NEURÁLIS HÁLÓ ALAPÚ ÖSSZETETT ESEMÉNYLÁNC FELTÁRÁSA

KOVÁCS LÁSZLÓ

A kutatási modul célja annak bizonyítása, hogy a mintafeltárás neurális háló alapú megközelítéssel is lehetséges legalább olyan hatékonysággal, mint amit a klasszikus, nem háló alapú módszerek biztosítanak. A kutatási modulban olyan neurálisháló-architektúra kerül kidolgozásra, amely

- *minimumelvárásként a művelettípusokat és azok időbeliségét,*
- *maximumelvárásként a műveletek költség- és sikerességstátuszát is*

figyelembe véve képes a folyamatmodell előállítására.
A modulban kidolgozandó modell jellegére minimumelvárás, hogy ne csak az esemény szekvenciákat, hanem az eseményfolyam-gráfmintákat is képes legyen előállítani.

1. A kutatás célja és lépései

Az első lépés a komplex gráfok elemzése volt, annak meghatározása, hogy milyen folyamatmodellezési szabványok vannak és azok mennyire támogatják a komplex eseményeket. Ennek során az alábbi lépéseket végeztem el:

- modellezési nyelvek áttekintése
- komplex események típusainak elemzése
- AND-típus vizsgálata
- XOR-típus vizsgálata
- ciklus vizsgálata
- eseményparaméterezések vizsgálata (ágens, objektum, ...)
- eseménygráf-feltáró

Az időszakban elvégzett kutatás az MLP-alapú eseményfeltáró modell elemzésére irányult, ahol a vizsgálat fő célja annak felderítése, hogy milyen mértékben alkalmas a modell valamely továbbfejlesztése az összetettebb gráfminták feltárására, illetve betanulására.

A tervezés során az alábbi vizsgálati pontokra tértem ki:

- eseménysor konverziója elemi előrejelzési lépésekre
- a háló tanítóhalmazának struktúrájának az előállítása
- tanító halmazok felépítése
- mintarendszer implementálása Pythonban
- tesztek elvégzése különböző paraméterbeállítások mellett
- rétegszám hatásának az elemzése

- inputvektor hosszhatásának az elemzése
- neuronszám hatásának az elemzése
- aktivációfüggvény hatásának az elemzése
- eredmények előzetes értékelése

A kutatás következő pontja a későbbi predikciós motorok tesztelési környezetének az előkészítése volt, amely egy tanítóhalmaz generáló rendszer előállítására szolgált. Ennek során pontosítani kellett a vizsgált modelleket, azok paramétereit, a generálás módját és a kimeneti eredmény formátumot.
A kimenetnek közvetlenül támogatnia kell a neurális hálók bemeneti formátumát.

A tervezés során az alábbi vizsgálati pontokra tértem ki:
- csomóponttípusok azonosítása
- eseményparaméterek azonosítása
- gráfreprezentáció meghatározása
- sémagráf-felépítés algoritmus kidolgozása
- sémafelépítő algoritmus kidolgozása
- sémavalidáció kifejlesztése
- random gráfpéldány-generáló algoritmus kidolgozása
- eredmények megjelenítési módjának megtervezése
- eredmények exportálása
- exportadatok utófeldolgozása

Az időszak további fő feladata a komplex eseménygráf feltárására szolgáló neurálisháló-modell megalkotása volt. Első lépésként két fő alternatíva az LSTM- és MLP-hatékonyság összevetését végeztem el. Ennek során az alábbi lépéseket végeztem el:
- elemi szekvenciák tanítóhalmazának előállítása
- LSTM-háló felépítése
- MLP-háló felépítése
- hatékonysági tesztek elvégzése
- paraméterezés hatásának tesztelése
- tesztek kiértékelése
- NN-típus kiválasztása

Az eredmények alapján az MLP került kiválasztásra az NN-modul elkészítéséhez.

A győztes NN-modell meghatározása után az NN-modell-architektúra részletes kidolgozása volt a fő cél. Ehhez előbb a gráf NN-modell logikáját kellett meghatározni, ennek során az NN-modulok specifikálását és azok kapcsolatának a kiépítését végeztem el.

A tervezés során az alábbi vizsgálati pontokra tértem ki:

- gráf felbontása feldolgozható modulokra
- a modulok előállítási, feltárási algoritmus kidolgozása
- az induló eseménysor modulokra bontási algoritmusának a kidolgozása
- a modulszintű elemzés megtervezése
- a modulok integrációs keretének a kidolgozása

Az időszak soron következő feladata a komplex eseménygráf feltárására szolgáló neurálisháló-modell véglegesítése volt. Kidolgozásra került a tanítási és a predikciós architektúra modell, elkészült a rendszer implementációja. Az elkészült rendszer ellenőrzésére teszteket hajtottam végre a generált tanítóhalmazokra építve.

Ennek során az alábbi lépéseket végeztem el:

- a réteges modell megvalósítása önálló MLP-modulokon keresztül
- az egyes MLP-modulok kapcsolatának kiépítése az adatok szintjén
- MLP-hálók implementálása
- hatékonysági tesztek elvégzése
- paraméterezés hatásának tesztelése
- tesztek kiértékelése
- keretrendszer értékelése

Az elkészített hálómodell tesztelése AND- és XOR-típusú szinkronizációs pontokat tartalmazó gráfmintákkal.

Az elért eredmények bemutatására egy külön publikációt tervezünk, ennek előkészítésére az alábbi lépéseket végeztem el:

- a szerzői csapat összeállítása, a feladatok kiosztása
- háttérirodalom szisztematikus feldolgozása
- a kidolgozott modell egyediségének kiemelése
- a tervezett cikk struktúrájának meghatározása

Az MLP/LSTM összevetési kísérleteket, a tanuló és a predikciós keretrendszer implementációját Keras-/Tensorflow-környezetben hajtottam végre. Majd a második fázisban csak elméleti munka folyt.

2. Kutatási eredmények összesítése

2.1. Elvégzett kísérletek bemutatása

Az irodalom elemzése alapján megállapítható, hogy az üzleti folyamatokat tevékenységekkel írják le, a tevékenységek sorrendjét pedig alkalmi függőségek modellezik. Ezt a relációt gráffal ábrázolják, amely csomópontokból és irányított élekből épül fel a csomópontok között, jelezve a folyamatok időrendi sorrendjét. Az él okozati és időbeli tulajdonságokat is leírhat, ill. meghatározza az adatok létrehozását és felhasználását a döntések modellezéséhez és annak módját. Az egyik legrégebbi folyamatmodellezési nyelv a Petri-háló, amely rendelkezik néhány magasabb szintű kiterjesztéssel, míg a legkifejezőbb üzleti folyamatmodell és jelölés a BPMN-nyelv [16]. Ezekhez a modellekhez szabványos XML-alapú adatcsereformátumokat fejlesztettek ki. A BPMN szabvány magában foglalja az XML Process Definition Language (XPDL), míg a Petri-hálók automatikusan feldolgozhatóak a Petri Net Markup Language (PNML) eszközzel. A Workflow Patterns Initiative szisztematikus elemzése eredményeként gráfminták gyűjteményét hozták létre. melyet a YAWL nyelv támogat. Ezek a minták az összes munkafolyamat-perspektívát lefedik. Például vannak vezérlési folyamatminták, adatminták vagy erőforrásminták. Vizsgálataink a control-flow perspektívára fókuszálnak, ahol 43 mintázat található, melyeket 8 osztályba sorolták.

A főbb vezérlési elemek:

- szekvencia
- elemi elágazás (XOR-ág)
- több választós elágazás (OR-ág)
- párhuzamos ág (AND-ág)

A folyamatfák blokkstrukturált modelleket képviselnek egy hierarchikus folyamatjelölést alkalmazva, ahol a (belső) csomópontok operátorok, mint pl. sorrend és választás, a levelek pedig tevékenységek. A levél csomópontjai egy process fa az eseménycsomópontoknak felel meg, míg a nem levél csomópontok operátorok az események végrehajtási sorrendjét leíró csomópontok. A modell biztosítja a következő operátor-csomópontokat: eseményszekvencia (→), párhuzamos végrehajtás (AND) (∧), nem kizárólagos választás (OR) (∨), kizárólagos választás (XOR) (×) és eseményhurok.

A fejlesztés során elsőként a gráf absztrakt modelljét kellett kidolgozni. Ennek eredményeképp a sémagráfot és a sémacsomópontokat az alábbi paraméterek jellemzik:

- eseménytípusok száma
- aktorok száma (több ágens, munkás dolgozhat egyidejűleg a rendszerben)
- megmunkálási idő alapparaméterei az egyes eseménytípusoknál
- bemenő objektumok az egyes eseménytípusoknál
- kimenő objektumok az egyes eseménytípusoknál
- eseményrákövetkezési reláció

A sémagráf felépítéséhez egy programkódot kell írni, melyben példányosítjuk a kívánt sémaelemeket.

A keretrendszer grafikusan megjeleníti a sémagráfot.

A mintaszám beállítása után a rendszer a sémára illeszkedő, random paraméterű eseménysorokat generál. A keletkező eseménysorlisták lesznek a neurális háló bemeneti adatsorai.

A mintarendszer Python-nyelven, Google Colab fejlesztési keretrendszerben készült.

A szekvencia előrejelzéséhez elkészítendő neurálishálózat-alternatívákat teszteltük öt változatban: két alapmodell és három új, saját változat. A két bázismodell a széles körben használt LSTM- és MLP-modellek. A javasolt módosítások az előtagsorozat hosszának a kiterjesztésére vonatkoznak, mely során nem szükséges az osztályozási hálózat bemeneti méretének kiterjesztése. A javasolt módszer egy egyedi előkészítő lépést alkalmaz a bemeneti vektor csökkentésére, mely lehet

- egyszerű szakszervezeti alapú csökkentés
- neurális hálózat alapú redukció

A modell a gráfot logikailag két szintre bontja: egy szinkronizációs, globális szint és egy ágens, aktor szint. A szinkronizálási, csatlakozási csomópont azt az esetet jelöli, amikor a több aktorszál folyamatai egymáshoz igazodnak. Ez lehet AND- és XOR-csomópont is. Ezek a vezérlő csomópontok automatikusan felfedezhetők, ha az eseménynapló tartalmaz egy objektum-/termékattribútumot is. Ez esetben a következő lépés megköveteli, hogy minden alkatrész rendelkezésre álljon, ami a bemeneten szükséges. A termékattribútum azonosítja a megmunkálás, cselekvés tárgyát. Ezzel a paraméter segítségével fedezhetjük fel a termék-/objektumszintű függőséget a különböző események között az eseménynaplóból.

Az aktoresemények mellett a betanítási folyamat kiterjesztett bemeneti eseménygráfja szinkronizációs vezérlő csomópontokat is tartalmaz, amelyek leírják a szomszédsági kapcsolatot az eseménysorozatok között. Feltételezzük, hogy minden A vezérlő csomópontnak van egy bemeneti eseménysorozata és egy kimeneti eseménykészlete is. A Petri-hálókhoz hasonlóan, a szinkronizációs vezérlő

csomópont csak akkor aktiválódik, amikor az összes bemeneti szekvencia befejeződött. Ha az átmenet elindul, minden kimeneti sorozatnál elindul a végrehajtás.

A vezérlő (AND- és XOR-) csomópontok bányászata a következőkön megfontolásokon alapul.

- Az egyes eseménycsomópontok feltárása, feldolgozása időbeli sorrendben, szekvenciában történik. A rendszer elsőként a szinkronizációs szekvenciát tárja fel.
- A vezérlő események szekvenciájánál a kimenő/bejövő objektumfüggőség által megszabott megszorításokra is tekintettel kell lenni.
- Eltérő aktorú események csatlakozásánál szinkronizációs csomópontra van szükség.
- A bejövő eseményszekvenciák alapján az első kimenő eredmény a szinkronizációs eseménysor feltárása.

A gráf sémaszintű modelljének a betanulása az alábbi lépéseken alapszik:

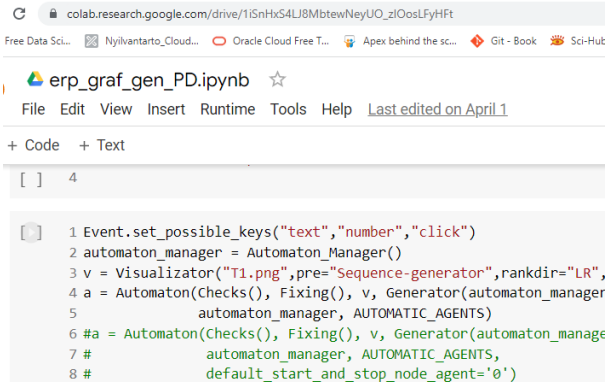
- az események láncának beolvasása
- a bemeneti/kimeneti termékfüggőség alapján a szinkronizációs pontok meghatározása
- a szinkronizációs (AND, XOR) eseményszekvenciák előállítása
- az eseményláncokból a (prefix szekvencia; következő esemény) tanítóhalmaz előállítása
- a tanítóhalmazok prefix részének redukciója a tanulóhálóknál
- szinkronizációs eseménylánc szekvencia modelljének meghatározása
- az aktorszintű szekvenciák, gráfrészletek kiemelése
- az eseményláncokból a (prefix szekvencia; következő esemény) tanítóhalmaz előállítása
- a tanítóhalmazok prefix részének redukciója a tanulóhálóknál
- aktorszintű eseménylánc-szekvencia modellek meghatározása

A gráf sémaszintű modell alapján történő predikció lépései:

- előzményrész inicializálása
- a szinkronizációs szekvenciamodell alapján predikció a soron következő szinkronizációs esemény meghatározására
- ezen lépések iterációja a végjelig
- a szinkronizációs eseménylánc előállítása
- az szinkronizációs események közötti aktorszintű eseményláncok típusainak meghatározása az eseményleíró adatbázis alapján
- az aktorszintű előzményrész inicializálása a háló már meglévő részei alapján

- az aktor-szekvenciamodell alapján predikció a soron következő aktoreseményre meghatározására
- ezen lépések iterációja a végjelig
- az aktoreseményláncok előállítása
- az elkészült láncok összefűzése egy közös eredménygráfba

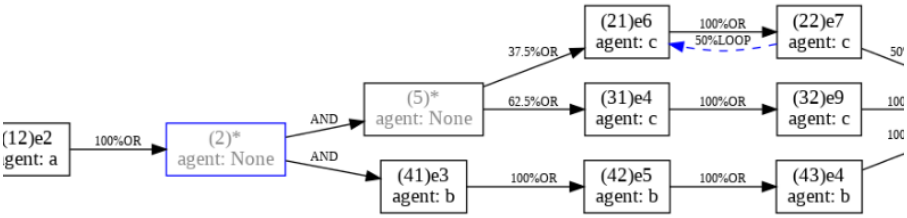
2.2. Eredményeket szemléltető képernyőképek



Séma generálási környezet

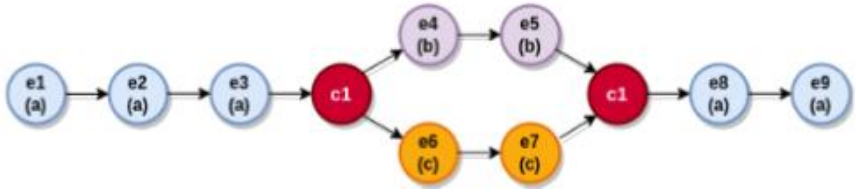
```
Event.set_possible_keys("text","number","click")
automaton_manager = Automaton_Manager()
v = Visualizator("T1.png",pre="Sequence-generator",rankdir="LR",debug =2)
a = Automaton(Checks(), Fixing(), v, Generator(automaton_manager),
              automaton_manager, AUTOMATIC_AGENTS)
#a = Automaton(Checks(), Fixing(), v, Generator(automaton_manager),
#              automaton_manager, AUTOMATIC_AGENTS,
#              default_start_and_stop_node_agent='0')
writer = FileWriter("file_Tuj.txt",pre="Sequence-generator")
#a.add_imaginary_node(new_node_id=1)
a.add_event(Event('e1',(1,2),{"text":"todo","click":True}), 0,
            new_node_id=11,
            agent_id='a')
a.add_event(Event('e2',(3,5),{"text":"todo","click":True}), 11,
            new_node_id=12, agent_id='a')
a.add_imaginary_node(12, new_node_id=2, new_node_type=AND)
a.add_imaginary_node(2, new_node_id=5, new_node_type=OR)
a.add_event(Event('e3',(1,4),{"text":"todo","click":True}), 2,
            new_node_id=41, agent_id='b')
a.add_event(Event('e5',(4,5),{"text":"todo","click":True}), 41,
            new_node_id=42, agent_id='b')
a.add_event(Event('e4',(2,3),{"text":"todo","click":True}), 42,
            new_node_id=43, agent_id='b')
a.add_event(Event('e6',(1,3),{"text":"todo","click":True}), 5,
            new_node_id=21,
            branch_possibilities=[0.3],
```

Sémagenerálás kódja



Előállított séma grafikus megjelenítése

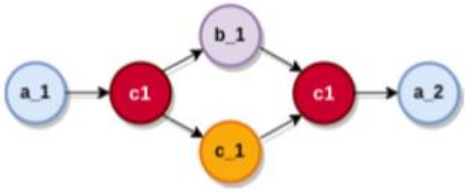
Kapcsolódó séma, amire a minták illeszkednek:



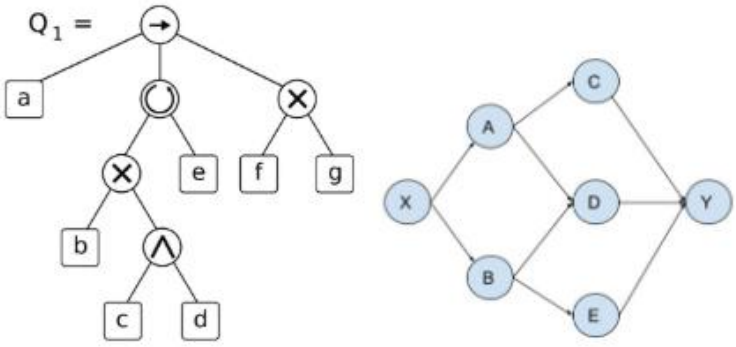
Generált mintaeseménysor:

```
EventT(1,1,'A',1)
EventT(2,2,'A',3)
EventT(3,3,'A',5)
EventT(4,4,'B',10)
EventT(5,5,'C',11)
EventT(6,6,'B',15)
EventT(7,7,'C',18)
EventT(8,8,'D',21)
EventT(9,9,'D',31)
```

Kapcsolódó szinkronizációs gráf:



2.3. Eredményeket szemléltető diagramok



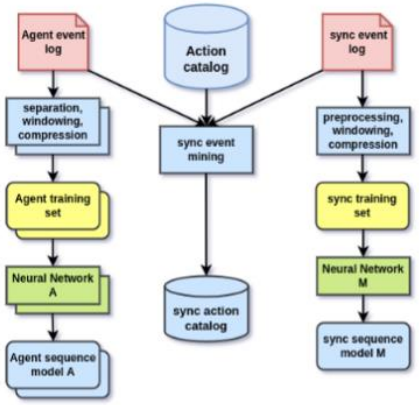
(a) Sample process tree [26] (b) Sample event graph

Table 1: Sample event log with artifact identification

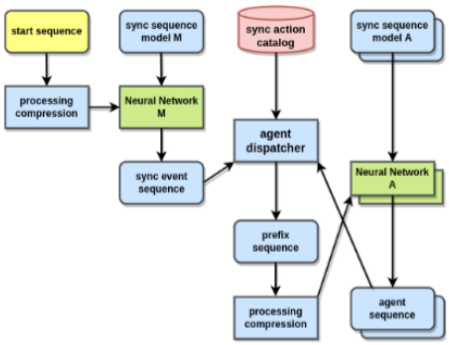
Trace	Actor	Action	Time	Input	Output
1	U1	A1	01.45	O1	O2
1	U1	A3	02.10	O2	O2
1	U2	A5	02.15	O2	O4,O5
1	U3	A8	02.18		O3
1	U3	A15	02.18	O3	O3
1	U4	A15	02.31	O2	O8
1	U4	A15	02.45	O8	O8

Az események szükséges leíró attribútumai

A tanuló NN-modul architektúrája



A predikciós NN-modul architektúrája



2.4. Kiértékelések eredményeinek bemutatása

Sequence generator rövid használati útmutató

A Sequence generator működtetéséhez elsőként a különböző osztályok példányosítását kell elvégezni. A fő kezelőosztály az *Automaton*, amiben a szerkezetet építhetjük, generálást hívhatunk meg. Az *Automaton* osztály konstruktora a többi osztály példányát várja függőségként.

*automaton.add_event(event,*parent_ids,new_node_type=,new_node_id=,branch_possibilities=[])*

Az első paraméter az esemény lesz, az ezt követő paraméterek vesszővel elválasztva a szülő csomópontok azonosítója. Az éppen létrehozott csomópont típusát, azonosítóját és a szülőktől az aktuális csomópontba vezető valószínűségek értékét a kulcsszavak kiírásával lehet csak megadni.

new_node_type esetében OR vagy AND típusú node-okról beszélhetünk (OR)
new_node_id esetében az adott node azonosítóját állítjuk be (automatikus)

branch_possibilities egy tömb, amely a sorrendben megadott szülőknél a valószínűség, hogy az éppen létrehozott csomópontba folytatódjon a szekvencia.

*automaton.add_imaginary_node(*parent_ids,new_node_type=,new_node_id=,branch_possibilities=[])*

Az add_eventhez hasonló, azonban az esemény helyére az IMAGINARY_NODE nevű konstanst állítja be. (add_eventtel megegyező)

automaton.add_event_between_nodes(parent_node_id,event,branch_possibilities,child_node_ids_list =, new_node_type=,new_node_id=)

ahol

parent_node_id: a szülő azonosítója
event: az adott node-ban tárolt esemény
branch_possibilities: tömb, amely a gyerekekhez kapcsolódás valószínűségét tárolja ugyanabban a sorrendben, mint ahogy a gyerekeket megadtuk
child_node_ids_list: a gyerekek azonosítója egy tömbben [3,10,32] (Az összes gyerek)
new_node_type: OR vagy AND típusú legyen-e a node (OR)
new_node_id: az új köztes node azonosítója (automatikus)

Elágazási valószínűség beállítása

automaton.reset_branch_possibilities_in_order(node_id,branch_possibilities)

ahol

a node_id azonosítójú node gyermekeinek valószínűségét módosítja;
node_id: node azonosító szám
branch_possibilities: tömb, amiben a legkisebb gyermekazonosítótól a legnagyobbig kell megadni a valószínűségeket. A tömbben megadott valószínűségek összege 1 kell hogy legyen, például: automaton.reset_branch_possibilities_in_order(2,[0.2,0.8])

A példa esetében a valószínűségek a következőképpen fognak kinézni.

Új él felvitele:

automaton.add_node_connection(start_node_id,end_node_id,branch_possibility,is_loop=)

A parancs két csomópontot köt össze az azonosítói alapján a megadott valószínűséggel. Az is_loop= adattag alapértéke false, szükséges, mert ha a kapcsolat referenciakört eredményez, végtelen ciklusba jutunk, ha ezt az adattagot igazra állítjuk, akkor a rendszer a végigiterálások alatt nem fogja vizsgálni ezt az ágat.

automaton.add_loopback(start_node_id,end_node_id,branch_possibility):

Az előző metódust hívja meg úgy, hogy itt egyértelműen loop típusú lesz a kapcsolat.

automaton.end_branch(last_node_id,end_possibility):

A last_node_id azonosítójú csomópontot a záró csomóponthoz köti egy *end_possibility* valószínűséggel.

automaton.visualize()

A gráf vizualizációját menti el a Visualize("file.png",pre=) osztály példányosításánál megadott helyre (alapesetben a generated_data/graphs/file.png).

automaton.finalize()

Lefuttatja az ellenőrzéseket és automatikusan kijavítja a következő hibákat. Ha egy node-nak egy gyereke van és a valószínűsége nem 1; Ha egy node-nak nincs gyereke, mégsincs hozzákötve a záró node-hoz.

automaton.generate(number_of_sequences):

A szerkezeten szimulációt futtat number_of_sequences darabszor, és lementi az utat egy tömbökből álló kódolt formára.

automaton.non_trivial_auto_fixing(node_id_list):

node_id_list: lista node id-kel, ahol a gyerekek valószínűségét magától beállítja (arányszámokként veszi a valószínűségeket).

A Sequence_Manager osztály főbb funkciói

create_agent_lists(generated)

Az automaton.generate() metódus által generált adatokat ágensek által végrehajtandó csomópontok tömbjére konvertálja.

create_sequence_objects(sequence_list)

A *create_agent_lists()* által kiadott listákból Sequencia listát csinál.

FileWriter writeSequence(seqence_list)

A create_sequence_objects által visszatért Sequencia listát a FileWriter(path_and_file,pre=) konstruktorában megadott path_and_file fileba írja ki.

Visualizator osztály

show_tree_colab()

Megjeleníti a gráf fileját a colab felületén.

Hibaüzenetek típusai

PossibilitySumException: kiírja, hogy mely node-ok esetében helytelen a valószínűségek összege.

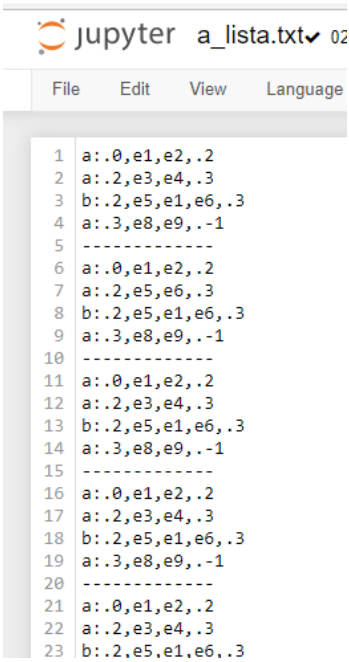
FailedStructureCheckException: általános strukturális hiba, a hibaüzenetben a hiba részletei is le vannak írva.

StructureNotClosedException: Amikor nincs egy node sem a záró node-hoz kötve. (Ha az utolsó node loop is.)

LoopNotAvailableException: Egy AND típusú node-ra szerettük volna a loop kezdetet rakni, ami így végtelen ciklushoz vezetne.

InvalidNodeIdException: A megadott felhasználó által beállított node_id már foglalt.

3. Technológiai folyamatok bemutatása



Generált eredménylista, a sémára illeszkedő random eseménysorokkal

Table 2: Accuracy comparison of the sequence prediction networks

Dataset	LSTM	MLP	NH-MLP	BE-MLP	BE_LSTM
$pdc_2016_1.xes$	63.5	63.4	63.4	63.9	64.1
$load_random$	58.2	57.5	56.6	58.7	58.0
$pdc_2016_9.xes$	82	82.5	81.2	81.8	82.6
$pdc_2017_5.xes$	62.4	62.8	63.7	64.8	64.2
$pdc_2019_2.xes$	64.6	65.2	63.9	66.27	65.6

Az LSTM-/MLP-teljesítmény-összehasonlító mérések eredményei (accuracy értékek). A mérések jól mutatják az MLP-változat (BELSTM) előnyeit:

- kisebb komplexitás
- gyorsabb végrehajtás
- ez egyik legjobb hatékonyság

A következő kódrészletek a három vizsgált hálótípus megvalósítását leíró kódot mutatják be.

```
                                # alap LSTM-háló
class lstm_A:

    def __init__(self,Lin, Lmid, Lout):
        self.model = Sequential()
        self.model.add(
            LSTM(Lmid,
                activation='relu',
                input_shape=(1, Lin))
        )
        self.model.add(Dense(Lout, activation="softmax"))
        self.model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
        #self.model.summary()

    def train (self, Xtr,Ytr,ep):

        self.model.fit(Xtr,Ytr,epochs=ep, verbose=0)
        self.model.fit(Xtr,Ytr,epochs=1, verbose=1)


    def predict (self, Xte, Yte):

        Yge = self.model.predict (Xte)

        db = 0
        for i in  range(Yge.shape[0]):
            yg = list(Yge[i,:])
            yt = list(Yte[i,:])
            if yg.index(max(yg))  == yt.index(max(yt)):
                db += 1
        print ("accuracy:",db/Yge.shape[0] )


                                # alap MLP háló
```

```
class mlp_A:

    def __init__(self,Lin, Lmid, Lout):
        self.model = Sequential()
        self.model.add(
            Dense(Lmid,
                activation='relu',
                input_shape=(Lin,))
        )
        self.model.add(Dense(Lout, activation="softmax"))
        self.model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
        #self.model.summary()

    def train (self, Xtr,Ytr,ep):

        self.model.fit(Xtr,Ytr,epochs=ep, verbose=0)
        self.model.fit(Xtr,Ytr,epochs=1, verbose=1)


    def predict (self, Xte, Yte):

        Yge = self.model.predict (Xte)

        db = 0
        for i in  range(Yge.shape[0]):
            yg = list(Yge[i,:])
            yt = list(Yte[i,:])
            if yg.index(max(yg))  == yt.index(max(yt)):
                db += 1
        print ("accuracy:",db/Yge.shape[0] )

                                    # saját nested MLP-háló
class nhmlp_A:

    def __init__ (self, X, Y, lra=0.01, P1=3,P2=6):

        L = len(X) # L : szintek száma
        inp_A = []  # bementi rétegek
        dense_A1 = []  # első rejtett rétegek
        dense_A2 = [] # második rejtett rétegek
        out_A = []  # kimeneti rétegek
```

```
    inps = []   # az összesítő háló bemenete

    for l in range(L):              # ciklus a rétegekre

      d1 = X[l].shape[1]            # bemeneti réteg
      d2 = Y.shape[1]
      inp_A.append(tf.keras.Input(shape=(d1,)))
                                    # első köztes réteg
      dense_A1.append( layers.Dense(P1*d2, activation="relu"))
      x1 = dense_A1[l](inp_A[l])
                                    # második köztes réteg
      dense_A2.append(layers.Dense(d2, activation="relu"))
                                    # kimenetek
      out_A.append( dense_A2[l](x1))

                                    # kimenetek összefűzése
    outs = []
    for l in range(L):
      outs.append(out_A[l])
                                    # bemenet az összesítő hálórészhez
    inp =  layers.concatenate(outs)
                                    # összesítő háló első köztes rétege
    d3 = Y.shape[1]
    dense_1 = layers.Dense(P2*d3, activation="relu")
                                    # összesítő háló második köztes rétege
    dense_2 = layers.Dense(d3, activation="softmax")
    x = dense_1(inp)
    out =  dense_2(x)               # kimenet
                                    # eredő háló összeállítása
    self.model = tf.keras.Model(inputs=inp_A, outputs=out)
    #self.model.summary()
    aopt = tf.keras.optimizers.Adam(lr=lra)
    self.model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),metrics=["accuracy"], opti-
mizer=aopt)

  def train (self, Xtr,Ytr,ep):

    self.model.fit(Xtr,Ytr,epochs=ep, verbose=1)
    self.model.fit(Xtr,Ytr,epochs=1, verbose=1)
```

```
def predict (self, Xte, Yte):

    Yge = self.model.predict (Xte)

    db = 0
    for i in  range(Yge.shape[0]):
        yg = list(Yge[i,:])
        yt = list(Yte[i,:])
        if yg.index(max(yg))  == yt.index(max(yt)):
            db += 1
    print ("accuracy:",db/Yge.shape[0] )
```

4. Összegzés

- Azonosítottama kezelendő gráfmodell szükséges vezérlőelemeit.
- A kapott eredmények alapján olyan neurális háló modell kell, ami támogatja az AND, XOR vezérlési elemeket.
- Elkészült a eseménygráfséma logikai modellje.
- Elkészült a sémaparaméterezés modellje.
- Elkészült a sémageneráló keretrendszer.
- Elkészült a sémára illő random eseményláncok generálását végző rendszer.
- Megterveztük és implementáltam az LSTM- és MLP-alapú hálókat.
- A gráfpredikciós motorhoz kiválasztásra került az MLP-alapú hálótípus.
- Elkészült az a háló séma feltárási modell, amely alkalmas az AND, XOR vezérlési elemek meghatározására is.
- Kétszintű feldolgozási modell kidolgozása a komplex eseménygráf feltárására.
- Algoritmus kidolgozása a szinkronizációs események feltárására.
- Algoritmus kidolgozása a prefix rész tömörítésére.
- Neurális háló alapú gráfpredikciós motor kidolgozása, amely alkalmas az AND, XOR vezérlési elemek meghatározására is.