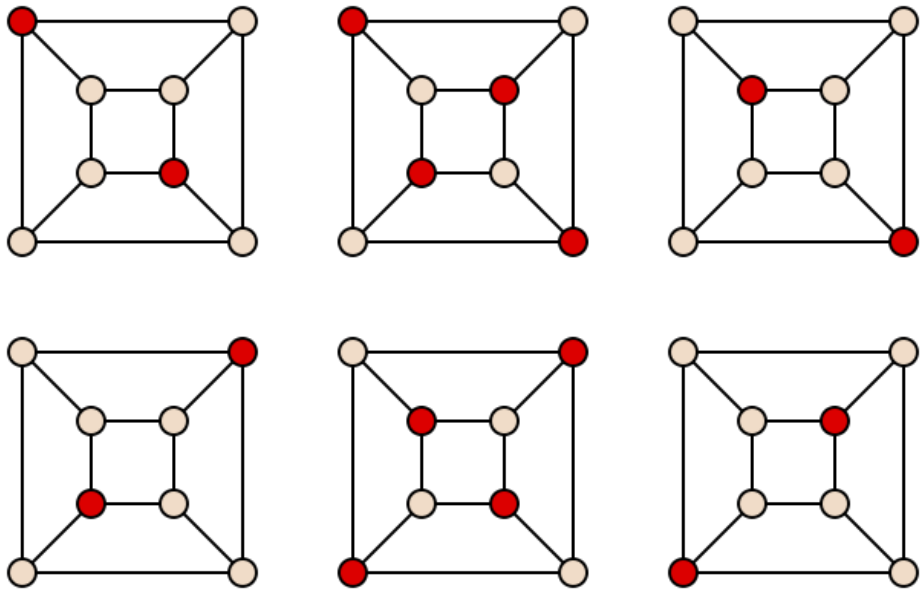
- MIS und HO sind sich sehr ähnlich, unterscheiden sich nur in der Definition der Überlappung
- Konstruieren aber beide einen Überlappungsgraphen
- Signifikanz ist dann die maximal unabhängige Menge des Überlappungsgraphen



1. Frequent-Subgraph-Mining

d) Signifikanzberechnung - MIS

- Maximal unabhängige Menge (MIS)= maximale Menge von Knoten in einem Graphen, zwischen denen keine Kante existiert

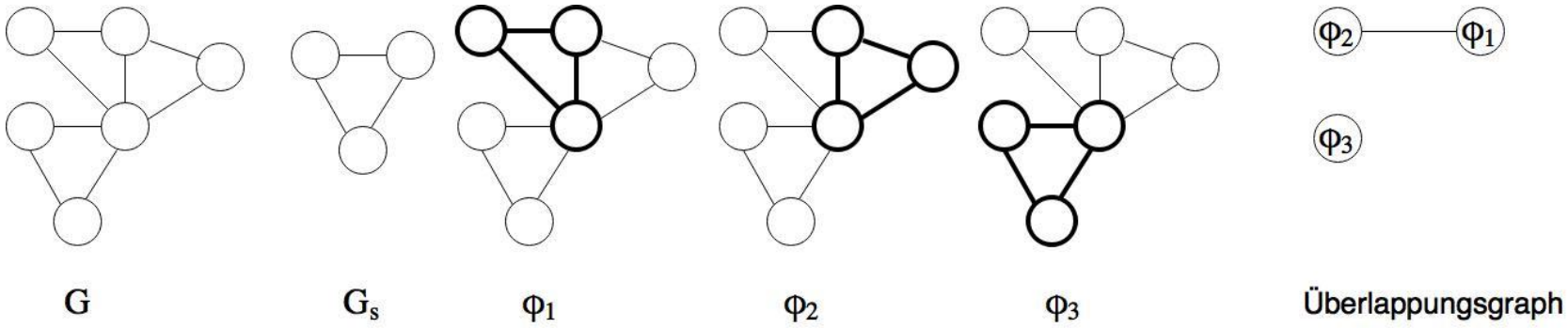


Quelle: https://en.wikipedia.org/wiki/Maximal_independent_set#/media/File:Cube-maximal-independence.svg

1. Frequent-Subgraph-Mining

d) Signifikanzberechnung

- MIS und HO sind sich sehr ähnlich, unterscheiden sich nur in der Definition der Überlappung
- Konstruieren aber beide einen Überlappungsgraphen
- Signifikanz ist dann die maximal unabhängige Menge des Überlappungsgraphen



Problem:

- Berechnung beider Varianten ist NP-vollständig

1. Frequent-Subgraph-Mining

d) Signifikanzberechnung

Lösung:

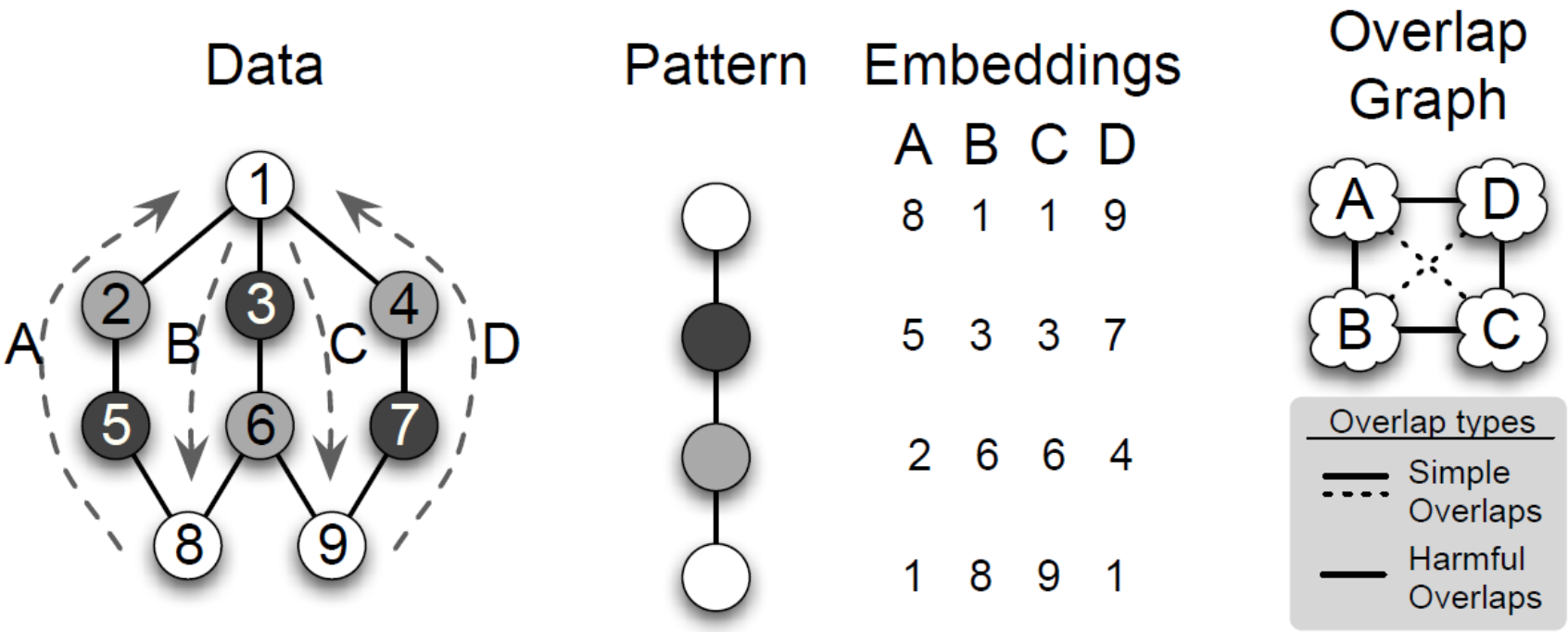
Definition 3.14 (Minimum-Image-Based-Support). *Gegeben ein Graph $G = (V, E)$ und einen Kandidaten $G_s = (V_s, E_s)$, dann ist der Minimum-Image-Based-Support definiert als:*

$$\sigma_{\wedge}(G_s, G) = \min_{v \in V_s} |\varphi_i(v) : \varphi_i \text{ ist ein Vorkommen von } G_s \text{ in } G|.$$

=> Basiert also auf der minimalen Anzahl der Knoten eines Kandidaten G_S, die den Knoten im Graph *G* zugeordnet werden können.

1. Frequent-Subgraph-Mining

d) Signifikanzberechnung

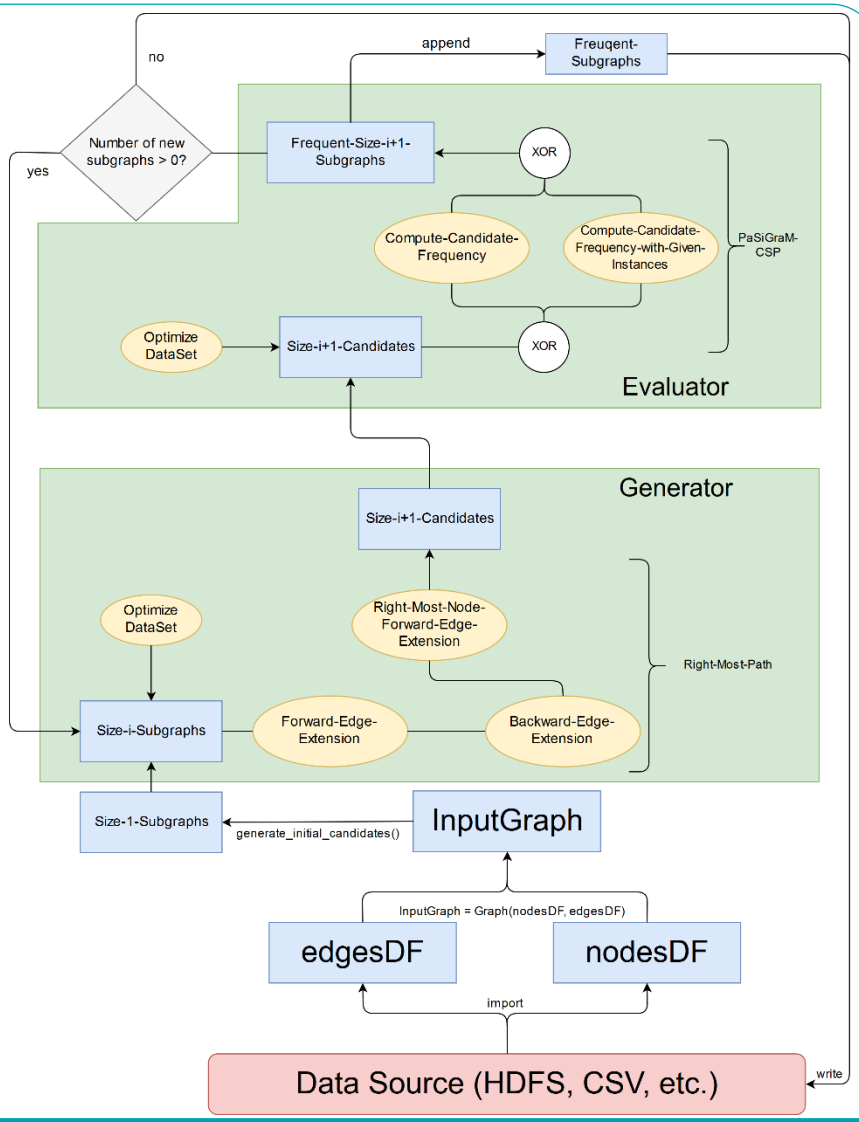


Quelle: *Bringmann, Björn* und *Nijssen, Sigfried*: *„What is frequent in a single graph?“* in *„Advanaces in Knowledge Discovery and Data Mining“*, Seiten 858-863, 2008

2. PaSiGraM

a) Allgemeiner Ansatz

- Kandidatengenerierung:
  - Forward-Edge-Extension
  - Backward-Edge-Extension
  - Right-Most-Node-Forward-Edge-Extension
- Signifikanzberechnung:
  - PaSiGraM-CSP
- Zusätzliche Heuristiken:
  - Optimierung der DataSets
- PaSiGraM arbeitet nach dem Prinzip der Breitensuche (level-basierte Generierung von Kandidaten und Signifikanzberechnung)



2. PaSiGraM

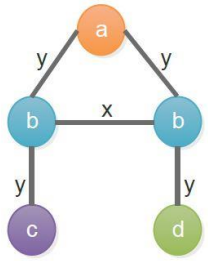
b) Datenstruktur

Graph
- nodes: Nodes - edges: Edges - adjacency_matrix: AdjacencyMatrix - adjacency_list: AdjacencyList - clusters: Clusters - canonical_code: String - csp_graph: DataFrame
+ countEdges(self): DataFrame + adjacencyMatrix(self): DataFrame + nodes(self): DataFrame + node_ids(self): List + edges(self): DataFrame + edge_ids(self): List + unique_edges(self): DataFrame + adjacency_list(self): DataFrame + node_degrees(self): DataFrame + clusters_by_label_and_degree(self): DataFrame + clusters_by_adjacency_list(self): DataFrame + canonical_code(self):String + csp_graph (self): DataFrame

2. PaSiGraM

b) Datenstruktur: CAM

- ein Graph wird durch seine *Canonical Adjacency Matrix* (kurz *CAM*) repräsentiert



G_1

a				
y	b			
y	x	b		
0	y	0	c	
0	0	y	0	d

M_1

a				
y	b			
y	x	b		
0	0	y	d	
0	y	0	0	c

M_2

b				
x	b			
y	0	d		
0	y	0	c	
y	y	0	0	a

M_3

- *lexikographisch kleinste* Adjazenzmatrix = *CAM* und wird auch als *kanonische Form* eines Graphen bezeichnet
- Aufwand für das Finden der *lexikographisch kleinste* Adjazenzmatrix = O(|V|!), wobei |V| die Anzahl der Knoten eines Graphen ist

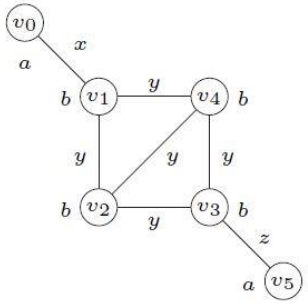
=> es werden Heuristiken benötigt, um nicht alle Permutationen der Adjazenzmatrix eines Graphen durchsuchen zu müssen

Quelle für das Bild : [QZL+18]

2. PaSiGraM

b) Datenstruktur: zusätzliche Heuristiken

- Knoten-Invarianten
  - Knotengrad und Labels
  - Adjazenzliste



(a)

b	y	y	y	0	0
y	b	y	y	0	0
y	y	b	0	x	0
y	y	0	b	0	z
0	0	x	0	a	0
0	0	0	z	0	a

p_0 p_1

code $= byby\,yby\,y0b00x0a000z0a$

(b)

b	y	y	y	0	0
y	b	y	y	0	0
y	y	b	0	x	0
y	y	0	b	0	z
0	0	x	0	a	0
0	0	0	z	0	a

p_0 p_1 p_2 p_3 p_4

code $= byby\,yby\,y0b00x0a000z0a$

(c)

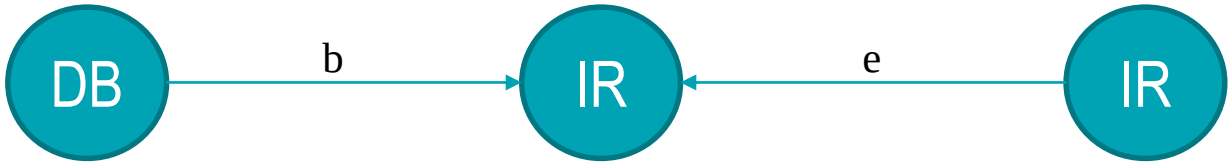
$(y, 3, b), (y, 3, b), (y, 3, b)$
$(y, 3, b), (y, 3, b), (y, 3, b)$
$(x, 1, a), (y, 3, b), (y, 3, b)$
$(y, 3, b), (y, 3, b), (z, 1, a)$
$(x, 3, b)$
$(z, 3, b)$

(d)

2. PaSiGraM

b) Datenstruktur: CSP-Graph

- Intern wird ein Graph durch seine Adjazenzliste repräsentiert
  - ist an die CAM angelehnt und übernimmt ihre Eigenschaften



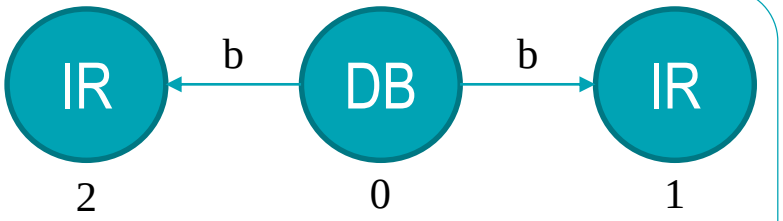
Node_id	Node_label	Indegree	Outdegree	Ingoing_neighbours	Outgoing_neighbours
0	DB	0	1	[]	[[b, IR, 1]]
1	IR	2	0	[[b, DB, 0], [e, IR, 2]]	[]
2	IR	0	1	[]	[[e, IR, 1]]

2. PaSiGraM

b) Datenstruktur: Cluster + kanonischer Code

- Der kanonische Code eines Graphen wird nicht durch seine *CAM* gebildet, sondern durch ein „Clustering-Verfahren“
- Dabei werden unterschiedliche Cluster für die Knoten gebildet, welche auf den Eigenschaften der Knoten (Knoten-Invarianten) beruhen (angelehnt an die Bildung durch die *CAM*)
  - Knotenlabel
  - Knoteneingangsgrad
  - Knotenausgangsgrad
  - Adjazenzliste

2. PaSiGraM



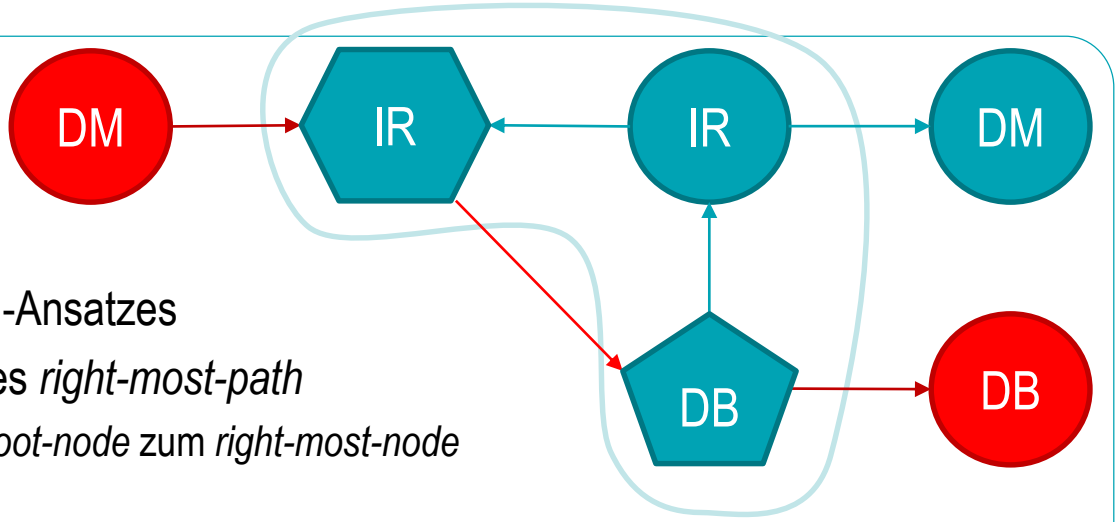
b) Datenstruktur: Cluster + kanonischer Code

Node_id	label	Indegree	Outdegree	elements
DB02	DB	0	2	[0]
IR10	IR	1	0	[1,2]

Node_id	label	Indegree	Outdegree	Ingoing_neighbours	Outgoing_neighbours	elements
DB02bIRbIR	DB	0	2	[]	[[b, IR, 1, 1, 0], [b, IR, 2, 1, 0]]	[0]
IR10bDB	IR	1	0	[[b, DB, 0, 0, 1]]	[]	[1,2]

Kanonischer Code = DB02bIRbIR#1#IR10bDB#2#

2. PaSiGraM



c) Kandidatengenerierung

- Nutzen des Pattern-Growth-Ansatzes
- basiert auf dem Konzept des *right-most-path*
  - der kürzeste Pfad vom *root-node* zum *right-most-node*
- Forward-edge-extension
  - Erweiterung des Graphen um eine Kante von einem existierenden Knoten im right-most-path zu einem neuen Knoten (Graph wird um neue Kante + neuen Knoten erweitert)
- Backward-edge-extension
  - Erweiterung des Graphen um eine Kante welche den *right-most-node* mit einem existierenden Knoten im *right-most-path* verbindet (Graph wird um neue Kante erweitert)
- Right-most-node-backward-edge-extension
  - Erweiterung des Graphen um eine Kante von einem neuen Knoten zum *right-most-node* (Graph wird um neue Kante + neuen Knoten erweitert)

2. PaSiGraM

d) Signifikanzberechnung - PaSiGraM-CSP

= Berechnung der Signifikanz (Häufigkeit) eines Teilgraphen in einem Input-Graphen

- (Teilgraph-)Isomorphie => NP-vollständig
- Zuordnungsproblem => *Constraint-Satisfaction-Problem* (*GraMi*)
- Erinnerung: jeder Graph wird durch seine *CAM* repräsentiert! (laut Theorie)
- *PaSiGraM-CSP* = eigene Variante eines *CSP*, angepasst auf das FSM
- beinhaltet zusätzliche Heuristiken die durch Eigenschaften der *CAM/CSP-Graph* gegeben sind

2. PaSiGraM

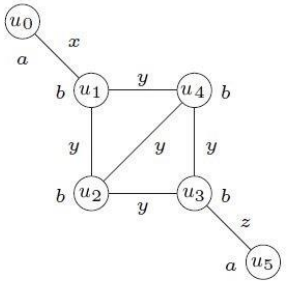
d) Signifikanzberechnung - PaSiGraM-CSP

Definition 6.8 (PaSiGraM-CSP). *Sei $G_s = (V_s, E_s, L_s)$ ein Teilgraph vom Graph $G = (V, E, L)$, $l(v) \in L$ das Label vom Knoten $v \in V$ und $l(v_s) \in L_s$ das Label vom Knoten $v_s \in V_s$. Sei $nl(v) = \{(l(e), nv)\}$ die Adjazenzliste eines Knotens $v \in V$ und $nl(v_s) = \{(l(e_s), nv_s)\}$ die Adjazenzliste eines Knotens $v_s \in V_s$, wobei $l(e)$ bzw. $l(e_s)$ die Label der Kanten $e \in E, e_s \in E_s$ und nv bzw. nv_s die Nachbarknoten, die über die Kanten e bzw. e_s mit v, v_s verbunden sind. Dann ist das PaSiGraM-CSP, ein $CSP = (X, D, C)$ für das gilt:*

1. *X enthält Variablen x_v für jeden Knoten $v_s \in V_s$.*
2. *D ist eine Menge von Domänen für jede Variable $x_v \in X$. Jede Domäne ist eine Teilmenge von V.*
3. *Seien in C die folgenden Bedingungen enthalten:*
   - *(a) $x_v \neq x'_v$, für alle $x_v, x'_v \in X$*
   - *(b) $l(v) = l(v_s)$*
   - *(c) $nl(v_s) \subseteq nl(v)$*

2. PaSiGraM

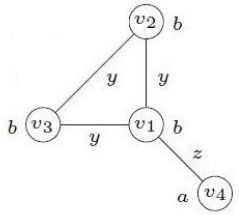
d) Signifikanzberechnung - PaSiGraM-CSP



(a)

	u_2	u_4	u_1	u_3	u_0	u_5
u_2	b	y	y	y	0	0
u_4	y	b	y	y	0	0
u_1	y	y	b	0	x	0
u_3	y	y	0	b	0	z
u_0	0	0	x	0	a	0
u_5	0	0	0	z	0	a
			$p0$		$p1$	

$(y,b),(y,b),(y,b)$
$(y,b),(y,b),(y,b)$
$(x,a),(y,b),(y,b)$
$(z,a),(y,b),(y,b)$
(x,b)
(z,b)



(b)

	v_1	v_2	v_3	v_4
v_1	b	y	y	z
v_2	y	b	y	0
v_3	y	y	b	0
v_4	z	0	0	a
	$p0$	$p1$	$p2$	

$(z,a),(y,b),(y,b)$
$(y,b),(y,b)$
$(y,b),(y,b)$
(z,b)

$$\Big((v_1,v_2,v_3,v_4), \{\{u_o,...,u_5\},...,\{u_0,...,u_5\}\}, \{v_1 \neq v_2 \neq v_3 \neq v_4, l(v_1)=l(v_2)=l(v_3)=b, l(v_4)=a \; nl(v_1)=\{(z,a),(y,b),(y,b)\}, nl(v_2)=nl(v_3)=\{(y,b),(y,b)\}, nl(v_4)=\{(z,b)\}\} \Big)$$

2. PaSiGraM

d) Signifikanzberechnung - PaSiGraM-CSP

Node_id	Node_label	Indegree	Outdegree	Ingoing_neighbours	Outgoing_neighbours
0	DB	0	2	[]	[[a, DM, 2]]
1	IR	1	1	[[f, IR, 1]]	[[c, DM, 2]]
2	DM	2	1	[[a, DB, 0], [c, IR, 1]]	[[d, DB, 3], [f, IR, 1]]
3	DB	1	2	[[d, DM, 2]]	[[b, IR, 4], [b, IR, 5]]
4	IR	1	0	[[b, DB, 3]]	[]
5	IR	1	1	[[b, DB, 3]]	[[e, DB, 6]]
6	DB	1	0	[[e, IR, 5]]	[]

Minimum image based support = 1

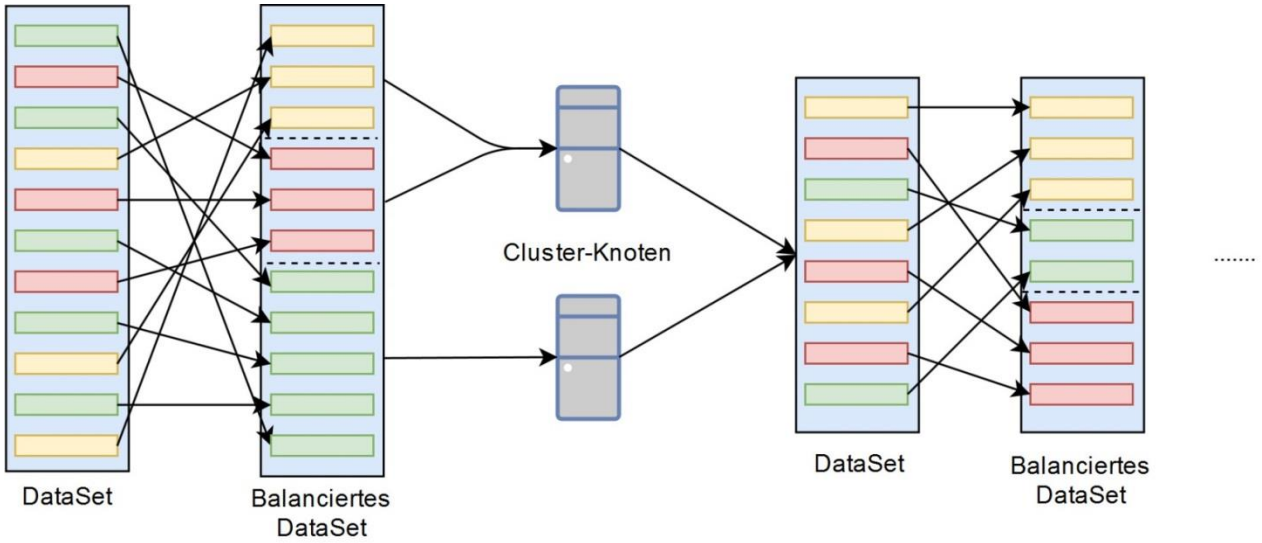
Node_id	Node_label	Indegree	Outdegree	Ingoing_neighbours	Outgoing_neighbours
0	DB	0	1	[]	[[b, IR, 1]]
1	IR	1	0	[[b, DB, 0]]	[]

2. PaSiGraM

e) Weitere Optimierungen – verbesserte Partitionierung der DataSets

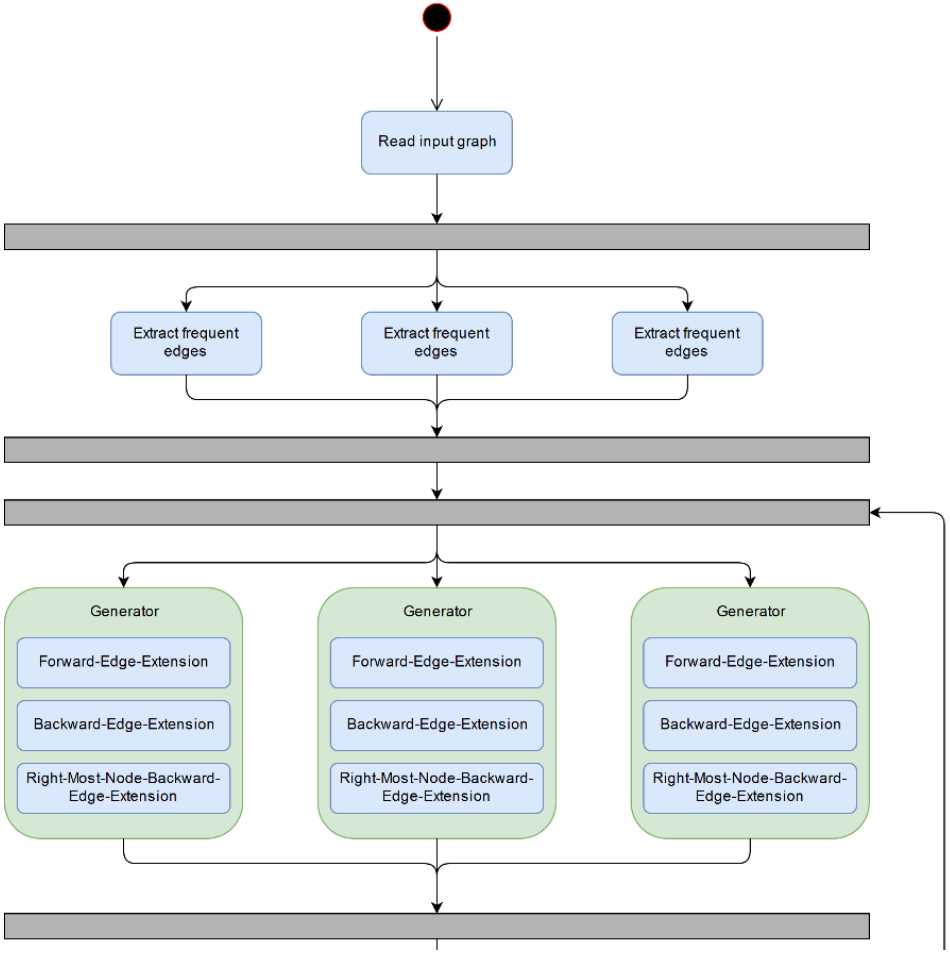
Zwei Probleme können auftreten:

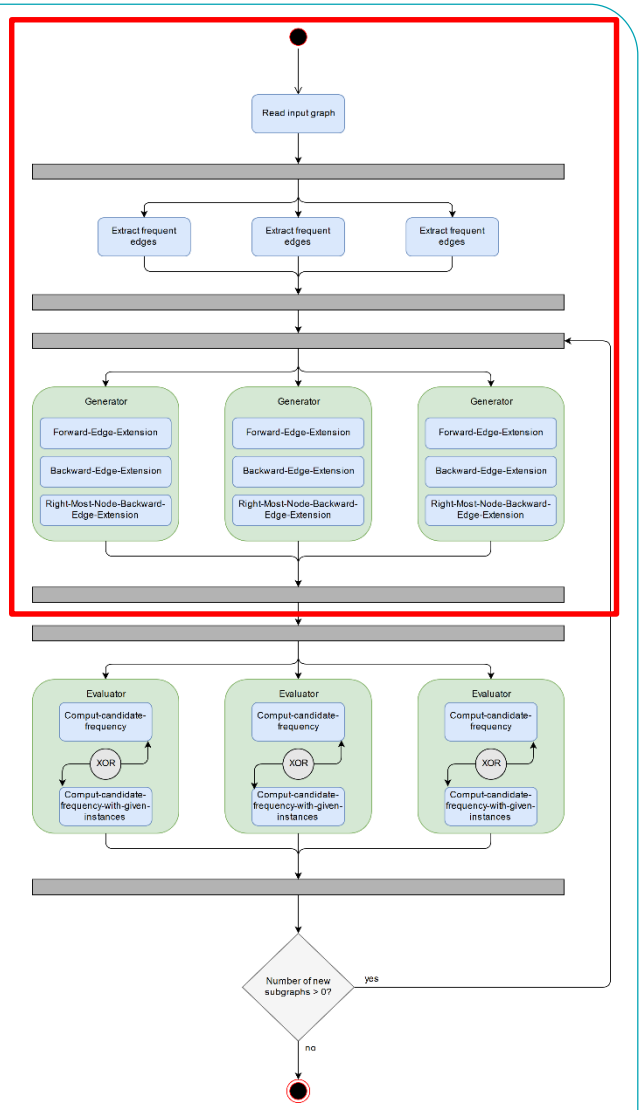
1. Ungleichmäßige Verteilung = ein Cluster-Knoten hat mehr Daten als alle anderen Knoten im Cluster
2. Ungünstige Aufteilung = bestimmten Cluster-Knoten fehlen benötigte Teile des jeweiligen Datensatzes



2. PaSiGraM

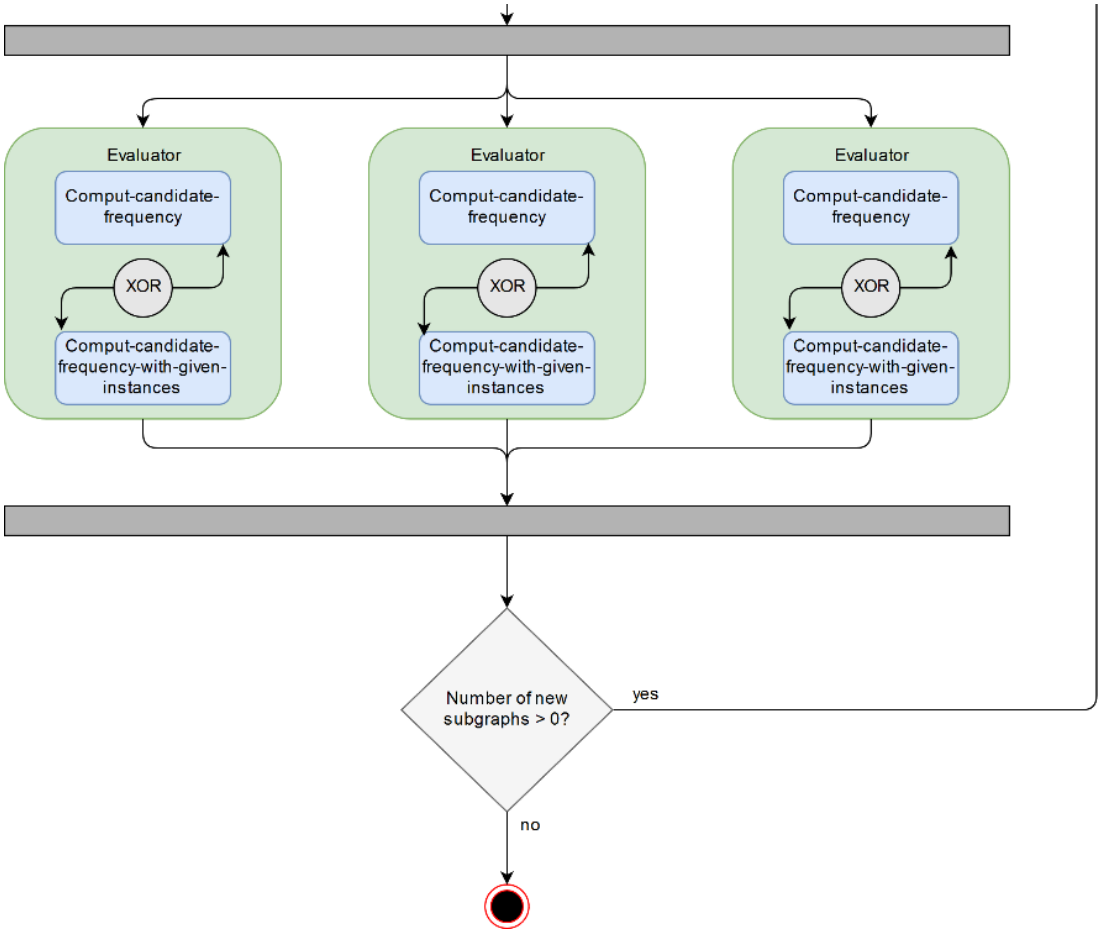
f) Parallelisierung

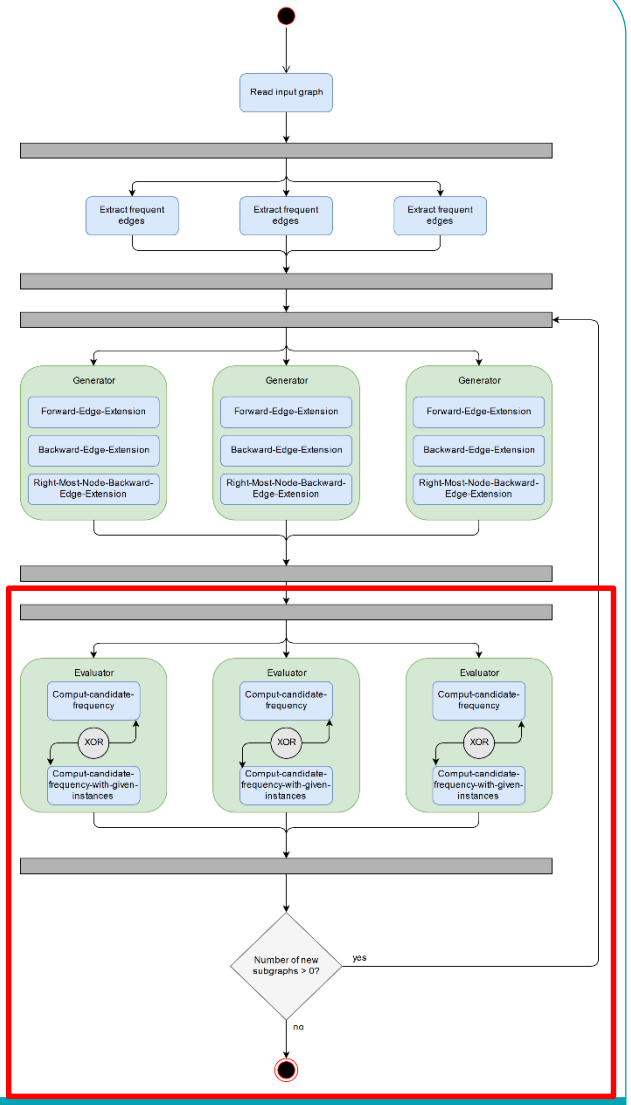




2. PaSiGraM

f) Parallelisierung





3. Zusammenfassung

- Implementierung von PaSiGraM war komplexer als gedacht
  - Viele Konzepte mussten nochmal überarbeitet werden
  - Vor allem die hohe Anzahl an Sonderfällen bei der Kandidatengenerierung und Signifikanzberechnung haben die Implementierung sehr erschwert
  - Auch die Definition einer komplett eigenen Graph-Datenstruktur war notwendig
- Dokumentation von Beginn an als wichtiger Bestandteil des Projekts
  - War dringend nötig durch die Komplexität des Projekts
  - Komplettes Projekt folgt einer Variante des MVC-Patterns
  - Einsetzen eines UML-Diagramms zur Planung der Strukturierung

4. Ausblick

- Veröffentlichen des PaSiGram-Algorithmus als Python-Bibliothek
- Integrieren der Parallelisierung in die Bibliothek
- Ausbauen der Input-Möglichkeiten für den Algorithmus
- Bauen eines *Cockpits* für die Performance-Analyse
- Einbinden der Bibliothek in eine Tool-Chain
- Hinzufügen von FSM auf einer Menge von Graphen

Vielen Dank für Ihre Aufmerksamkeit!