Musical-spoon

ein Package zum Alignment von Graphen

für das Seminar

Fortgeschrittene Methoden der Bioinformatik

Sommersemester 2019

von Max Feussner, Christopher Klapproth, Roman Matern und Joshua Stoppel

betreut von

Dr. Christian Höner zu Siederdissen

&

Prof. Dr. Peter F. Stadler

1. Einleitung

Wir wollen im Folgenden die Hauptmodule des musical-spoon Packages vorstellen und alle aufzurufenden Funktionen und Klassen mit ihren Inputparametern und Ausgaben anwendungsbezogen auflisten.

2. Zielsetzung

Folgende Aufgaben kann mit dem musical-spoon Package gelöst werden:

- .graph-Dateien können mit unserem Parser als Graph-Objekte eingelesen werden
- Einlesen von chemischen Graphen im JSON-Format aus PupChem
- Zufällige Graph-Objekte können erstellt werden und mit unserem Reverse-Parser als .graph-Datei ausgegeben werden
- Implementation einer Funktion, die Knoten/Kanten löschen kann
- Graph-Objekte können mittels der networkx-Library visualisiert werden
- Implementation von Bron-Kerbosch mit pivot (Pivot-Element entweder zufällig wählbar oder das Element mit den meisten Nachbarn wird als Pivot-Element bestimmt)
- Ankereingabe bei Bron-Kehrbosch implementiert
- Regeln für Knoten-Matching bei Bron-Kerbosch implementiert
- Implementation von Cordella
- Multiples Alignment von Graphen durch einen Guidetree
- Newick-format aus fertigen Guide-Trees erzeugen

Folgende Libraries (evtl. extern) werden zusätzlich benötigt:

- mendeleev
- numpy
- matplotlip.pyplot
- itertools
- networkx

3.1 Modul graph.py

Das Modul graph.py ist Teil unserer zugrundeliegenden Datenstruktur für Graphen. Jeder Graph wird als Objekt initialisiert, dem verschiedene Inputparameter übergeben werden können. Dazu gehören eine Liste aus Knoten, die als Vertex-Objekte des vertex.py Moduls dargestellt werden und eine Liste aus Kanten, die als Edge-Objekte des edge.py Moduls dargestellt werden. Weiterhin kann die Anzahl der Knoten und Kanten, sowie die Eigenschaft ob Label existieren, ob es sich um einen gerichteten Graphen handelt und ein Graphname als Attribute zugewiesen werden. Ein neuer gerichteter Graph *g* bestehend aus der Knotenliste *vlist* und der Kantenliste *elist* wird mit folgendem Befehl generiert:

g = Graph(vlist, elist, directed_graph=True, name="g")

Klassen:

3.1.1 Graph

3.1.1 Klasse: Graph(vertices, edges, no_of_vertices, no_of_edges, vertices_labelled, edges_labelled, directed_graph_ name)

Inputparameter	Typ	Default	Kurzbeschreibung
vertices	Liste	[]	Beinhaltet Vertex-Objekte
edges	Liste	[]	Beihnhaltet Edge-Objekte
no_of_vertices	Int	-	Anzahl der Knoten
no_of_edges	Int	-	Anzahl der Kanten
vertices_labelled	boolean	-	Gibt an ob Knoten Label besitzen
edges_labelled	boolean	-	Gibt an ob Kanten Label besitzen
directed_graph	boolean	-	Gibt an ob der Graph gerichtet ist
name	String	-	Name des Graophen

3.2 Modul vertex.py

Hiermit werden Knoten als Objekte dargestellt. Knoten verfügen über einen Namen, der einzigartig innerhalb eines Graphen sein sollte, und können optional Labels besitzen, beispielsweise um ein chemisches Element zu kennzeichnen. Zusätzlich besitzen Knoten eine Liste aus Vorgängerknoten und eine Liste aus nachkommenden Knoten, die für die Implementation des cordella.py Moduls benötigt wird. Diese beiden Listen werden automatisch bei der Edge-Objekterstellung ausgefüllt. ModularVertex-Objekte sind als Erweiterung von normalen Knoten zu sehen, da sich ihr Name aus den Namen zweier Knoten zusammensetzt. Ein neuer Knoten *v1* mit dem Label *„C“* wird mit folgendem Befehl generiert:

v1 = Vertex(v1, label=“C“)

Klassen:

3.2.1 Vertex
3.2.2 ModularVertex

3.2.1 Klasse: Vertex(name, label, successors, predecessors)

Inputparameter	Typ	Default	Kurzbeschreibung
name	String	-	Name des Vertex-Objekt. Muss einzigartig innerhalb eines Graphen sein.
label	String	-	Label des Knoten
successors	Liste	[]	Liste aus Vertex-Objekten die Vorgänger des Objektes sind. Wird automatisch erstellt.
predecessors	Liste	[]	Liste aus Vertex-Objekten die Nachkommen des Objektes sind. Wird automatisch erstellt.

3.2.2 Klasse: ModularVertex(name, vertex1, vertex2, label, predecessors, successors)

Inputparameter	Typ	Default	Kurzbeschreibung
name	String	-	Name des Vertex-Objekt. Muss einzigartig innerhalb eines Graphen sein.
vertex1	Objekt der Klasse Vertex	-	Wird automatische bei der Berechnung des Modularen Produkts ausgefüllt.
vertex2	Objekt der Klasse Vertex	-	Wird automatische bei der Berechnung des Modularen Produkts ausgefüllt.
label	String	-	Label des Knoten
successors	Liste	[]	Liste aus Vertex-Objekten die Vorgänger des Objektes sind. Wird automatisch erstellt.
predecessors	Liste	[]	Liste aus Vertex-Objekten die Nachkommen des Objektes sind. Wird automatisch erstellt.

3.3 Modul edge.py

Bei der Erstellung einer Kante werden jeweils zwei Knoten-Objekte übergeben. Dabei kann optional die Liste der Vorgänger-/Nachkommenknoten der betroffennen Knoten ausgefüllt werden. Eine neue Kante *e* zwischen den Knoten *v1* und *v2* mit dem Label *„einfachbindung“* wird mit folgendem Befehl generiert:

e = Edge(v1, v2, label=“einfachbindung“)

Klassen:

3.3.1 Edge

3.3.1 Klasse: Edge(vertex_a, vertex_b, label, directed_graph)

Inputparameter	Typ	Default	Kurzbeschreibung
vertex_a	Objekt der Klasse Vertex	-	Erster Vertex der Kante
vertex_b	Objekt der Klasse Vertex	-	Zweiter Vertex der Kante
label	String	-	Label der Kante
directed_graph	boolean	-	Wird für den Cordella-Algorithmus benötigt

3.4 Modul: parser2

Das Modul parser2 stellt mehrere Funktionen zur Verarbeitung von Graphen-Dateien dar. Ziel ist es eine Schnittstelle zwischen dem in 3.4.1 beschriebenen Graphen-Format und der internen Verarbeitung der Graphen als Graph-Objekt (siehe Abschnitt 3.1) zu ermöglichen. Des Weiteren ermöglicht das Modul die Ausgabe eines solchen Graph-Objekts in genanntem Graphen-Format. Außerdem ist es möglich eine chemische Struktur (im JSON-Format) einzulesen und diese als Graphobjekt intern weiter zu verarbeiten.

Eingabeformate:
- 3.4.1 Arbeitsgruppen-internes Graphen-Format
- 3.4.2 JSON-Dateien

Funktionen:
- 3.4.3 parse
- 3.4.4 reverse_parser
- 3.4.5 parse_chem

3.4.1 Arbeitsgruppen-internes Graphen-Format
Innerhalb der Seminargruppe “Fortgeschrittene Methoden der Bioinformatik“ (SS19) wurde sich auf folgendes Graphen-Format geeinigt:

```
#nodes;5
#edges;8
Nodes labelled;True
Edges labelled;False
Directed graph;False

1;a;2;5;4;3
2;b;5;4;3;1
3;c;1;2
4;d;1;2;5
5;e;4;1;2

1;2
1;5
1;4
1;3
2;3
2;4
2;5
4;5
```

Dieses Format dient der Speicherung aller relevanten Informationen eines Graphen und dient als Inputformat für die Funktion parse() [Abschnitt 3.4.3] sowie als Outputformat der Funktion reverse_parser() [Abschnitt 3.4.4]. Die Datei ist in drei Blöcke unterteilt, welche durch Leerzeilen separiert sind. Informationen innerhalb einer Zeile werden durch ein Semikolon voneinander getrennt.
In Block 1 werden allgemeine Informationen zum Graphen zusammengefasst (Anzahl der Knoten, Anzahl der Kanten, Knoten und Kantenlabel sowie die Information, ob es sich um einen gerichteten Graphen handelt.
In Block 2 sind die Knoten des Graphen aufgelistet. Jede Zeile stellt einen neuen Knoten dar. Jeder Knoten beginnt mit dem jeweilen Namen, gefolgt von dem Label (falls in Block 1 definiert) sowie den benachbarten Knoten.
In Block 3 sind die Kanten des Graphen aufgelistet. Jede Zeile stellt eine neue Kante dar. Jede Kante ist durch die beiden Knoten definiert, welche sie verbindet. Im Falle gerichteter Graphen ist

die Kante vom erstgenannten Knoten zum zweitgenannten Knoten definiert, gefolgt vom Label der Kante.

3.4.2 JSON-Dateien


```
{
  "PC_Compounds": [
    {
      "id": {
        "id": {
          "cid": 962
        }
      },
      "atoms": {
        "aid": [
          1,
          2,
          3
        ],
        "element": [
          8,
          1,
          1
        ]
      },
      "bonds": {
        "aid1": [
          1,
          1
        ],
        "aid2": [
          2,
          3
        ],
        "order": [
          1,
          1
        ]
      },
      .
      .
      .
      .
    }
  ]
}
```


Das JSON-Format ist eins der 4 möglichen Formate um chemische Strukturen aus der PubChem-Datenbank zu speichern. Es dient der Funktion parse_chem() als Eingabe.

3.4.3 parse(filename)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
filename	String	-	Dateipfad und Name der Textdatei, die die Grapheninformationen beinhaltet

Die Funktion parse() dient dem Einlesen eines Graphen aus dem festgelegten Graphen-Format zur internen Weiterverarbeitung. Sie erstellt ein Objekt der Klasse Graph (s. Abschnitt 3.1) aus den in der Input-Textdatei enthaltenen Informationen. Die Inputdatei muss zwingend dem in Abschnitt 3.4.1 beschriebenen Format vorliegen.

3.4.4 reverse_parser(g)

Rückgabe: Graphen-Format (siehe 3.4.1) als Standard-Output

Inputparameter	Typ	Default	Kurzbeschreibung
g	Graph	-	Das zu speichernde Graph-Objekt.

Die Funktion reverse_parser() ermöglicht die Ausgabe eines Programm-intern verwendeten Graph-Objekts in dem festgelegten Graphen-Format. Zur Speicherung der Informationen muss der STDOUT des Programms in einer Textdatei gespeichert werden. Dazu im Terminal main.py >> output.graph verwenden.

3.4.5 parse_chem(filename)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
filename	String	-	Dateipfad und Name der JSON-Datei, die die Informationen über die einzulesende chemische Struktur beinhaltet

Ähnlich der Funktion parse() wird mit parse_chem() ein Graph-Objekt aus einer Inputdatei erzeugt und damit der Programm-internen Verarbeitung zugänglich gemacht. Die Besonderheit ist, dass die Knoten- und Kanten-Labels den Informationen der chemischen Struktur zugeordnet sind. So entspricht das Label eines jeden Knoten dem entsprechenden Atom in der chemischen Struktur, dargestellt durch seine Ordnungszahl. Die Label der Kanten stellen die Art der chemischen Bindung dar. So entspricht das Kantenlabel „1“ einer Einfachbindung und das Kantenlabel „2“ einer Doppelbindung. Die Funktion parse_chem() ermöglich somit die Verarbeitung chemischer Strukturen in Form eines internen Graph-Objektes.

3.5 Modul show_graph.py

Durch die Nutzung der matplotlib und networkx Library können mittels der Funktionen aus show_graph.py Graphen anschaulich dargestellt werden. Als Eingabe benötigen die Funktionen Graphen-Objekte. Wird eine Funktion aufgerufen öffnet sich ein Anzeigefenster mit dem darzustellenden Graphen. Während das Anzeigefenster geöffnet ist pausiert das restliche Programm und läuft erst weiter sobald das Anzeigefenster geschlossen wurde.

Funktionen:

3.5.1 **show_graph(g)**

Zeigt den Graphen *g* in einem Fenster an.

3.5.2 **show_two_graphs(g1, g2)**

Zeigt nebeneinander die Graphen *g1* (links) und *g2* (rechts) in einem Fenster an.

3.5.3 **show_graph_comparable(g1, g2, g3)**

Zeigt in einem Fenster die Graphen *g1* (links oben), *g2* (rechts oben) und *g3* (mitte unten) an.

3.6 Modul: random_graph

Das Modul random_graph stellt eine Reihe von Funktionen zur zufälligen Generation und Modifikation von Graphobjekten dar. Hierbei können einige Inputparameter übergeben werden, um den zurückgegebenen Graphen den jeweiligen Bedürfnissen anzupassen. Insbesondere ist es möglich, individuelle Kanten wahrscheinlichkeitsbasiert aus dem Graphen zu entfernen. Es werden über eine Reihe von aufrufbaren Funktionen vier Haupttypen von erzeugbaren Graphen unterschieden (s. u.). „Normale“ Zufallsgraphen bestehen aus einer Anzahl vorgegebener Knoten, die zufällig durch Kanten miteinander verbunden werden. „Schachbrettgraphen“ und „Trianguläre Graphen“ erhalten ein Set aus Kanten, dass die Knoten in bestimmten geometrischen Formen ordnet. Der vollständige Graph ist nach der gängigen Definition ein Graph, in dem jeder Knoten mit allen anderen Knoten durch eine Kante verbunden ist. Die Funktion cut_edges erlaubt weiterhin auch das Entfernen von kanten an einem bereits erzeugten Graph.

Funktionen:

3.6.1 random_graph(lower_node_limit, upper_node_limit, more_edges)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
lower_node_limit	Int	1	Die Mindestanzahl an Knoten, die der zu generierende Graph enthalten wird.
upper_node_limit	Int	10	Die Maximalanzahl an Knoten, die der zu generierende Graph enthalten wird.
more_edges	boolean	False	Eine Möglichkeit, „dichtere“ Graphen zu erzeugen. Die Anzahl der zufällig eingefügten Kanten verdoppelt sich im Mittel. Für eine genauere Feineinstellung der Kantenzahl, kann die Funktion cut_edges verwendet werden.

Die Funktion random_graph erhält als Eingabe ein Intervall, aus dem die Anzahl der Knoten bestimmt wird, die anschließend wahrscheinlichkeitsbasiert miteinander verbunden werden. Die Rückgabe erfolgt in Form eines Objekts der Klasse Graph. Hierbei wird vermeiden, einzelne Knoten zu isolieren, voneinander getrennte Teilgraphen mit Knotenzahl > 2 sind allerdings möglich. Üblicherweise wird random_graph einen Graph entwerfen, dessen Kantenzahl sich proportional zur Knotenzahl verhält. Der Parameter more_edges erhöht die Anzahl der zufällig gezogenen Kanten im statistischen Mittel um den Faktor 2. Die Funktion wird keine vollständigen Graphen erzeugen, hierfür ist die Funktion complete_graph zu verwenden, je nach Zielsetzung evtl. in Konjunktion mit der Funktion cut_edges.

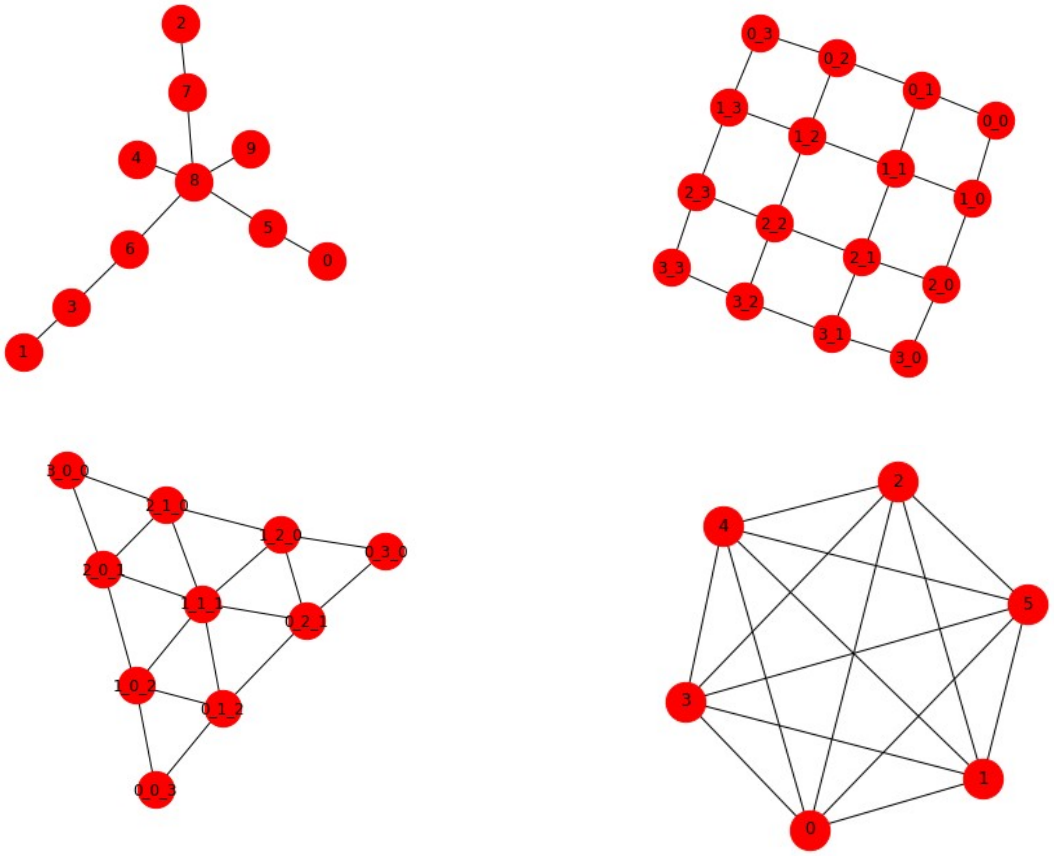

Abbildung 3.6.1: 4 beispielhaft ausgewählte Graphen, die mit dem random_graph Modul erzeugt wurden. Oben links: Funktion random_graph mit Inputparametern: lower_node_limit = 10, upper_node_limit = 10, more_edges = False. Oben rechts: Funktion random_chess_graph mit Inputparametern lower_node_limit = 16, deletion_chance = 0. Unten links: Funktion random_triangular_graph mit Inputparametern lower_node_limit = 10, deletion_chance = 0. Unten rechts: complete_graph mit Inputparameter nodes = 6.

3.6.2 random_chess_graph(lower_node_limit, deletion_chance)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
lower_node_limit	Int	4	Die Mindestanzahl an Knoten, die der zu generierende Graph enthalten wird.
deletion_chance	Int	0	Die Wahrscheinlichkeit, dass eine gegebene Kante nicht eingefügt wird. Dieser Parameter ist als Shortcut gegenüber der Funktion cut_edges (s. u) zu betrachten, da er im Wesentlichen die selbe Aufgabe erfüllt. Der Eingabewert muss im Intervall [0,1] liegen. Je höher der Wert, desto weniger Kanten hat der erzeugte Graph.

Die Funktion random_chess_graph erzeugt ein Graphobjekt, dass sich geometrisch an einem Schachbrettmuster orientiert (siehe Abbildung 3.6.1). Zu beachten ist, dass die Funktion immer versuchen wird, ein vollständiges Schachbrett zu generieren, d. h. in Abhängigkeit vom eingegebenen unteren Knotenlimit könnten mehr Knoten eingefügt werden, um ein vollständiges Muster zu erhalten. Der Default für die Erzeugung der Kanten sieht ein vollständiges Schachbrettmuster vor, allerdings kann durch Veränderungen am Parameter deletion_chance (s. o.) eine wahrscheinlichkeitsbasierte Anzahl an Kanten entfernt werden. Dies erzeugt einen unvollständigen Schachbrettgraph (siehe Abbildung 3.6.2).

3.6.3 random_triangular_graph(lower_node_limit, deletion_chance)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
lower_node_limit	Int	3	Die Mindestanzahl an Knoten, die der zu generierende Graph enthalten wird.
deletion_chance	Int	0	Die Wahrscheinlichkeit, dass eine gegebene Kante nicht eingefügt wird. Dieser Parameter ist als Shortcut gegenüber der Funktion cut_edges (s. u) zu betrachten, da er im Wesentlichen die selbe Aufgabe erfüllt. Der Eingabewert muss im Intervall [0,1] liegen. Je höher der Wert, desto weniger Kanten hat der erzeugte Graph.

Ähnlich der Funktion random_chess_graph wird ein Graphobjekt erzeugt, dessen Kanten einer geometrischen Struktur angepasst werden (siehe Abbildung 3.6.1). Die Inputparameter verhalten sich hier analog, lower_node_limit korrespondiert mit der Mindestanzahl der zu erzeugenden Knoten, welche zur Vervollständigung des Musters von der Funktion erhöht werden kann. Der Default für die Erzeugung der Kanten sieht ein vollständiges Dreiecksmuster vor, allerdings kann

durch Veränderungen am Parameter deletion_chance (s. o.) eine wahrscheinlichkeitsbasierte Anzahl an Kanten entfernt werden.

3.6.4 complete_graph(nodes)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
nodes	Int	10	Die Anzahl an Knoten, die der zu generierende Graph enthalten wird.

Die Funktion complete_graph generiert einen vollständigen Graphen mit der für den Inputparameter nodes vorgegebenen Anzahl an Knoten (siehe Abbildung 3.6.1). Ein wahrscheinlichkeitsbasiertes Element liegt hier nicht vor, da für einen vollständigen Graph alle Knoten verknüpft sein müssen.

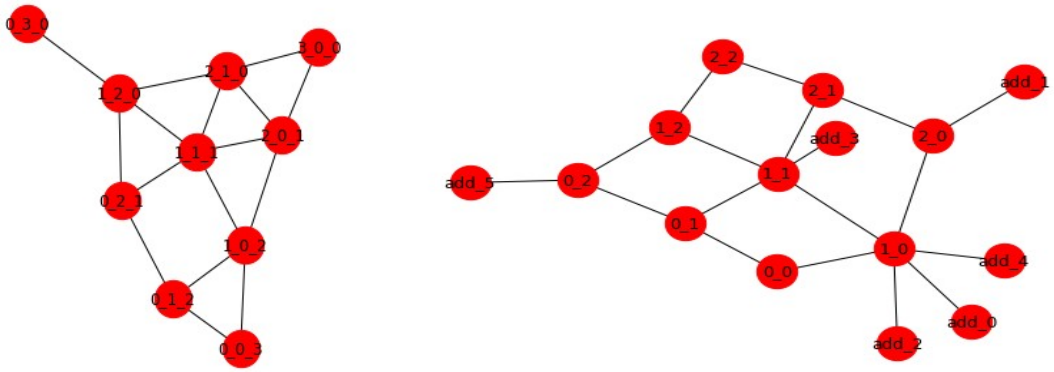

Abbildung 3.6.2: Beispiele für unter Verwendung der Funktionen cut_edges und add_random_nodes veränderte Graphen. Links: Ein Triangelgraph nach Anwendung der Funktion cut_edges mit Inputparameter deletion_chance = 0.2. Rechts: Ein Schachbrettgraph nach Anwendung der Funktion add_random_nodes mit Inputparameter n = 6.

3.6.5 cut_edges(graph, deletion_chance)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
graph	Graph	-	Das zu verändernde Graphobjekt.
deletion_chance	Int	0.1	Die Chance für jede individuelle Kante, aus dem vorliegenden Graph entfernt zu werden. Die Eingabe sollte sich im Intervall [0,1] befinden.

Mit der Funktion cut_edges können wahrscheinlichkeitsbasiert Kanten aus einem Graphobjekt gelöscht werden. Der Parameter deletion_chance kontrolliert dabei die individuelle Wahrscheinlichkeit jeder einzelnen Kante, entfernt zu werden. Hierbei ist es möglich, dass einzelne Knoten oder Teilgraphen isoliert werden. Diese Funktion eignet sich zum Trimmen oder randomisieren von bekannten Graphen. Der Inputparamter deletion_chance der beiden vorherigen Funktionen erfüllt im Wesentlichen die selbe Funktion.

3.6.6 add_random_nodes(graph, n)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
graph	Graph	-	Das zu verändernde Graphobjekt.
n	Int	1	Die Anzahl der zusätzlich in den Graphen einzufügenden Knoten.

Eine Funktion zum zufallsbasierten Einfügen einer vorgegebenen Anzahl an Knoten in einen Graphen. Jeder Knoten wird zufällig mit einem bereits im Graph vorliegenden Knoten verknüpft. Zur späteren Identifikation werden die Bezeichner der neuen Knoten um das Präfix „add_“ ergänzt.

3.7 Modul bronk_pivot.py

Dieses Modul besitzt mehrere Funktionen zur *MCIS* Bestimmung zweier Inputgraphen. Dies erfolgt intern durch die Berechnung des Modularproduktes der Inputgraphen und darauffolgender Cliquensuche durch den Bron-Kerbosch-Algorithmus. Die gefundenen Knotenmatches werden in Textform in der Konsole angegeben und zusätzlich ein Graphenobjekt zurückgegeben, das z.B. mit dem show_graph.py Modul visualisiert werden kann.

Funktionen:

3.7.1 find_mcis
3.7.2 find_mcis_without_prompt
3.7.3 find_ankered_mcis

3.7.1 find_mcis (graph1, graph2, checklabels, choose_pivot_randomly)

Rückgabe: Ein Objekt der Klasse Graph und Textausgabe im Terminal von gepaarten Knoten

Inputparameter	Typ	Default	Kurzbeschreibung
graph1	Graph	-	Zu matchender Graph Nummer 1
graph2	Graph	-	Zu matchender Graph Nummer 2
checklabels	boolean	False	Ermöglicht Matching nur von Knoten mit identischen Labels wenn True
choose_pivot_randomly	boolean	False	Bestimmt die Wahl des Pivot-Elements im Bron-Kerbosch-Algorithmus

Als Eingabe benötigt diese Funktion die zwei Graphenobjekte *graph1* und *graph2*. Der Parameter *checklabels* ist optional und ist per Default False. Er sollte nur auf True gesetzt werden wenn beide Inputgraphen die Anforderung *vertices_labelled*=True erfüllen, denn dann ergeben nur Knoten mit identischen Labels einen Match. Der Parameter *choose_pivot_randomly* beeinflusst die Wahl des Pivot-Elementes des Bronk-Kerbosch-Algorithmus. Wenn der Parameter True übergibt, wird das Pivot-Element zufällig gewählt. Per Default ist der Parameter False, wodurch der Knoten mit den meisten Nachbarknoten als Pivot-Element ausgewählt wird. Bei erfolgreicher Durchführung gibt das Programm im Terminal eine Textausgabe der gepaarten Knoten des ersten gefundenen MCIS und auch das zugehörige Graph-Objekt zurück. Sollte es mehr als ein MCIS mit der gleichen Knotenanzahl geben, so wird gefragt ob diese auch ausgegeben werden sollen. Wird die Eingabeaufforderung mit „y“ beantwortet, wird der Bron-Kerbosch-Algorithmus ein zweites Mal durchgeführt und alle MCIS im Terminal als Text ausgegeben.

3.7.2 find_mcis_without_prompt(graph1, graph2, checklabels, choose_pivot_randomly)

Besitzt die selbe Funktionsweise wie *find_mcis()*, gibt jedoch nur den ersten gefundenen *MCIS* in der Konsole und als Graphenobjekt zurück und terminiert dann.

3.7.3 find_ankered_mcis(graph1, graph2, anker, checklabels, choose_pivot_randomly)

Rückgabe: Ein Objekt der Klasse Graph und Textausgabe im Terminal von gepaarten Knoten

Inputparameter	Typ	Default	Kurzbeschreibung
graph1	Graph	-	Zu matchender Graph Nummer 1
graph2	Graph	-	Zu matchender Graph Nummer 2
Anker	Liste	-	Beinhaltet gewollte Knotenpaarungen als Strings
checklabels	boolean	False	Ermöglicht Matching nur von Knoten mit identischen Labels wenn True
choose_pivot_randomly	boolean	False	Bestimmt die Wahl des Pivot-Elements im Bron-Kerbosch-Algorithmus

Besitzt die selbe Funktionsweise wie *find_mcis_with_prompt()*, bietet jedoch die Möglichkeit die Liste *anker*, welche gewollte Knotenmatches enthält, beizufügen. Soll z.B. ein Matching zwischen den Knoten *v1a* von *graph1* und *v1b* von *graph2* sowie zwischen den Knoten *v2a* von *graph1* und *v2b* von *graph2* erzwungen werden, so sieht die Liste *anker* wie folgt aus:

anker = [„v1a;v1b“, „v2a;v2b“]

Essentiell für ein richtiges Ergebnis der *MCIS* Analyse ist, dass die Knotenmatches in *anker* möglich sind.

3.8 Modul: Cordella_max

In dem Modul Cordella wird der Algorithmus implementiert, welcher in dem Paper: An Improved Algorthim for Matching Large Graphs von Cordella et al. beschrieben wird. Dieser Algorithmus dient zum Finden von Graph und Graph-Subgraph Isomorphismen. Dieser Funktion können zwei Graphen übergeben werden, welche darauf hin auf die oben genannten Isomorphismen untersucht werden. Falls ein Matching gibt, werden die zueinander gemachten Knoten der beiden Graphen ausgegeben. Der Algorithmus ist vor allem durch seinen kleinen Platzverbauch für große Graphen geeignet. Da er durch eine spezielle Implementierung einen Platzverbrauch von nur O(n) aufweist. In dem Modul finden sich nicht nur der Algorithmus zum Matchen von Graphen, sondern auch noch die Funktionen „create_output_table“. Diese Funktion gibt alle gematchen Paare von Knoten der beiden input Graphen aus. Außerdem noch die Funktionen „find_successors“ und „find_predecessors“, welche die die Vorfahren und Nachfahren der übergebenen Konten finden.

Funktionen:

3.8.1 Cordella(g1, g2)
Rückgabe: Keine

Inputparameter	Typ	Default	Kurzbeschreibung
g1	Graph	-	Graphobjekt, welches gematcht werden soll
g2	Graph	-	Graphobjekt, welches gematcht werden soll

Diese Funktion initialisiert das Matching nach dem verbesserten Cordella-Algorithmus. Der Output wird in Form einer Tabelle generiert , die die einzelnen Knotenpaare nach ihren Bezeichnern im Graphobjekt auflistet. Ein Output in Form eines Graphobjekts wurde hier nicht für sinnvoll erachtet, da Cordella ein vollständiges Matching eines Graphen auf den anderen anstrebt (i.e. der Output wäre deckungsgleich mit mindestens einem Inputgraphen).

3.8.2 find_successors(v)
Rückgabe: Liste von Namen der Nachfolger des Inputknotens

Inputparameter	Typ	Default	Kurzbeschreibung
v	String	-	Name des Knotens, dessen Nachfolger bestimmt werden sollen

Diese Funktion ist eine interne Routine des Cordella-Algorithmus und erzeugt eine Liste aller direkten Nachfolger des Eingabeknotens im dazugehörenden Graphen. Diese Funktion wird benötigt um für den Cordella-Algorithmus den Suchraum einzugrenzen. Es ist allerdings denkbar, dass diese Funktion auch unabhängig davon einen Nutzen erbringen kann, weshalb sie hier mit aufgeführt wird.

3.8.3 find_predecessors(v)

Rückgabe: Liste von Namen der Vorgänger des Inputknotens

Inputparameter	Typ	Default	Kurzbeschreibung
v	String	-	Name des Knotens, dessen Vorgänger bestimmt werden sollen

Diese Funktion ist eine interne Routine des Cordella-Algorithmus und erzeugt eine Liste aller direkten Vorgänger des Eingabeknotens im dazugehörenden Graphen. Diese Funktion wird benötigt um für den Cordella-Algorithmus den Suchraum einzugrenzen. Es ist allerdings denkbar, dass diese Funktion auch unabhängig davon einen Nutzen erbringen kann, weshalb sie hier mit aufgeführt wird.

3.9 Modul: guide_tree

Das Modul guide_tree dient der Implementierung einer Klasse von progressiven multiplen Alignments von Graphen. Berechnete Bäume können ihrerseits als Graphobjekt gespeichert und angezeigt werden. Jedem Baum wird ein eindeutiger *result*-Parameter zugeordnet, der den größten gefundenen gemeinsamen Subgraph des Alignments enthält. Die Berechnung erfolgt dabei automatisch bei der Erzeugung des Baum-Objekts, d. h. die Erzeugung des Baumes und die Generation des Alignments sind in dieser Implementierung untrennbar verknüpft. Die Eingabe der zu alignierenden Graphen erfolgt dabei im Format einer einfachen Python-Liste und ist in ihrem Umfang theoretisch nach oben hin unbegrenzt. Es ist allerdings zu bedenken, dass für die Berechnung der Subgraphen intern eine Variation des Bronk-Kerbosch-Algorithmus verwendet wird (siehe 3.7), weshalb die Performance für die Alignmentberechnung insbesondere bei großen Graphen mitunter ungünstig ausfällt (mehr dazu unter Abschnitt 4.1). Im Kern steht bei der Erzeugung des Guide Trees das Aufstellen einer Scoringmatrix, die eine paarweise Bewertung aller vorliegenden Graphen nach einer Ähnlichkeitsfunktion (score_similarity) enthält. Auf Basis dieser Matrix werden initial zu alignierende Paare von Graphen gebildet, deren größte gemeinsame Subgraphen die Knoten der nächsten Alignmentstufe darstellen. Dieser Prozess wird fortgesetzt, bis die Wurzel des Baumes erreicht ist. Das an diesem Punkt vorgenommene Alignment entspricht dann dem Alignmentresultat des Baumes, das unter dem Attribut *result* abgespeicherte Graphobjekt entspricht also dem größten gemeinsamen induzierten Subgraphen der Inputgraphen nach Berechnung des guide_tree Moduls.
Das Modul stellt auch Methoden zur Traversierung eines von außen eingegebenen Guide Trees zur Verfügung, auch wenn diese Funktion für mit diesem Modul erzeugte Bäume nicht erforderlich ist. Sie könnte allerdings Nutzen im Hinblick auf einen Vergleich mit nach anderen Algorithmen errechneten Guide Trees erbringen.

Klassen:

Funktionen:

3.9.1 Klasse: guide_tree(graph_list, construct_tree, name)

Attribut	Typ	Default	Kurzbeschreibung
graph_list	Liste	-	Eine Liste von zu alignierenden Graphobjekten.
construct_tree	boolean	True	True, falls das Programm eine Repräsentation des Guide Trees als Graphobjekt erzeugen soll. Kann False gesetzt werden, falls nur das Ergebnis des Alignments, sprich der größte gemeinsame induzierte Subgraph gesucht wird.
name	String	None	Der Guide Tree kann individuell benannt werden. Eine optionale Eingabe.
result	Graph	-	Hält den größten gemeinsamen induzierten Subgraph des Alignments. Wird automatisch mit der Erzeugung des Guide Tree berechnet.
tree_structure	Graph	-	Eine Repräsentation des Guide Tree als Graphobjekt. Die Erzeugung ist optional und wird über das Attribut construct_tree geregelt.

3.9.2 create_tree(graph_list, draw_tree, name)

Rückgabe: Ein Objekt der Klasse guide_tree

Inputparameter	Typ	Default	Kurzbeschreibung
graph_list	Liste	-	Eine Liste von zu alignierenden Graphobjekten.
draw_tree	boolean	True	True, falls das Programm eine Repräsentation des Guide Trees als Graphobjekt erzeugen soll. Kann False gesetzt werden, falls nur das Ergebnis des Alignments, sprich der größte gemeinsame induzierte Subgraph gesucht wird.
name	String	None	Der Guide Tree kann individuell benannt werden. Eine optionale Eingabe.

Eine Brückenfunktion, die die Initialisierung eines guide_tree Objekts startet. Das Objekt kann bei Bedarf auch manuell erzeugt werden.

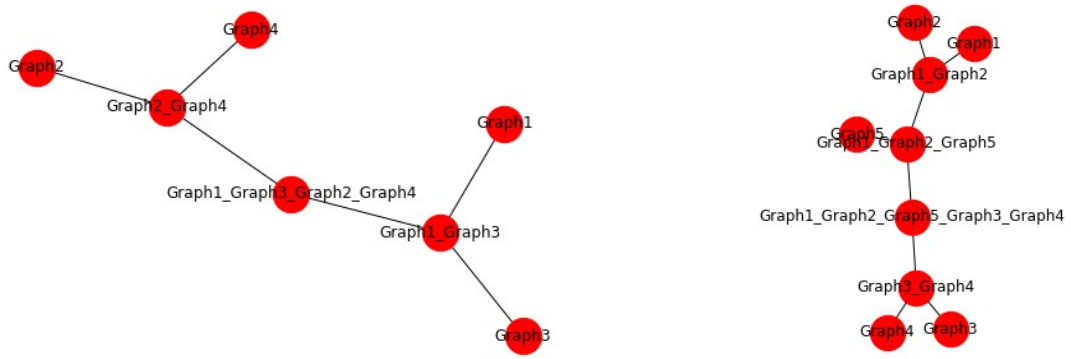

Abbildung 3.9.1: Graphenrepräsentationen von Guide Trees, links mit 4 Graphen als Eingabe, rechts mit 5 Graphen. Die Blätter der Bäume stellen die individuellen Inputgraphen dar, die inneren Knoten listen alle bis zu diesem Punkt im Alignment einbezogenen Graphen getrennt durch Unterstriche auf.

3.9.3 traverse_tree(tree, graph_list)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
tree	guide_tree	-	Ein externes guide_tree Objekt. Bei der Erzeugung des Objekts muss ein Guide Tree als Graphobjekt erzeugt worden sein, dass unter dem Attribut tree_structure gespeichert ist. Die in dieser Repräsentation angegebenen Graphobjekte müssen namentlich mit den Graphobjekten in der im nächsten Punkt übergebenen Liste korrespondieren.
graph_list	Liste	-	Eine Liste von Graphobjekten. Muss mit dem obigen guide_tree Objekt korrespondieren.

Diese Funktion stellt eine Möglichkeit dar, einen Baum der Klasse guide_tree manuell zu durchlaufen. Für die meisten Anwendungen des Packages wird dies nicht erforderlich sein, da die Berechnung des Baumes automatisch bei seiner Initialisierung erfolgt. Diese Funktion stellt allerdings eine wichtige Vergleichsmöglichkeit mit Guide Trees aus anderen Quellen/früheren Versionen her. Für die korrekte Berechnung des Alignments ist es zwingend erforderlich, dass die als Liste übergebenen Graphenobjekte a) eindeutige Bezeichner (Graphattribut: name) besitzen und diese b) mit den Bezeichnern der äußeren Blätter des Guide Tree übereinstimmen, welche die Inputgraphen im Alignment repräsentieren.

3.9.4 traverse_linear(graph_list)

Rückgabe: Objekt der Klasse Graph

Inputparameter	Typ	Default	Kurzbeschreibung
graph_list	Liste	-	Eine Liste von Graphobjekten.

Eine alternative und simplere Möglichkeit eines multiplen Alignments, bei dem die Graphen in der Reihenfolge ihres Inputs ins Alignment hinzugefügt werden. Als Rückgabeparameter wird der größte gefundene induzierte Subgraph übergeben. Eine weniger rechenaufwändige Variante gegenüber dem guide_tree, wenn die Reihenfolge des Alignments von vornherein feststeht, oder die eingegebenen Graphen sehr hohe Ähnlichkeit aufweisen.

3.9.5 score_similarity(graph1, graph2)

Rückgabe: Integer

Inputparameter	Typ	Default	Kurzbeschreibung
graph1	Graph	-	Ein zu vergleichendes Graphobjekt.
graph2	Graph	-	Ein zu vergleichendes Graphobjekt.

Eine simple Vergleichsfunktion für zwei Graphen, die bei der Konstruktion der Scoringmatrix (s. o.) eine Rolle spielt. Die Funktion kann auch extern aufgerufen werden. Sie berücksichtigt in dieser Version nur die Anzahl der vorliegenden Kanten und sollte für spezifischere Anwendungen konkret an den Bedarf angepasst werden.

3.10 Modul: core

Das Modul core stellt keine eigenen Funktionen zur Verfügung. Stattdessen handelt es sich um eine Programmbibliothek, die den unkomplizierten Aufruf aller in den vorherigen Abschnitten aufgelisteten Funktionen ermöglicht. Der Import des core Moduls genügt zum Ausführen aller relevanten Funktionen des musical_spoon packages. Für die Funktionsweise, die Input- und Ausgabeparameter der im Modul gelisteten Funktionen ist die Dokumentation auf den vorherigen Seiten heranzuziehen. Derzeit sind folgende Funktionen und Klassen über das Modul core aufrufbar:

Vertex
Edge
Graph
show_graph
show_two_graphs
show_graph_comparable
parse
parse_chem
reverse_parser
random_graph
random_chess_graph
random_triangular_graph
complete_graph
cut_edges
add_random_nodes
find_ankered_mcis
find_mcis
find_mcis_without_prompt
modular_product
Cordella
find_successors
find_predecessors
guide_tree
create_tree
traverse_tree
traverse_linear
score_similarity