Универзитет у Београду

Математички факултет

Решавање проблема комбинаторне оптимизације помоћу графовских неуронских мрежа

Мастер рад

Студент:
Владимир Јанковић

Ментор:
проф. др Зорица Станимировић

Чланови комисије

др Зорица Станимировић, редовни професор,
Универзитет у Београду, Математички факултет

др Зоран Станић, редовни професор,
Универзитет у Београду, Математички факултет

др Зорица Дражић, ванредни професор,
Универзитет у Београду, Математички факултет

Резиме

С обзиром на експанзију области вештачке интелигенције, у досадашњој литератури су предложене бројне хеуристике засноване на машинском учењу (енгл. *learn-heursitic*), које користе разне моделе машинског учења да би пронашли квалитетна решења и краће време извршавања алгоритма.

У овом раду, користе се модели машинског учења који представљају фузију графова и неуронских мрежа, под називом графовске неуронске мреже (енгл. *Graph Neural Networks*). Специфични модели који се користе су граф-конволуциона мрежа (енгл. *Graph Convolutional Network - GCN*), граф-атенциона мрежа (енгл. Graph Attention Network - GAT) и граф-агрегациона мрежа (енгл. *Graph Sample and Aggregate - GraphSAGE*).

Комбинаторни проблеми који су у овом раду решавани поменутим моделима машинског учења су проблем мултидимензионалног ранца (енгл. *Multidimensional Knapsack Problem-MdKP*) и проблем тежинског максималног независног скупа (енгл. *Weighted Maximum Independent Set - WMIS*). За тренирање три модела, користе се доступне инстанце из литературе, као и додатне генерисане инстанце.

Рад се састоји из пет поглавља, где се у првом поглављу уводе дефиниције и појмови графовских неуронских мрежа, у другом поглављу се описују комбинаторни проблеми MdKP и WMIS са њиховом математичком формулацијом и објашњава се како се могу комбинаторни проблеми представити у облику графа. У трећем поглављу се описује начин како су примењени модели машинског учења, заједно са инстанцама малих и средњих димензија. Четврто поглавље садржи приказ добијених резултата и дискусију о резултатима. У петом поглављу је дат закључак и наведени су могући правци даљег рада на модификацијама предложених модела.

Abstract

Due to the rapid expansion of artificial intelligence, numerous learning-heuristic methods have been proposed in the literature, which use various machine-learning models to find high-quality solutions while reducing algorithm running time.

In this thesis, the machine learning models that combine graphs and neural networks, known as Graph Neural Networks (GNNs), are used. The specific models considered are the Graph Convolutional Network (GCN), the Graph Attention Network (GAT), and Graph Sample and Aggregate (GraphSAGE).

The combinatorial problems solved by the aforementioned models are the Multidimensional Knapsack Problem (MdKP) and the Weighted Maximum Independent Set (WMIS). To train the three models, we use benchmark instances from the literature as well as additionally generated instances.

The thesis is organized into five sections. The first introduces the basic definitions and concepts of graph neural networks. The second section introduces the MdKP and WMIS, presents their mathematical formulations, and explains how combinatorial problems can be represented as graphs. The third section provides details on how the models are applied, and introduces small and medium-sized instances for training data. The fourth section presents the results and their discussion. The fifth section provides the conclusion and outlines work directions on possible modifications of the presented models.

Садржај

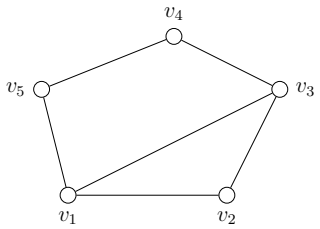
1 Графовске неуронске мреже

1.1 Графови - основни појмови и дефиниције

Неусмерени граф је уређен пар $G = (V, E)$, где је V коначан и непразан скуп, а E је подскуп скупа свих двочланих подскупова од V. Елементи скупа V су *чворови*, а елементи скупа E су *гране* графа G.

Усмерени граф је уређен пар $G = (\hat{V}, \hat{E})$, где је $\hat{V}$ коначан скуп, а $\hat{E}$ подскуп скупа $\hat{V} \times \hat{V}$. Елементе скупа $\hat{V}$ називамо *чворови*, а елементе $\hat{E}$ називамо *лукови*. Пошто нам у даљем разматрању неће бити непоходан усмерени граф, у даљем тексту ћемо графом сматрати неусмерени граф.

Графови се најчешће графички представљају у равни тако што се сваком чвору додели једна тачка, затим се тачке које одговарају чворовима спајају кривом ако постоји грана одређена тим чворовима.



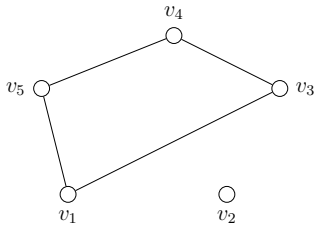
Слика 1: Илустрација графа G.

Ако за два чвора, x и y, постоји грана $xy \in E$, онда кажемо да су они *суседни*. Скуп свих суседа чвора v означавамо са $N(v) = \{u \in V | uv \in E\}$. *Степен* чвора v, који означавамо са $d(v)$, је број његових суседа. На пример, за граф на слици 1, скуп суседа за чвор v_4 је $N(v_4) = \{v_3, v_5\}$, док је степен чвора $d(v_4) = 2$. За граф G можемо дефинисати

$$\delta(G) = \min_{v \in V(G)} d(v), \qquad \Delta(G) = \max_{v \in V(G)} d(v)$$

као *минималан и максималан степен графа*, респективно.

Граф $G_1 = (V_1, E_1)$ је *подграф* графа $G = (V, E)$, ако је $V_1 \subset V$ и за све $xy \in E_1$ важи $xy \in E$.



Слика 2: Подграф графа G.

Шетња у графу G је низ чворова тако да између два узастопна чвора у низу постоји грана, односно $v_1, v_2, ..., v_n$ такав да за свако $i \in \{1, 2, ..., n-1\}$ важи $v_i v_{i+1} \in E$. За гране $v_i v_{i+1} \in E$, $i \in \{1, 2, ..., n-1\}$, кажемо да су садржане у шетњи.

Шетња је

- *затворена*, ако су v_1 и v_n исти, односно $v_1 = v_n$,
- *пут*, ако су сви чворови различити,
- *циклус*, ако је затворена и сви чворови су различити осим v_1 и v_n.

Дужина шетње, у ознаци $\mathrm{dist}(u, v)$, је број различитих чворова у шетњи. Пример шетње за граф на слици 1 је $v_1, v_3, v_4, v_5, v_1, v_2, v_3$, пример пута је v_1, v_2, v_3, v_4, v_5 и пример циклуса v_1, v_2, v_3, v_1.

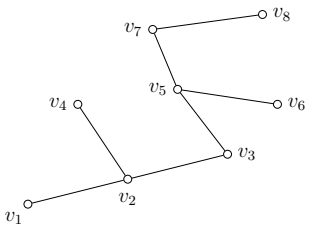
Околина реда k чвора v дата је као $N^k(v) = \{\, u \in V \mid \mathrm{dist}(u, v) = k \,\}$, а *затворена* k*-околина* као $\mathcal{N}_k(v) = \{\, u \in V \mid \mathrm{dist}(u, v) \leq k \,\} = \bigcup_{j=0}^{k} N^j(v)$, уз ознаке $N^0(v) = \{v\}$ и $N^1(v) = N(v)$, по договору.

Помоћу шетње можемо увести релацију

$$u \sim v \quad \Longleftrightarrow \quad \text{постоји шетња у графу } G \text{ од } u \text{ до } v. \tag{1}$$

Класе еквиваленције релације (1) су компоненте повезаности графа G. Уколико постоји тачно једна компонента, тада за граф кажемо да је *повезан*, у супротном је *неповезан*. Граф на слици 1 је повезан, док је његов подграф, који је приказан на слици 2 неповезан.

Стаблом (или дрветом), називамо граф $G = (V, E)$, који је повезан и не садржи циклус. Чворови који су степена 1 називају се листови. На слици 3, дат је пример стабла.

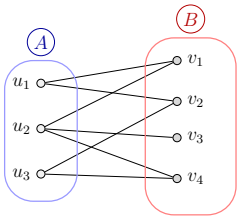


Слика 3: Повезан граф без циклуса - стабло.

Граф $G = (V, E)$ је *бипартитан* уколико постоје скупови A и B за које важи

- $A \cap B = \emptyset,\ A \cup B = V$,
- за сваку грану $xy \in E$, важи да је један крај у A, други у B.

Скупове A и B називамо *партиције* скупа G. Један пример бипартитног графа је дат на слици 4.



Слика 4: Бипартитан граф са пратицијама A и B.

Слика 5: Истгејт Шопинг Центр (енгл. Eastgate Shoping Centre, Zimbabve).[1]

1.2 Неуронске мреже

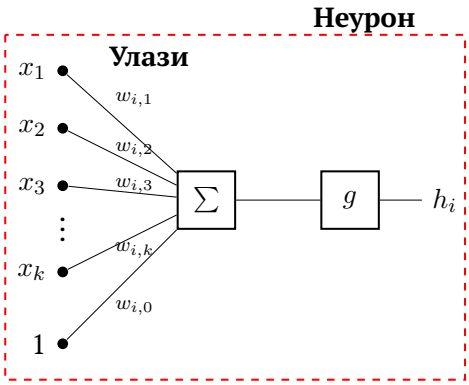
Најбитније ствари у науци и уметности криле су одговор у природи, попут тржног центра Истгејт Шопинг Центр у Зимбабвеу, који је приказан на слици 5, где је тражено од архитекте да дизајнира велику грађевину у којој би лети било пријатно и без коришћења климатизације, јер би увођење климатизације прекорачило буџет власника. Тада је архитекта Мајк Пирс схватио да може увидети инспирацију у односу на термите и структуру њихове колоније јер је закључио да је унутра знатно хладније него напољу током летњег периода, па те је широм центра скицирао отворе који топао ваздух износе напоље по узору на канале у колонији термита.

У оптимизацији је корисно проучавати процесе и појаве у природи како би се креирали биоинспирисани алгоритми. То су најчешће (мета)хеуристички алгоритми засновани на популацији јединки, односно скупу решења. На пример, *генетски алгоритам* (енгл. *Genetic Algorithm - GA*) [1] по узору на природну селекцију ствара нове генерације решења применом оператора селекције, укрштања и мутације, где се решења која учествују у стварању наредне генерације бирају на основу своје функције прилагођености. Популациона хеуристика *мрављa колонија* (енгл. *Ant Colony Optimization - ACO*) [2], опонаша тражење хране у колонији мрава помоћу појачавања или смањивања феромона, конструише решења уз помоћ феромонских трагова, где се са испаравањем спречава прерано конвергирање, док појачавањем феромонских трагова усмерава алгоритам ка бољим решењима. Хеуристика *колонија пчела* (енгл. *Bee Colony Optimization - BCO*) [3] је инспирисана понашањем пчела у кошници које су у потрази за поленом. Скупу решења одговара скуп вештачких пчела које размењују информације плесом и одлучују који је допустиви скуп за претраживање обећавајући, притом напуштајући делове допустивог скупа који нису фаворизовани за квалитетна решења. По узору на механизам транспорта материје које пролазе кроз ћелијске мембране где се притом ћелије мењају или деле, конструисан је алгоритам *мембранско израчунавање* (енгл. *Membrane computing-MC*) [4]. Решења се сместе у мембране које имају своја правила, где се решења применом правила мутирају или побољшавају, где се комуникацијом између мембрана, решења бирају процесом селекције. Креиран је алгоритам *ћелијски аутомати* (енгл. *Cellular automata-CA*) [5] инспирисан

[1] Преузето са: `https://earthbound.report/2020/05/15/building-of-the-week-eastgate-zimbabwe/`

процесом раста ткива, где се ћелије множе док има ресурса. Ствара се решетка тако да на сваком пољу постоје локални помераји и где се на сваком пољу налази део решења. Помераји се примењују док се не крше услови, попут ограничења или лошијег квалитета решења, стварајући систем који конвергира ка бољем решењу. Створен је алгоритам *ДНК израчунавање* (енгл. *DNA computing*) [6], инспирисан специфичним спаривањем база у ДНК молекулу. Велики скуп решења се кодира у ДНК низ и смешта се у епрувету, где се решења одбацују уз помоћу филтера изводљивости и квалитета.

Познато је да се мозак састоји од великог броја нервних ћелија, које су међусобно повезане. Информације које мозак задржава чувају се помоћу низа неурона. Да би се стекле нове информације, неопходно је да се нерви прерасподеле, где репетицијом знања чувају своју структуру. Средином двадесетог века, разматрани су биолошки неурони као јединице рачунања и постављено је питање: шта мреже таквих јединица могу да израчунају? МекКулох и Питс [7] формализују логички неурон и показују да мреже таквих јединица реализују логичка кола и секвенцијалне машине, постављајући темељ каснијим неуронским мрежама. Основну јединицу грађе таквог система чинио би неурон који се састоји од рачунарских операција. Сваки неурон рачуна линеарну комбинацију улазних података и притом користи активациону функцију. Структура неурона је дата на слици 6.



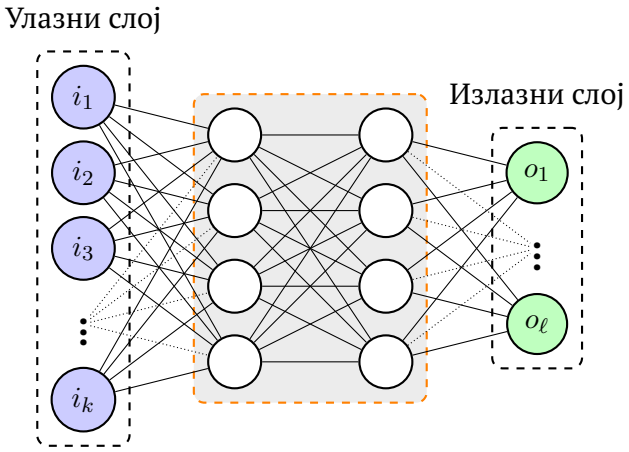
Слика 6: Структура појединачног неурона.

Неурони су смештени у слојеве, где први слој називамо *улазни слој*, док последњи слој називамо *излазни*. Сви преостали слојеви се називају *скривени слојеви*. Размена информација је таква да излазне вредности једног слоја постају улазне вредности следећег слоја, почевши од улазног до излазног слоја. На овај начин, модел ствара нове атрибуте да би разазнао појединости објеката и боље апроксимирао циљну функцију. Структура неуронске мреже је приказана на слици 7. Приметимо да два узастопна слоја чине бипартитан подграф.

Модел машинског учења $h = (h_0, h_1, \ldots, h_L)$ се формално дефинише:

$$\begin{aligned} h_0 &= x \in \mathbb{R}^{d_0}, \\ h_i &= g_i\Big(W_i\, h_{i-1} + w_{i0}\Big), \qquad i = 1, \ldots, L, \end{aligned} \tag{2}$$

где је $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $w_{i0} \in \mathbb{R}^{d_i}$, d_i димензија за i-ти слој, $x \in \mathbb{R}^{d_0}$ представља вектор улазних података и L број слојева. Овде W_i представља матрицу тежина у i-том слоју, док је w_{i0} вектор слободних чланова, који се назива праг. Активациона функција g_i је у општем случају произвољна нелинеарна функција.



Слика 7: Потпуно повезана неуронска мрежа.

Следећа теорема нам оправдава конструкцију модела.

Теорема 1 (Теорема о универзалној апроксимацији) *Нека је g ограничена и строго растућа непрекидна реална функција. Тада за сваку реалну функцију $f \in C([0,1]^n)$ и свако $\varepsilon > 0$, постоји број $m \in \mathbb{N}$, матрица $W \in \mathbb{R}^{m\times n}$, вектор $w_0 \in \mathbb{R}^m$ и вектор $v \in \mathbb{R}^m$, тако да за свако $x \in [0,1]^n$ важи*

$$\left|v^T g(Wx + w_0) - f(x)\right| < \varepsilon. \tag{3}$$

Теорема 1 нам говори да је скуп неуронских мрежа свуда густ у скупу непрекидних реалних функција на интервалу $[0,1]^n$. Доказ теореме 1 се може пронаћи у [8].

С обзиром на на претходну теорему, активациона функција $g(x)$ мора бити ограничена и строго растућа на реалном скупу. Дуго времена у литератури су се углавном користиле следеће две функције:

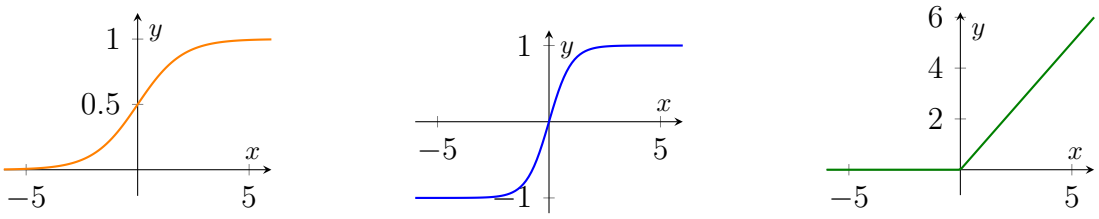
- Сигмоид:
$$\sigma(x) = \frac{1}{1+e^{-x}}$$
- Тангенс хиперболички:
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Можемо закључити да претходне функције испуњавају услове теореме 1. Због своје природе, односно због својства да се порастом вредности x брзо приближавају константи, представљале су проблем у оптимизацији модела. Стога је било неопходно дефинисати нову активациону функцију. У пракси се показало да се најбоља апроксимација функције достиже када се за активациону функцију користи исправљена линеарна функција која је дата формулом

$$ReLU(x) = \max(0, x).$$

На основу њених својства видимо да није ограничена и да није ни строго растућа функција, па не испуњава услове теореме 1.

Оправданост коришћења активационе функције $ReLU(x)$ огледа се у неколико чињеница. Прво, приметимо да је ова функција диференцијабилна скоро свуда, односно једино је недиференцијабилна у тачки $x = 0$. Додатно, користи се модификација функције која у околини нуле има вредности αx, за мале вредности параметра α. Извод функције је 0 или 1 на одговарајућим подинтервалима, што значајно смањује рачунску сложеност налажења извода композиције функција у којој она учествује. Графици поменутих активационих функција се налазе на слици 8.



$\sigma(x)$ $\tanh(x)$ $ReLU(x)$

Слика 8: Три најчешће коришћене активационе функције у неуронским мрежама.

Након постављања модела, неопходно је дефинисати одговарајући проблем оптимизације. Нека је дат скуп

$$\Phi = \{\, (x_j, y_j) \mid j = 1, .., N \,\},$$

где су са x_i дати улазни подаци, док са y_i вредности функције која се апроксимира. Уколико користимо ознаке из модела (2) и дефинишемо *функцију хипотезе* са $f_w(x) = h_L(x, w)$, где је $f_w(x) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$, тада задатак изгледа

$$\min_w \sum_{j=1}^{N} (f_w(x_j) - y_j)^2. \tag{4}$$

Овај проблем је у литератури познат као метода најмањих квадрата у еуклидској норми. Минимизација се врши одабиром скупа тежина модела неуронске мреже. Поред проблема оптимизације (4), неопходно је пазити на процес претприлагођавања. Наиме, може се десити да модел даје добре резултате на подацима на којим се тренирао, док на неким новим даје значајно лошије резултате.

Скуп могућих вредности параметара ω је високодимензионалан, пошто за сваки неурон у сваком слоју треба изабрати посебне тежине и праг. Уколико је скуп података Φ мали, скоро је сигурно да ће модел довести до претприлогођавања, што мора по сваку цену да се избегне. Због тога је неопходно да Φ буде довољно велико. На питање колики тај скуп мора бити није лако одговорити и зависи од функције која се апроксимира.

Поред скупа података, неопходно је наћи добар метод како израчунати неопходне тежине. Идеја је следећа: потребно је ићи корак унапред, (енгл. *forward pass*), где за дате улазне податке израчунава се излаз модела, након чега се одређује грешка излазног слоја у односу на циљне вредности. Пошто је циљ што боља апроксимација, кораком уназад (енгл. *backward pass*) од последњег слоја рачунају

се грешке у слојевима и градијенти функција за сваки неурон, где се напослетку адаптирају тежине у складу са величином грешке у претходном слоју до првог слоја. Овај процес представља једну итерацију или епоху учења у неуронским мрежама која је приказана на слици 9.

Нека је функција грешке последњег слоја и вредности функције у улазним подацима дефинисана као $\mathcal{L}(h_L, \hat{y}) = \frac{1}{2}||h_L - \hat{y}||_2^2$. Током корака уназад, рачунају се грешке на слојевима тако што се креће од последњег слоја ка првом и извршавају се следећи кораци:

1. Израчунати грешку на излазном слоју:

$$\delta_L = \frac{\partial \mathcal{L}}{\partial h_L} \odot g_L'(a_L),$$

   где је $a_L = W_L h_{L-1} + w_{L0}$, $g_L'(a_L) = (g'(a_{L,1}), \ldots, g'(a_{L,d_L}))$, g је активациона функција последњег слоја, а $\odot$ Адамаров производ.

2. Пропагирати грешку уназад кроз слојеве:

$$\delta_i = W_{i+1}^T \delta_{i+1} \odot g_i'(a_i), \quad i = L-1, \ldots, 1$$

   где је $a_i = W_i h_{i-1} + w_{i0}$, δ_i представља грешку на слоју i и $g_i'(a_i) = (g'(a_{i,1}), \ldots, g'(a_{i,d_i}))$, где је g активациона функција $i-$тог слоја.

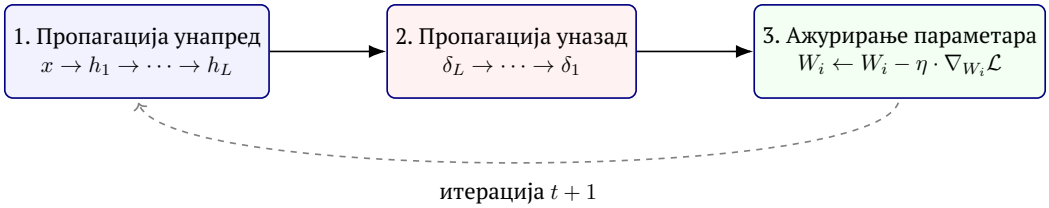
3. Израчунати градијенте:

$$\frac{\partial \mathcal{L}}{\partial W_i} = \delta_i h_{i-1}^T, \quad \frac{\partial \mathcal{L}}{\partial w_{i0}} = \delta_i$$

Кораком уназад се рачунају вредности градијента, али је потребно да се ажурирају параметри у сваком слоју

$$W_i \leftarrow W_i - \eta \cdot \frac{\partial \mathcal{L}}{\partial W_i}, \quad w_{i0} \leftarrow w_{i0} - \eta \cdot \frac{\partial \mathcal{L}}{\partial w_{i0}}, \quad i = 1, \ldots, L, \tag{5}$$

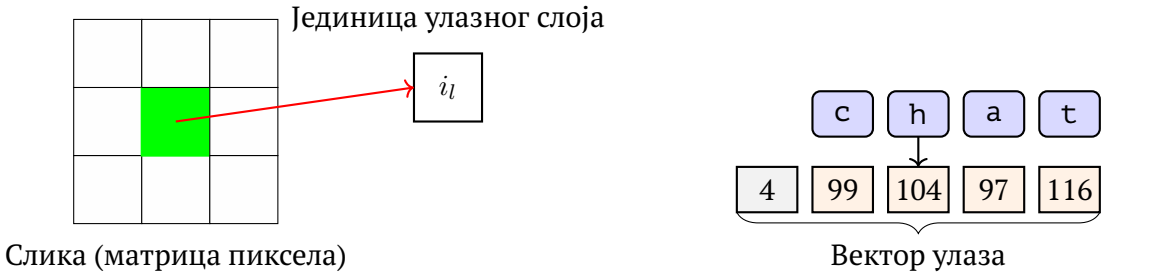
где је непоходно користити неку од метода градијентог спуста, од које зависи параметар η. Градијенте методе које се користе приликом ажурирања параметара су метода најбржег спуста, метода коњугованих градијената и Њутнова метода. Постоји класа стохастичких градијентних метода које процењују градијент ослањајући се на податке из претходних корака, али оне неће бити предмет нашег разматрања.



Слика 9: Циклус учења неуронске мреже: пропагација унапред, пропагација уназад и ажурирање параметара.

1.3 Графовске неуронске мреже

Посматрајмо проблем препознавања предмета на сликама. Пошто се свака слика састоји од пиксела, природно је да улазне јединице неуронске мреже буду вредности једног пиксела, те се за ову класу проблема користи *конвулуциона неуронска мрежа*. Уколико имамо речник и ако је задатак креирати чет-бота, речи можемо кодирати по дужини слова и да за свако слово придружимо вредности из ASCII табеле, па су речи претворене у векторе, где се у овом случају могу користити *рекурентне неуронске мреже*. Приказ улазних података за неуронске мреже је дат на слици 10.



(а) Један пиксел се прослеђује као улаз неуронске мреже

(б) Кодирање речи у ASCII вектор

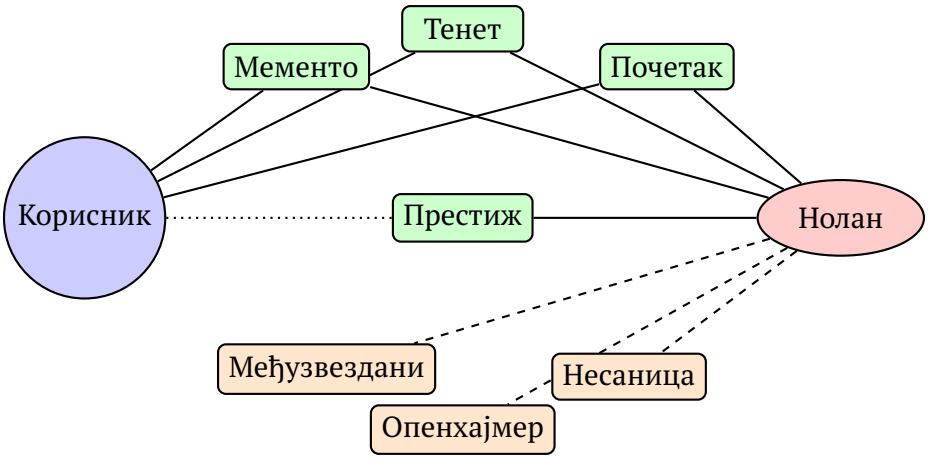
Слика 10: Различити улазни формати за улазе у неуронску мрежу.

Међутим, у многим реалним ситуацијама, подаци имају структуру графа, као што су:

- друштвене мреже, где су чворови корисници, а ивице представљају пријатељства;
- молекули, где чворови представљају атоме, а гране хемијске везе;
- мреже препорука, где су чворови корисници и производи, а ивице означавају интеракције;
- транспортне мреже, где чворови представљају станице или раскрснице, а гране путеве.

Фокусирајмо се на проблем мреже препоруке и претпоставићемо да је корисник приступио платформи за гледање филмова. Уколико корисник жели да гледа филм, али није сигуран који наслов, платформа на основу историје претраге и претходних гледаних филмова може да претпостави који су жанрови, редитељи и сценаристи фаворизовани код корисника. На пример, ако знамо да је корисник гледао филмове *Мементо*, *Тенет*, *Почетак* и да је у претрази имао интересовање за филм *Престиж*, оно што је слично за све ове филмове је да их је режирао *Кристофер Нолан* и природно је да се препоруче филмови попут *Међузвездани*, *Опенхајмер* и *Несаница*. Граф препоруке је дат на слици 11.

У оваквим случајевима, није довољно посматрати само атрибуте појединачних чворова, већ и структуру повезаности међу њима. Због тога је потребан модел који је у стању да комбинује и обрађује информације са графовских структура, где управо ту долазе до изражаја **графовске неуронске мреже (GNN)**.



Слика 11: Пример препоруке у графовском моделу: гледани, претражени и непознати филмови повезани са редитељем.

Графовске неуронске мреже омогућавају да се информација шири кроз структуру графа, тако да сваки чвор током учења агрегира информације од својих суседа. На тај начин, чворови добијају контекстуалне представе (енгл. *node embeddings*) које у себи садрже и локалну и глобалну структуру графа.

Формално, улаз у графовску неуронску мрежу чине:

- граф $G = (V, E)$, где је V скуп чворова, а E скуп ивица;
- матрица атрибута чворова $X \in \mathbb{R}^{|V| \times d}$, где сваки ред $x_i, i = 1, ..., |V|$ представља вектор атрибута чвора v_i,
- опционо, матрица атрибута грана $S \in \mathbb{R}^{|E| \times q}$, где сваки ред $s_j, j = 1, ..., |E|$ матрице S одговара једној ивици e_j и представља вектор њених атрибута дужине q. Другим речима, за сваку ивицу e_j познато је q атрибута.

Уколико претпоставимо да је структура графа дата матрицом повезаности $A \in \mathbb{R}^{|V| \times |V|}$, GNN модел има за циљ да научи функцију

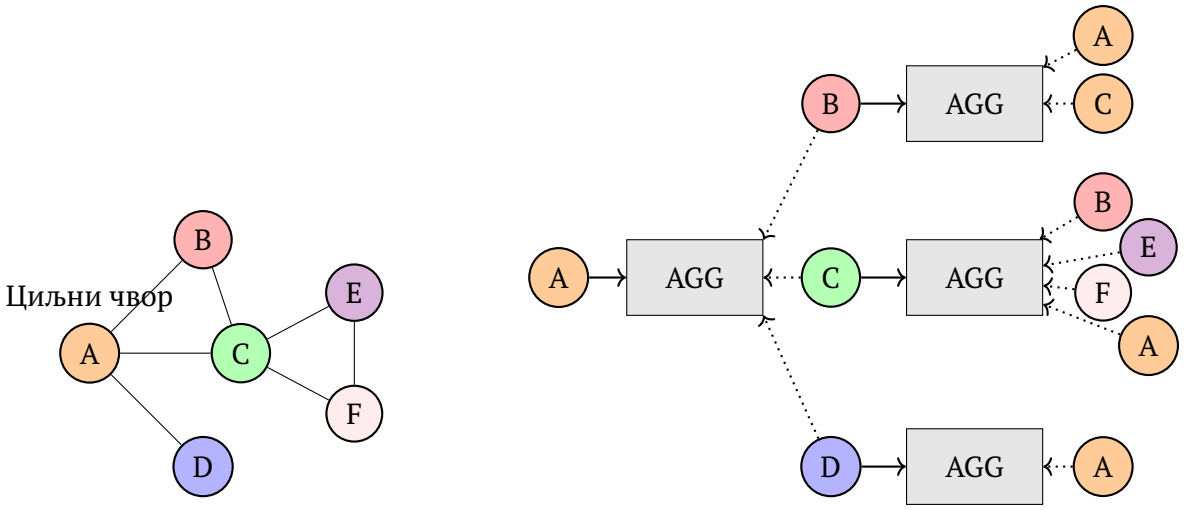
$$f : (A, X) \mapsto Z,$$

где је $Z \in \mathbb{R}^{|V| \times p}$ матрица нових репрезентација чворова (енгл. *node embeddings*) која се касније може користити за класификацију, кластеровање, препоруке и друге задатке.

Сваки слој GNN модела трансформише репрезентацију чвора тако што комбинује вредности вектора његовог атрибута са информацијама из његове околине. У току сваке итерације прослеђивања порука, слој $\mathbf{h}_u^{(k)}$ за чвор $u \in V$ ажурира се на основу агрегираних података од суседних чворова из скупа $\mathcal{N}_k(u)$. Дводубински приказ агрегације у GNN приказан је на слици 12. Формално, трансформација вектора атрибута чвора u из слоја $k-1$ у слој k може се записати на следећи начин:

$$\mathbf{h}_u^{(k)} = \text{Update}^{(k-1)}\left(\mathbf{h}_u^{(k-1)}, \text{Aggregate}^{(k-1)}\left(\left\{\mathbf{h}_v^{(k-1)} \mid v \in \mathcal{N}_k(u)\right\}\right)\right) \tag{6}$$

$$= \text{Update}^{(k-1)}\left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}_k(u)}^{(k)}\right), \quad k = 1, ..., L, \tag{7}$$



(а) Улазни граф *(б) Агрегационо стабло*

Слика 12: Дводубински приказ агрегације у GNN: (а) чвор A и његова околина у графу, (б) стабло агрегације са затвореном околином реда 2.

где је L број слојева. Функције Update$^{(k)}$ и Aggregate$^{(k)}$ могу бити произвољне диференцијабилне функције. У општем случају, ове функције се разликују од слоја до слоја, али могу бити исте. Израз $\mathbf{m}_{\mathcal{N}_k(u)}^{(k)}$ представља агрегирану „поруку“ из затворене околине реда k чвора u.

На почетку, врши се иницијализација као $h^{(0)} := X$, тј. матрица почетних атрибута чворова. У сваком следећем слоју, за $k = 1, ..., L$, чворови прикупљају и обрађују информације од својих суседа, користећи одговарајуће нелинеарне функције и припремајући се за наредни корак. Начин на који се агрегирају ове поруке представља кључну разлику између различитих типова графовских неуронских мрежа, што ће бити детаљније објашњено у наставку.

Ако за функцију Update на сваком слоју, узмемо неку од активационих функција које су обрађене у делу са неуронским мрежама и ако за функцију Aggregate заменимо са простим сабирањем, тада добијамо **основни облик графовских неуронских мрежа**

$$\mathbf{h}_u^{(k)} = g\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}_k(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right), \tag{8}$$

где су $W_{self}^{(k)}$ тежине чвора u у слоју $k = 1, ..., L$, g активациона функција ,док су $W_{neigh}^{(k)}$ тежине суседних чворова из затворене околине $\mathcal{N}_k(u)$ унапред дефинисаног реда k.

1.3.1 Графовске конвулуционе мреже

У основном облику графовске мреже (8) користили смо суму као обично агрегирање. Како се у пракси показало да тако долази до преурањене конвергенције и претприлагођавања, дат је нови предлог у виду архитектуре **графовске конвулуционе мреже** (енгл. *Graph Convolutional Network (GCN)*). Основна идеја је да се репрезентације чворова ажурирају путем нормализоване агрегације суседа, при чему се сваком чвору придружи петља, грана која садржи тачно један чвор.

Ажурирање се врши по следећој формули [9]:

$$\mathbf{h}_u^{(k)} = g\left(\mathbf{W}^{(k)} \sum_{v \in \mathcal{N}_k(u)} \frac{\mathbf{h}_v^{(k-1)}}{\sqrt{|\mathcal{N}_k(u)| \cdot |\mathcal{N}_k(v)|}}\right), \quad k = 1, ..., L. \tag{9}$$

Формула 9 се добија за следећи избор функција Aggregate и Update из општег GNN модела:

- Aggregate врши нормализовано сабирање суседа:

$$\mathbf{m}_{\mathcal{N}_k(u)}^{(k)} = \sum_{v \in \mathcal{N}_k(u)} \frac{\mathbf{h}_v^{(k-1)}}{\sqrt{|\mathcal{N}_k(u)| \cdot |\mathcal{N}_k(v)|}}, \quad k = 1, ..., L.$$

- Update је једноставна афина трансформација:

$$\mathbf{h}_u^{(k)} = g\left(\mathbf{W}^{(k)} \cdot \mathbf{m}_{\mathcal{N}_k(u)}^{(k)}\right), \quad k = 1, ..., L.$$

1.3.2 Графовске атенционе мреже

Графовске атенционе мреже (енгл. *Graph Attention Network (GAT)*) представља архитектуру графовске неуронске мреже која уводи механизам пажње у процес агрегације суседних чворова. За разлику од GCN-а који користи фиксне нормализоване тежине засноване на степенима чворова, GAT омогућава да се тежине везане за суседе уче током тренинга, што омогућава флексибилнију и прилагодљивију агрегацију информација.

Механизам пажње у GAT-у има следећи концепт: за сваки чвор u и његовог суседа $v \in \mathcal{N}(u)$ у слоју $k = 1, ..., L$, рачуна се коефицијент пажње $\alpha_{uv}^{(k)}$ који одређује колико информација из чвора v треба да утиче на чвор u током агрегације. Ови коефицијенти се рачунају по следећој формули [10]:

$$\alpha_{uv}^{(k)} = \frac{\exp\left(\text{ReLU}\left(\mathbf{a}^{(k)\top}\left[\mathbf{W}^{(k)}\mathbf{h}_u^{(k)} \parallel \mathbf{W}^{(k)}\mathbf{h}_v^{(k)}\right]\right)\right)}{\sum\limits_{j \in \mathcal{N}_k(u)} \exp\left(\text{ReLU}\left(\mathbf{a}^{(k)\top}\left[\mathbf{W}^{(k)}\mathbf{h}_u^{(k)} \parallel \mathbf{W}^{(k)}\mathbf{h}_j^{(k)}\right]\right)\right)} \tag{10}$$

где је $\parallel$ оператор конкатенације, $\mathbf{W}^{(k)}$ матрица тежина за трансформацију угнежђења, а $\mathbf{a}^{(k)}$ је вектор тежина који се користи за израчунавање пажње.

Ажурирање представљања чвора u у слоју $k+1$ врши се као линеарна комбинација трансформисаних векторских представљења његових суседа:

$$\mathbf{h}_u^{(k+1)} = g\left(\sum_{v \in \mathcal{N}_k(u)} \alpha_{uv}^{(k)}\, \mathbf{W}^{(k)}\mathbf{h}_v^{(k)}\right), \tag{11}$$

где је g активациона функција. У пракси се често користи *више глава пажње* (енгл. *multi-head attention*), где се извршава више паралелних механизама пажње.

1.3.3 Графовске агрегационе мреже

Графовске агрегационе мреже (енгл. *Graph Sample and Aggregate (GraphSAGE)*) је архитектура графовских неуронских мрежа која се фокусира на скалабилност и

ефикасност приликом обраде великих графова. За разлику од GCN и GAT модела који користе целокупно окружење чвора током агрегације, GraphSAGE узоркује на случајан начин фиксни број суседа, где се за чвор u и околину $\mathcal{N}_k(u)$, добија околина $\overline{\mathcal{N}}_k(u) \subset \mathcal{N}_k(u)$.

Основна идеја GraphSAGE-а је да се у сваком слоју $k = 1, ..., L$ за сваки чвор u:

1. Изврши узорковање $\overline{\mathcal{N}}_k(u)$ скупа суседа $v \in \mathcal{N}_k(u)$, тако да је $|\overline{\mathcal{N}}_k(u)| = \min\{s_k, \deg(u)\}$, где је s_k хиперпараметар за слој k.

2. Агрегира њихова представљања $\mathbf{h}_v^{(k)}$ користећи диференцијабилну функцију $\text{Aggregate}^{(k)}$.

3. Комбинује добијену агрегирану поруку чвора u.

Формално, ажурирање чвора u у слоју $k+1$ рачуна се по следећој формули [11]:

$$\mathbf{h}_u^{(k+1)} = g\left(\mathbf{W}^{(k)} \cdot \left[\mathbf{W}^{(k)}\mathbf{h}_u^{(k)} \,\|\, \text{Aggregate}^{(k)}\left(\{\mathbf{W}^{(k)}\mathbf{h}_v^{(k)} \mid v \in \overline{\mathcal{N}}_k(u)\}\right)\right]\right) \tag{12}$$

2 Комбинаторни проблеми оптимизације

2.1 Општи појмови и особине

Често се тражи најбоље решење за одређену ситуацију, на пример одређивање најкраћег пута између два града, прављење идеалног распореда решавања обавеза или прављење идеалне стратегије у играма попут шаха или гоа.

Овим питањима се бави грана математике под називом математичка оптимизација. Задатак математичке оптимизације је налажење минималне или максималне вредности циљне функције на допустивом скупу решења S (13). Надаље ћемо увек тражити максималну вредност, јер је $\min f(x) = -\max(-f(x))$.

$$\max_{x \in S} \ f(x) \tag{13}$$

Решење $x^* \in S$ математичког програма називамо *оптимамлно решење*, док вредност таквог решења $f(x^*)$ називамо *оптимална вредност или оптимум*. Допустиви скуп представља хиперповрш која је одређена са ограничењима

$$S \ = \ \Big\{ x \in \mathbb{R}^n \ \Big| \ g_j(x) \leq 0, \ j = 1, \ldots, m, \ h_i(x) = 0, \ i = 1, \ldots, r \Big\}.$$

Ако допустиви скуп S садржи коначно или пребројиво много елемената, тада кажемо да проблем припада класи комбинаторних проблема.

2.1.1 Линеарни програм

Уколико су ограничења допустивог скупа g_j, h_i и функција циља линеарне функције, тада се ради о проблему *линеарног програмирања* (енгл. *Linear programing - LP*). Скуп S у том случају називамо полиедар, а пример проблема оптимизације над полиедром као допустивим скупом је дат на слици 13.

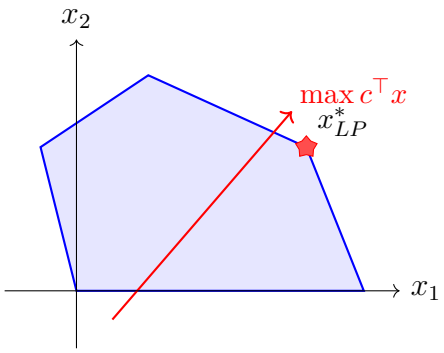
Најчешће се разматра канонска форма

$$\begin{aligned} &\max \ c^\top x \\ &\text{при услову} \ \ A\,x \ = \ b, \\ &\qquad\qquad x \ \geq \ 0, \end{aligned} \tag{14}$$

где је $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, а $c \in \mathbb{R}^n$ вектор коефицијената .

Кључне особине.

- *Временска сложеност:* LP се може тачно решити у полиномском времену, попут Карамаркове методе унутрашње тачке, прималне-дуалне методе унутрашње тачке.
- *Врх полиедра:* оптимум је увек у некој теменој тачки (тзв. *basic feasible solution*), што објашњава ефикасност симплекс алгоритма .



Слика 13: Линеарни програм: максимум циљне функције у темену x^*_{LP} полиедра.

2.1.2 Целобројни програм

Када бар једна од променљивих има *целобројну* вредност, добијамо *мешовито–целобројни* линеарни програм. Уколико су све променљиве целобројне, онда добијамо *целобројни* линеарни програм.

Математичка формулација за целобројни линеарни програм је ILP,

$$\begin{aligned} & \max && c^\top x \\ \text{при условима} \quad & && Ax \leq b, \\ & && x \in \mathbb{Z}^n, \end{aligned} \tag{ILP}$$

док је за мешовити целобројни линеарни програм математичка формулација је MILP.

$$\begin{aligned} & \max && c^\top x \\ \text{при условима} \quad & && Ax \leq b, \\ & && x_j \in \mathbb{Z} \qquad (j \in I), \\ & && x_j \geq 0 \qquad (j \notin I). \end{aligned} \tag{MILP}$$

Ограничења $x_j \in \mathbb{Z},\ (j \in I)$ моделирају **дискретне одлуке**:

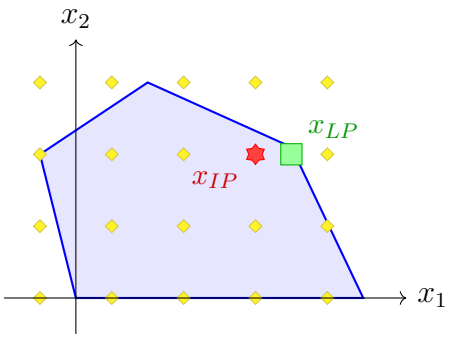
- да ли изградити нову фабрику $(x_j = 1)$ или не $(x_j = 0)$,
- колико камиона послати на руту $(k \in \mathbb{Z}^+_0)$,
- редослед обилазака градова (пермутацијске променљиве).

Најпознатији проблеми комбинаторне оптимизације попут проблема трговачког путника, успостављање p-центара, p-медијане и распоређивање послова на једној или више машина, управо су проблеми целобројног линеарног програмирања.

Уколико се занемари услов целобројности, онда се проблем своди на LP и тада се ради о релаксацији целобројног проблема , као што је на слици 14 где се види LP полиедар (плава зона) и решетка *допустивих* целобројних тачака,

- зелена тачка x_{LP} је оптимално решења релаксације, али није целобројна,
- црвена x_{IP} је најбоља решеткаста тачка и оптимално решење ILP-а.

Велики број проблема комбинаторне оптимизације који се могу представити помоћу ILP или MILP је NP-тежак, што значи да је тежак онолико колико и најтежи



Слика 14: LP релаксација (плави полиедар) и целобројни оптимум x_{IP}; LP оптимално решење x_{LP} није целобројан.

проблем у класи NP проблема. Другим речима, досад није утврђено да постоји алгоритам чија је временска сложеност полиномска тако да овај проблем реши.

У случају када егзактне методе (попут метода гранања и ограничавања, гранања и сечења) не дају решења услед временског или хардверског ограничења, проблеми комбинаторне оптимизације се решавају метахеуристичким и матхеуристичким приступима, као и хеуристикама заснованим на машинском учењу.

2.2 Проблем вишедимензионалног ранца

2.2.1 Опис проблема

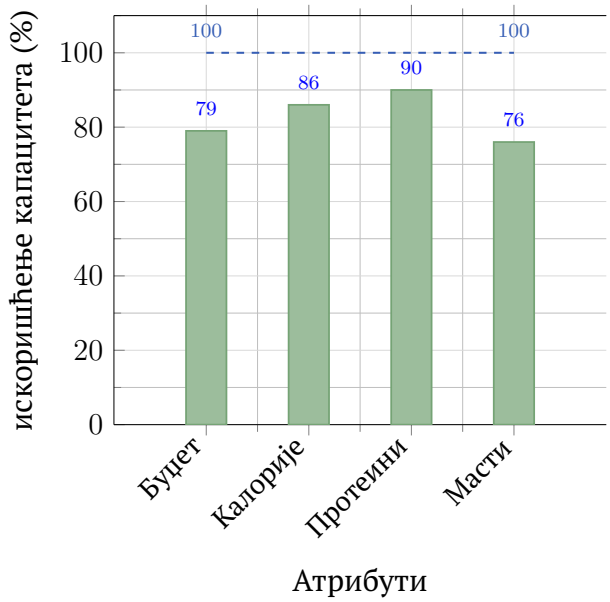
Проблем вишедимензионалног ранца (енгл. *Multidimensional Knapsack Problem - MdKP*), представља уопштење *проблема ранца* (енгл. *Knapsack Problem- KP*). Задатак KP је одабир i предмета од n, где тежинска сума предмета не сме да пређе капацитет b, тако да је сума профита предмета највећа [13].

Проблем KP има широку примену у пракси, попут одабира акција на берзи и избор трансакција за блок у криптовалутама. Илустративан пример је максимални одабир приоритетних хранљивих намирница у продавници како се не би прекорачио буџет. Уколико желимо да испланирамо дијету, у којој је неопходно пазити на калорије, протеине и масти, примећујемо да нам KP не даје могућност моделовања нашег захтева.

Наведени реални проблем се може адекватно описати коришћењем модела $MdKP$. За сваку намирницу придружујемо колико има калорија, протеина и масти. Тада је задатак да максимизујемо број i намирница, тако да се не прекорачи буџет, укупна количина протеина и масти, као и укупан број калорија. Овај проблем је дискретан, али је и NP тежак. На слици 15 дат је пример планирања дијете.

Проблем $MdKP$ је први пут представљен у раду [12], док је прва формална математичка формулација дата у раду [14]. У раду [15] аутори развијају оптималне алгоритме за решавање проблема мултидимензионалног ранца. Приступ из рада [15] се заснива на комбинацији методе гранања и ограничавања са ефикасним стратегијама релаксације, као и на методама за редукцију величине проблема. Ови поступци омогућавају значајно скраћење времена решавања у односу на претходне методе.

У раду [16] проблем је решаван помоћу метахеуристике генетског алгоритма (GA). Аутори користе бинарно кодирање решења у виду хромозома где свака генска позиција представља избор или изостављање предмета. Оператори укрштања



Слика 15: Потрошња ресурса у примеру $MdKP$-дијете: сва ограничења су испуњена, јер зелени стубићи остају испод плаве линије на 100 %.

и мутације комбинују се са хеуристиком која обезбеђује да нова решења остану допустива у оквиру свих ограничења. Оваква комбинација класичних GA оператора са знањем специфичним за проблем доприноси бољој разноврсности решења и бржем проналажењу решења високог квалитета у односу на оптималне методе из [15], посебно на већим инстанцама.

Аутори у раду [17] предлажу развој матхеуристике засноване на меметичком алгоритму. Меметички алгоритам комбинује еволутивни механизам генетских алгоритама са локалном претрагом ради интензивирања у области решења високог квалитета. У њиховој имплементацији се глобална претрага спроводи GA процедурама, док се на најбољим решењима примењују процедуре локалне претраге. На овај начин се добија баланс између експлорације и експлоатације, што доводи до знатно бољих резултата у односу на уобичајне GA методе.

У раду [24] је представљен FEPSS (енгл. *Finding and Exploring Promising Search Space*), матхеуристички оквир који спаја еволутивну популацију за издвајање добрих делимичних решења са кратким, циљано ограниченим тачним претрагама у потпростору који садржи "перспективна" решења. Поступак циклично ажурира популацију, а локалне процедуре покушавају са побољшањем "преспективних" решења како би се убрзала конвергенција ка решењима високог квалитета уз контролисану рачунску цену.

2.2.2 Математичка формулација MdKP

Дато је $n \in \mathbb{N}$ предмета при чему је за $i \in \{1, ..., n\}$, i-том предмету придружен профит p_i, као и ранац димензије $d \in \mathbb{N}$, где је за сваку димензију $k \in \{1, ..., d\}$ дат капацитет b_k. За сваки предмет i и димензију k, придружена је тежина a_{ki}. Циљ проблема вишедимензионалног ранца је проналажење подскупа предмета I, тако да је сума профита предмета из подскупа I максимална, притом поштујући ограничења капацитета за сваку димензију k.

Уводимо следеће променљиве:

- x_i - бинарна променљива која означава да ли је i-ти предмет одабран, где је тада $x_i = 1$, у супротном је $x_i = 0$.

Користећи претходну нотацију, проблем се може записати као целобројни линеарни програм [14]:

$$\max \quad \sum_{i=1}^{n} p_i \, x_i \tag{15}$$

$$\text{при условима:} \quad \sum_{i=1}^{n} a_{k_i} \, x_i \;\leq\; b_k, \quad \forall k = 1, \ldots, d \tag{16}$$

$$x_i \in \{0, 1\}, \qquad \forall i = 1, \ldots, n \tag{17}$$

Израз (15) означава да је циљ максимизација суме профита одабраних предмета. Ограничење (16) означава да укупна тежина предмета на нивоу k не прелази капацитет b_k. Ограничењем (17) обезбеђује се да су променљиве x_i бинарне.

2.3 Проблем максималног тежинског независног скупа

2.3.1 Опис проблема

Проблем максималног независног скупа са тежинама (енгл. *Weighted Maximum Independent Set - WMIS*) представља уопштењење *проблема максималног независног скупа* (енгл. *Maximum Independent Set - MIS*) [18], где је циљ одабрати максималан број неуседних чворова графа.

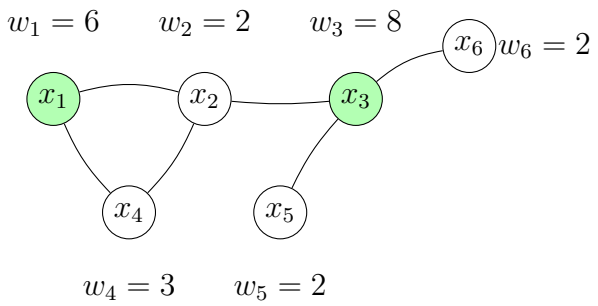
Посматрајамо телекомуникациону мрежу са потенцијалним локацијама. Циљ је да оператер покрије целу мрежу и да максимизује свој профит. Овај проблем се може моделовати са проблемом $WMIS$, где је задатак да бирамо несуседне чворове, тако да је сума тежина максимална. Комбинаторни проблем $WMIS$ је NP тежак. На слици 16 приказано је оптимално решење проблема на једном графу малих димензија.

Проблем $WMIS$ је формулисан у раду [19]. Савремени приступи обухватају више метахеуристика усмерених на брзу локалну претрагу и избегавање локалних оптимума. У раду [20] $WMIS$ решаван је убрзаном локалном претрагом за $WMIS$, са циљем ефикасне примене на великим графовима.

У раду [21] предложен је хибридни ILS са VND локалном претрагом. Дефинишу се две околине: (ω,1)-замена (убацивање једног чвора уз избацивање ω суседа) и тежинска (1,2)-замена, које се систематски испробавају до локалне оптималности, док адаптивна пертурбација контролише однос интензификације и диверзификације.

У [22, 23] развијена је METAMIS, напредна меметичка/GRASP метахеуристика која комбинује ефикасне локалне потезе са механизмом повезивања путања, спајањем различитих добрих решења ради проналажења нових и потезом наизменичног увећања који омогућава ефикаснији прелаз ка бољем независном скупу, уз употребу брзих структура података. Овај приступ постиже конкурентну или супериорну тачност на графовима великих димензија, са најбољим резултатима на великој већини инстанци.

У раду [25] је представљена метода гранања и редуковања (енгл. *branch and reduce* $(B\&R)$) за $WMIS$, која се састоји од следећих фаза. Најпре се граф смањује низом ефикасних редукција, затим се на редукованој инстанци спроводи претрага гранањем, а добијено решење се врати назад на изворни граф. Аутори показују да овај приступ, нарочито варијанта Red+HILS, даје вискоквалитетна решења на великом броју графова добијених из података стварног света.



Слика 16: Граф G са тежинама и оптималним независним скупом $\{x_1, x_3\}$.

2.3.2 Математичка формулација WMIS

Нека је дат граф $G = (V, E)$ и тежинска функција $w : V \to \mathbb{R}_0^+$ која сваком чвору v_i графа G додељује ненегативну вредност w_i која представља тежину чвора v_i. Циљ проблема максималног тежинског скупа је налажење подскупа $A \subset V$ тако да су чворови несуседни и да је сума тежина чворова одабраних у подскупу A максимална.

Уводимо следеће променљиве:

- x_i - бинарна променљива која означава да ли је чвор $i \in V$ изабран у подскуп A.

Користећи горњу нотацију, проблем може бити записан у виду целобројног линеарног програма [19]:

$$\max \quad \sum_{i=1}^{n} w_i \, x_i \tag{18}$$

$$\text{при условима:} \quad x_i + x_j \;\leq\; 1, \quad \forall i, j = 1, ..., n, \; i \neq j, \; v_i v_j \in E, \tag{19}$$

$$x_i \in \{0, 1\}, \qquad \forall i = 1, ..., n. \tag{20}$$

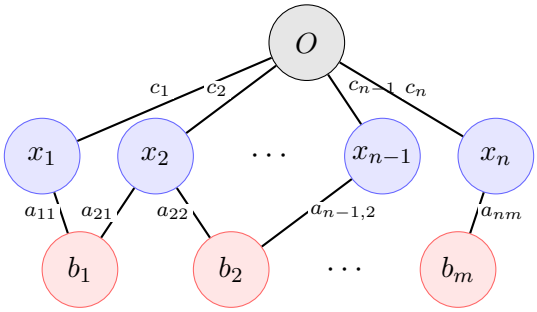
Циљ проблема је максимизација функције циља која представља суму тежина чворова, што је записано са (18). Ограничење (19) осигурава да су изабрани несуседни чворови. Са ограничењем (20), обезбеђује се да су променљиве бинарне.

2.4 Представљање комбинаторних проблема помоћу графова

Један од начина представљања комбинаторних проблема оптимизације, чија је математичка формулација целобројни линеарни програм, јесте у облику графа

који садржи бипартитан подграф. У таквом графу, чворови представљају различите компоненте као што су ограничења и променљиве, док посебан циљни чвор одговара линеарној функцији која се максимизује.

Чвор који одговара једној променљивој x_i, повежемо са циљним чвором o тако да је тежина гране c_i. Уколико променљива учествује у ограничењу, тада се на исти начин повеже чвор који одговара променљивој x_i са чвором који одговара ограничењу b_j тако да је тежина гране коефицјент a_{ij}. Самим тим, сада комбинаторни задатак постаје одабир чворова који одговарају бинарним променљивима $x_1, x_2, ..., x_n$, тако да сума тежина грана са циљним чвором максимална, под условом да чворови испуњавају ограничења. На слици 17 је дат уопштен пример представљања целобројног линеарног програма помоћу графа.

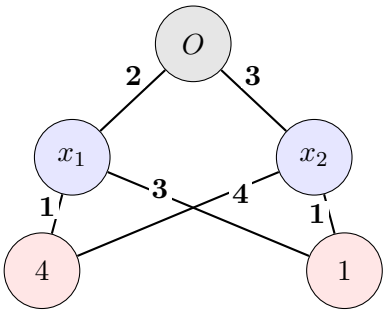


Слика 17: Графовско представљање целобројног програма: повезивање функције циља, променљивих и ограничења.

Једно такво представљање дато је следећим примером:

Целобројни линеарни програм:

$$\begin{aligned} \max \quad & 2x_1 + 3x_2 \\ \text{при ограничењу} \quad & x_1 + 4x_2 = 4 \\ & 3x_1 + x_2 = 1 \\ & x_1, x_2 \in \{0, 1\} \end{aligned}$$



Слика 18: Пример представљања целобројног програма у облику графа.

3 Примена GNN на комбинаторне проблеме

Програм за графовске неуронске мреже је кодиран у програмском језику *Python* на рачунару са процесором i5-1155G7 на FedoraOS оперативном систему. За решавање комбинаторних проблема у облику целобројног програма, коришћен је решавач *CPLEX 22.1.1*.

За потребе тренирања и тестирања, креиране су инстанце малих, средњих и великих димензија. За проблем мултидимензионалног ранца креиране су инстанце са називом `mdkp_n_m_tx`, где су параметри:

- n – број предмета,
- m – број димензија (ограничења),
- tx – фактор затегнутости, који представља горњу границу за коефицијент a_{ij} у односу на b_i.

Пример: `mdkp_12_3_40` $\rightarrow$ 12 предмета, 3 ограничења, фактор затегнутости 40.

```
12 3 %број предмета и ограничења
706 573 806 961 985 356 365 622 764 882 793 930 %профити
268 152 255 223 207  30  30 285 168  65  81 178 %димензија 1
 17  39  31  11  33   1  53  29  21  31   3  12 %димензија 2
 15 162 182 145 113  30 220  83 142 236 221  83 %димензија 3
738 136 643 % капацитети
```

Скуп параметра tx затегнутости за све инстанце је $\{20, 30, 40\}$. За све инстанце где је број предмета n, тада број ограничења m, зависи од n тако да је $m = \lceil ln \rceil$, где је l скуп параматара $\{0.1, 0.3, 0.5, 0.75, 1.0, 1.25, 1.5, 2.0\}$. Све вредности које се генеришу насумично, користе целобројну униформну расподелу на датом интервалу.

За мале инстанце узето је $n \in \{10, 12, \ldots, 100\}$ са кораком 2, док се капацитети b_i, профити p_j и тежине w_{ij} насумично узоркују из интервала $[100, 1000]$, $[50, 1000]$ и $\left[1, \frac{tx}{100} \cdot b_i\right]$, редом. Укупно инстанци малих димензија је 1081.

За средње инстанце узето је $n \in \{100, 102, \ldots, 200\}$ са кораком 2, док се капацитети b_i, профити p_j и тежине w_{ij} насумично узоркују из интервала $[500, 5000]$, $[100, 5000]$ и $\left[1, \frac{tx}{100} \cdot b_i\right]$, редом. Укупно инстанци средњих димензија је 1081.

За велике инстанце узето је $n \in \{300, 330, \ldots, 510\}$ са кораком 30, док се капацитети b_i, профити p_j и тежине w_{ij} насумично узоркују из интервала $[1000, 10000]$, $[500, 1000]$ и $\left[1, \frac{tx}{100} \cdot b_i\right]$, редом. Укупно инстанци великих димензија је 192.

Што се тиче проблема независног скупа са тежинама, креиране су инстанце са називом `wmis_n_p_`α, где су:

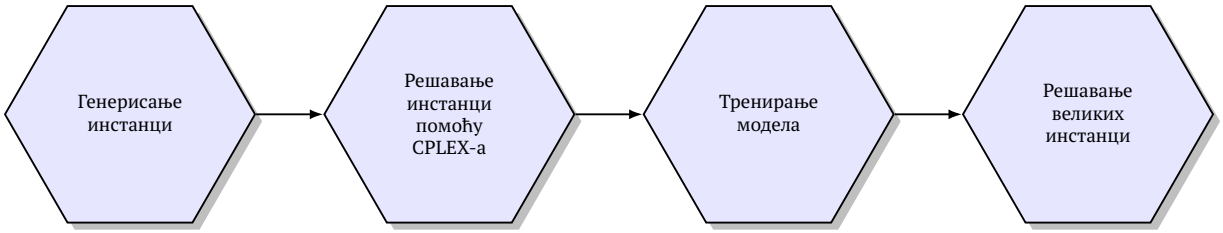
- n – број чворова у графу,
- p – вероватноћа да су два чвора повезана у графу,
- α – коефицијент максималне тежине између два чвора.

Пример: `wmis_10_0.3_1.5` $\rightarrow$ 10 чворова, 0.3 вероватноћа, 1.5 коефицјент.

```
10 %број чворова
0 0 0 1 1 1 1 0 1 1 %матрица повезаности
0 0 0 0 1 0 0 0 1 0
0 0 0 1 1 1 0 0 0 0
1 0 1 0 1 1 0 0 1 0
1 1 1 1 0 0 1 0 1 1
1 0 1 1 0 0 0 0 1 0
1 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 1 1
1 1 0 1 1 1 1 1 0 0
1 0 0 0 1 0 0 1 0 0
4 6 15 2 14 1 11 2 6 2 %тежине чворова
```

Вероватноћа да су два чвора повезана је у скупу $\{0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50\}$, док је максимална тежина у скупу, $t_{max} = \lceil \alpha x \rceil$, где је коефицијент α у скупу $\{1.0, 1.5, 2.0, 2.5\}$. Одабир тежине чвора користи униформну целобројну расподелу на интервалу $[1, t_{max}]$. За мале истанце је број чворова у интервалу $[10, 100]$ са кораком 5, за средње инстанце $[100, 200]$ са кораком 5 и за велике је $[300, 510]$ са кораком 30. Укупан број инстанци малих димензија је 608, број инстанци средњих димензија је 608, док је број инстанци великих димензија 256.

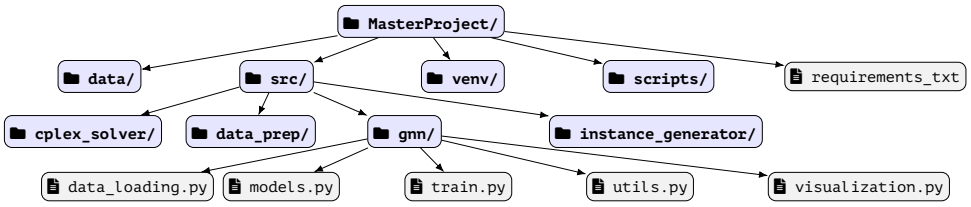
Генерисане инстанце су најпре решаване помоћу CPLEX решавача у временском интервалу од сат времена, као што је приказано у дијаграму тока пројекта 19. Резултати који су добијени, која су оптимална решења за све мање и средње инстанце, представљају се у облику графа и користе се за скуп података на којим ће модели бити креирани. За велике инстанце нису добијена допустива решења, па самим тим нису коришћена за тренирање. За сваки комбинаторни проблем, модели GCN, GAT и GraphSAGE се тренирају на свом скупу података, где се затим упоређују резултати и најбољи модели се користе за тражења најбољег решења за веће инстанце из литературе. Поред скупа података генерисаних инстанци, за тренирање модела су коришћене мање и средње инстанце са сајта OR-LIB [27] за проблем *MdKP*. Димензије инстанци из литературе за предмете су $\{6, 10, 20, 28, 39, 50, 100\}$, а за број ограничења $\{5, 10, 30\}$.



Слика 19: Ток извршавања пројекта.

Структура пројекта, која је дата на слици 20, је модуларна, где се кодови за графовске неуронске мреже налазе у директоријуму src/gnn. Инстанце се налазе у директоријуму data, које су креиране помоћу src/instance_generator. У директоријуму src/cplex_solver се налазе кодови за покретање решавача на инстанцама.

У наставку су дати кодови за датотеке models.py и train.py, док се преостали део налази на Гитхаб (енгл. GitHub) репозиторијуму [26]. Коришћен је пакет `torch` који садржи класе за моделе графовских неуронских мрежа. У датотеци



Слика 20: Структура пројекта.

`models.py` се налазе модели за GCN, GAT и GraphSAGE, где су за сва три модела, активациона функција је $ReLU(x)$ за све слојеве, осим за последњи слој, где је активациона функција $\sigma(x)$, како би излазне вредности биле класификоване у вредностима 0 или 1. Метода најбржег спуста је коришћена као градијентна метода за ажурирање параметара модела.

Кодови за моделе GCN, GraphSAGE и GAT су, редом, приказани у 1, 2 и 3.

Алгоритам 1 DeepGCN модел

```
class DeepGCNModel(nn.Module):
   def __init__(self, input_dim, hidden_dim, num_layers=4):
      super(DeepGCNModel, self).__init__()
      self.convs = nn.ModuleList()
      self.convs.append(GCNConv(input_dim, hidden_dim))
      for _ in range(num_layers - 1):
         self.convs.append(GCNConv(hidden_dim, hidden_dim))
      self.fc = nn.Linear(hidden_dim, 1)
      self.dropout = nn.Dropout(p=0.2)

   def forward(self, data):
      x, edge_index = data.x, data.edge_index
      for conv in self.convs:
         x = F.relu(conv(x, edge_index))
         x = self.dropout(x)
      x = self.fc(x)
      return torch.sigmoid(x).squeeze()
```

Алгоритам 2 DeepGraphSAGE модел

```
class DeepGraphSAGEModel(nn.Module):
   def __init__(self, input_dim, hidden_dim, num_layers=4):
      super(DeepGraphSAGEModel, self).__init__()
      self.convs = nn.ModuleList()
      self.convs.append(SAGEConv(input_dim, hidden_dim))
      for _ in range(num_layers - 1):
         self.convs.append(SAGEConv(hidden_dim, hidden_dim))
      self.fc = nn.Linear(hidden_dim, 1)
      self.dropout = nn.Dropout(p=0.2)

   def forward(self, data):
      x, edge_index = data.x, data.edge_index
      for conv in self.convs:
         x = F.relu(conv(x, edge_index))
         x = self.dropout(x)
      x = self.fc(x)
      return torch.sigmoid(x).squeeze()
```

Алгоритам 3 DeepGAT модел

```
class DeepGATModel(nn.Module):
   def __init__(self, input_dim, hidden_dim, num_layers=4, heads=4):
      super(DeepGATModel, self).__init__()
      self.convs = nn.ModuleList()
      self.convs.append(GATConv(input_dim, hidden_dim, heads=heads, concat=True))
      for _ in range(num_layers - 2):
         self.convs.append(GATConv(hidden_dim * heads, hidden_dim, heads=heads, concat=True))
      self.convs.append(GATConv(hidden_dim * heads, hidden_dim, heads=1, concat=False))
      self.fc = nn.Linear(hidden_dim, 1)
      self.dropout = nn.Dropout(p=0.2)

   def forward(self, data):
      x, edge_index = data.x, data.edge_index
      for conv in self.convs:
         x = F.relu(conv(x, edge_index))
         x = self.dropout(x)
      x = self.fc(x)
      return torch.sigmoid(x).squeeze()
```

У фајлу `train.py`, модел се тренира тако да се у свакој епохи рачуна корак унапред и корак уназад. Повратна вредност функције је листа грешака кроз епохе. Код алгоритма је дат у 4. Решења представљамо вектором дужине n чије су вредности из скупа $\{0,1\}$. Грешку по чвору $i=1,\dots,n$ дефинишемо $\ell(\hat{y}_i,y_i)=(\hat{y}_i-y_i)^2$, где је вектор $\hat{y}\in[0,1]^n$ добијен предвиђањем модела, а вектор је y стварно решење. Следећи пример показује рачунање просека грешке по свим чворовима. Нека је оптимално решење скуп $\{1,3,5\}$, односно $y=[1,0,1,0,1,0]$, а $n=6$.

- Ако модел врати $\{2,4,6\}$, онда је $\hat{y}=[0,1,0,1,0,1]$ и

$$\frac{\ell(\hat{y}_i,y_i)}{6}=\frac{1}{6}\sum_{i=1}^{6}(\hat{y}_i-y_i)^2=\frac{6}{6}=1.0.$$

- Ако модел врати $\{3,4,6\}$, онда је $\hat{y}=[0,0,1,1,0,1]$ и

$$\frac{\ell(\hat{y}_i,y_i)}{6}=\frac{1}{6}\sum_{i=1}^{6}(\hat{y}_i-y_i)^2=\frac{4}{6}\approx 0.667,$$

  што је боље јер се барем чвор 3 поклопио.

У процесу тренирања током једне епохе, скуп података се изнова групише у неколико група узорака са ознаком batch$^{(b)}$, где је $b=1,\dots,B$, B број група у скупу података, док је подскуп чворова који одговара том узорку V_b. Грешка по једној групи рачуна се као

$$\mathcal{L}_{\text{batch}}^{(b)}=\frac{1}{|V_b|}\sum_{i\in V_b}\ell(\hat{y}_i,y_i).$$

Нека је епоха састављена од група $b=1,\dots,B$ са бројем чворова $|V_b|$. Тада је губитак епохе за цео скуп података:

$$L_{\text{epoch}}=\frac{\sum_{b=1}^{B}\mathcal{L}_{\text{batch}}^{(b)}\,|V_b|}{\sum_{b=1}^{B}|V_b|}\,.$$

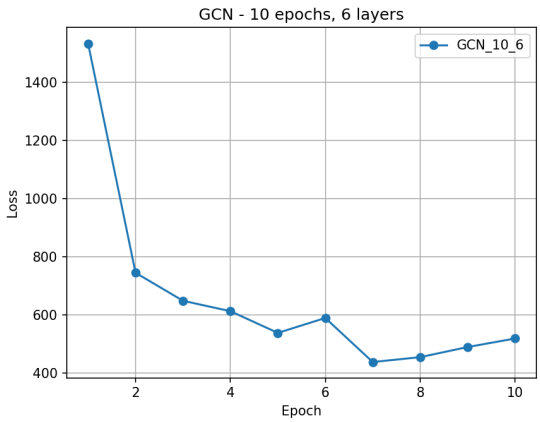
Грешка се рачуна по просеку чворова како би оцена грешке била непристрана у зависности од величине графа.

Алгоритам 4 Поступак обуке модела (`train_model`)

```python
#train.py
def train_model(model, train_loader, optimizer, device, epochs=50, lambda_output=0.1,
    lambda_capacity=0.1):
   """
   A simple training loop across epochs, returning a list of per-epoch losses.
   """
   model.to(device)
   epoch_losses = []

   for epoch in range(1, epochs + 1):
      model.train()
      total_loss = 0.0
      total_nodes = 0

      for data in train_loader:
         data = data.to(device)
         optimizer.zero_grad()

         out = model(data)
         loss = custom_loss_function(out, data.y, data,
                              lambda_output=lambda_output,
                              lambda_capacity=lambda_capacity)
         loss.backward()
         torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=2.0)
         optimizer.step()

         num_nodes = data.num_nodes
         total_loss += loss.item() * num_nodes
         total_nodes += num_nodes

      if total_nodes > 0:
         avg_loss = total_loss / total_nodes
      else:
         avg_loss = 0.0

      epoch_losses.append(avg_loss)
      print(f'Epoch {epoch}, Loss: {avg_loss:.4f}')

   return epoch_losses
```

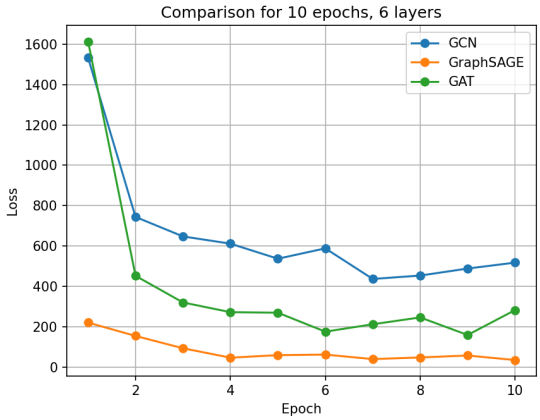
4 Резултати и дискусија

За оцењивање квалитета модела коришћени су графици који представљају зависност грешке у односу на број епоха. Један график који описује квалитет модела GCN за проблем MdKP са 6 слоја и са 10 епоха је дат на слици 21.



Слика 21: Резултати модела GCN са шест слојева и десет епоха.

Како би се испитало који модел има најбоље перформансе, три модела се пореде за исти број слојева и исти број епоха. Поређење је приказано на слици 22.

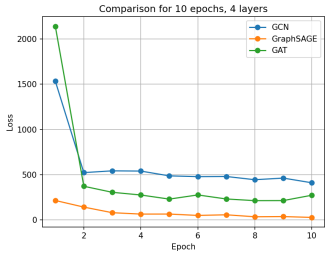


Слика 22: Поређење три модела са шест слојева и десет епоха.

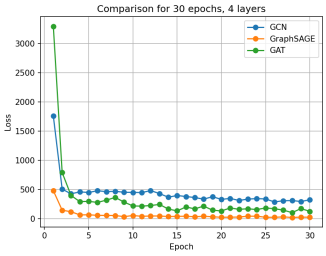
Поређења су груписане на основу истог броја слојева, при чему су на свакој слици приказани графици за различити број епоха, а исти број слојева. Број слојева је у скупу {4,5,6,7}, док је број епоха у скупу {10,30,50,70,90,100}.

4.1 Резултати за MdKP

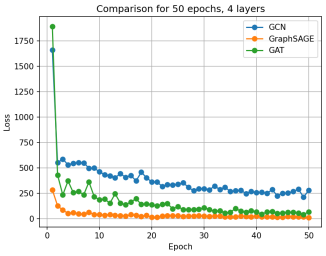
За моделе са четири слоја који се тренирају на инстанцама за проблем MdKP, дата су поређења на слици 23. Можемо закључити да је за ову групу модела, најбоље перформансе имао модел GraphSAGE.



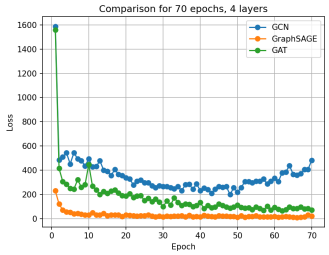
(а) 10 епоха



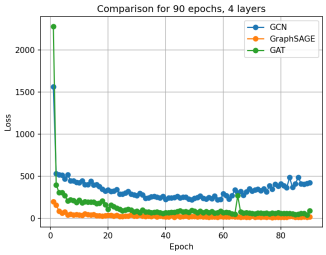
(б) 30 епоха



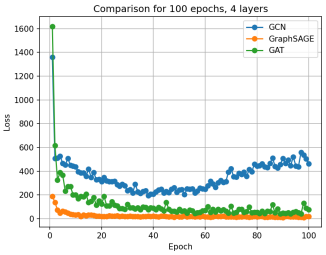
(в) 50 епоха



(г) 70 епоха



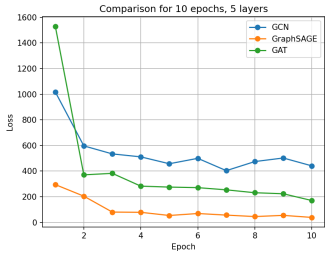
(д) 90 епоха



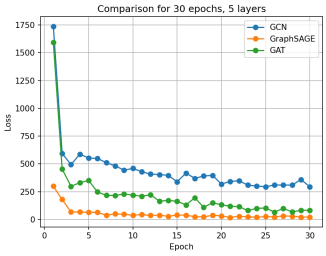
(ђ) 100 епоха

Слика 23: Поређење резултата за 4 слоја.

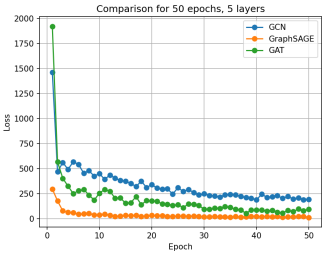
У случају са пет слојева, чија поређења можемо видети на слици 24 , можемо закључити да се модел GraphSAGE најбоље показао.



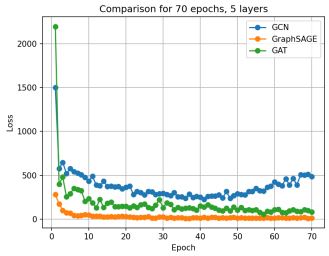
(а) 10 епоха



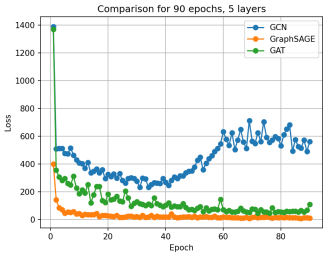
(б) 30 епоха



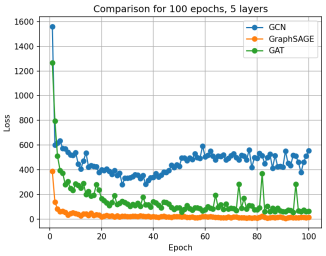
(в) 50 епоха



(г) 70 епоха



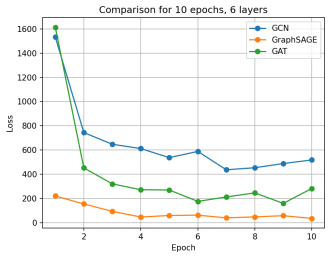
(д) 90 епоха



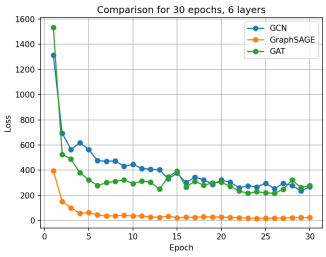
(ђ) 100 епоха

Слика 24: Поређење резултата за 5 слојева.

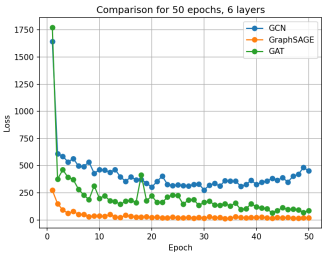
На основу поређења резултата са слике 25, можемо закључити да је модел GraphSAGE дао најбоље резултате.



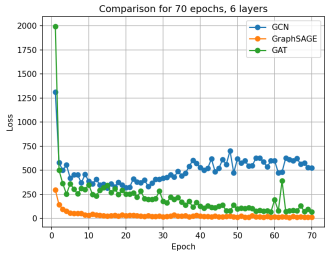
(а) 10 епоха



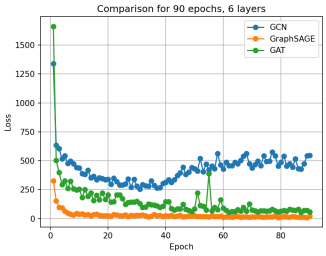
(б) 30 епоха



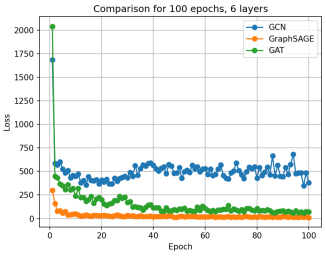
(в) 50 епоха



(г) 70 епоха



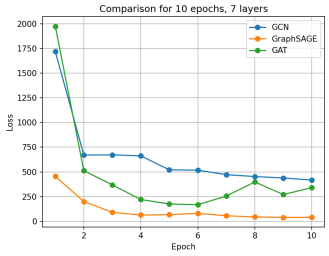
(д) 90 епоха



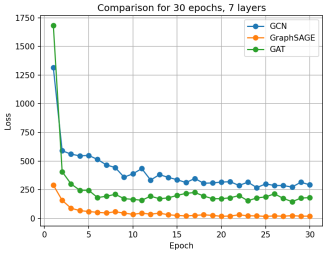
(ђ) 100 епоха

Слика 25: Поређење резултата за 6 слојева.

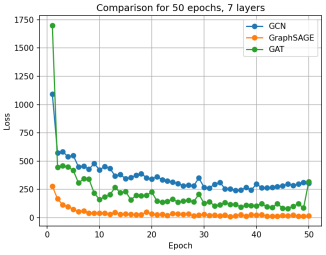
У случају седам слојева, можемо закључити да је модел GraphSAGE најбољи за сваки број епоха, што можемо закључити на основу поређења са слике 26.



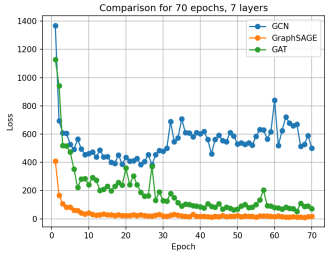
(а) 10 епоха



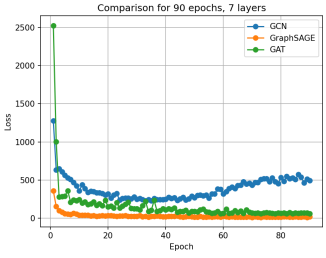
(б) 30 епоха



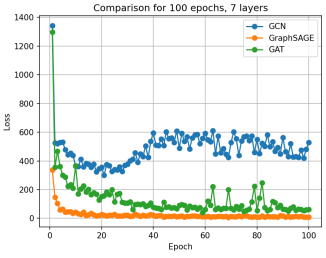
(в) 50 епоха



(г) 70 епоха



(д) 90 епоха



(ђ) 100 епоха

Слика 26: Поређење резултата за 7 слојева.

На основу претходних поређења може се закључити да модел GraphSAGE постиже најбоље резултате. Да би се идентификовао најбољи модел у односу на број слојева и број епоха, из сваке групе формиране према броју слојева издвојени су најперспективнији модели и тестирани на већим инстанцама sa OR-LIB [27]. Представници тих група су GraphSAGE-4-30, GraphSAGE-5-90, GraphSAGE-6-90, GraphSAGE-7-100.

Резултати добијени на већим инстанцама су упоређени са резултатима из рада [24] који су приказани у табели 1, чије су димензије 500 предмета и 30 ограничења. Најбољи резултати модела су приказани подвученом цртом, док су резултати који се поклапају са најбољим познатим потамљени.

Табела 1: Упоређивање резулта GraphSAGE модела и матхеуристике FEPSS.

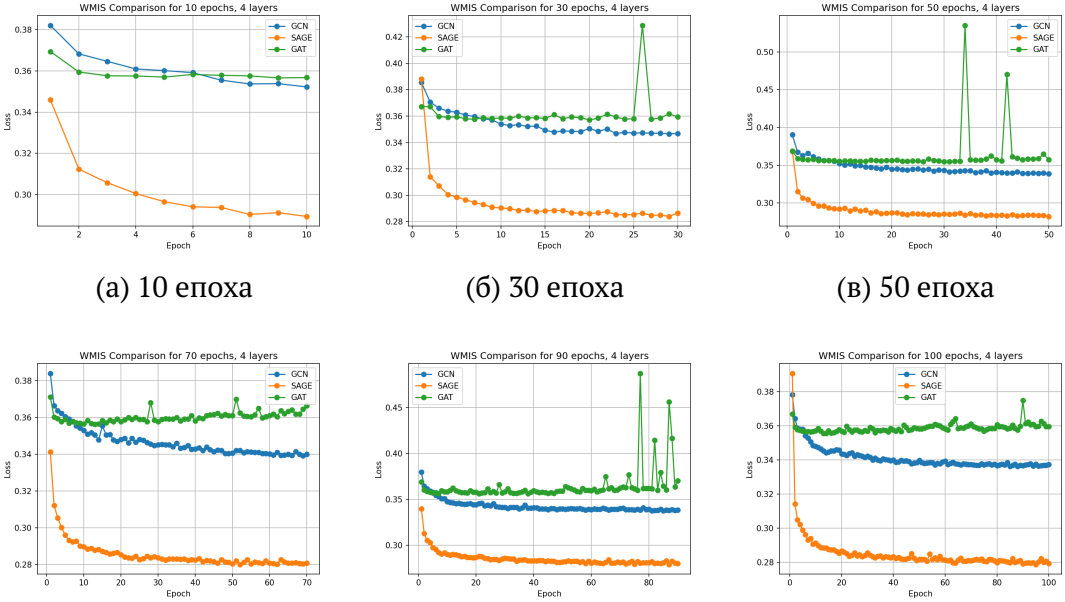
Инстанца	FEPSS f_{best}	GraphSAGE-4-30	GraphSAGE-5-90	GraphSAGE-6-90	GraphSAGE-7-100
500-30-0	**116014**	107210	107294	107294	115689
500-30-1	**114810**	110572	112015	112870	113420
500-30-2	**116741**	112445	113753	113823	113901
500-30-3	**115370**	108210	111410	112412	115214
500-30-4	**116539**	102492	108212	108673	110673
500-30-5	**115741**	98459	105741	105825	108200
500-30-6	**114181**	102284	107756	107784	109119
500-30-7	**114344**	101251	106121	106901	113901
500-30-8	**115419**	107720	110420	110931	114751
500-30-9	**117116**	105635	112475	113025	115193
500-30-10	**218104**	194514	207174	215794	211437
500-30-11	**214645**	202435	210435	211084	214084
500-30-12	**215945**	191602	208764	209210	213902
500-30-13	**217910**	201861	211344	211750	215031
500-30-14	**215640**	204005	207339	207549	211403
500-30-15	**215919**	200294	210108	210239	213329
500-30-16	**215907**	192376	204817	204952	211613
500-30-17	**216542**	184391	211414	215427	212319
500-30-18	**217364**	203086	210926	211005	216215
500-30-19	**214739**	211427	212049	212253	**214739**
500-30-20	**301675**	285296	294015	294720	298074
500-30-21	**300055**	272181	288528	289170	291497
500-30-22	**305087**	302279	303102	303289	**305087**
500-30-23	**302032**	274412	284002	285374	293216
500-30-24	**304447**	291538	292108	292571	295065
500-30-25	**297012**	270247	282573	283005	289219
500-30-26	**303364**	298305	301455	301532	**303364**
500-30-27	**307007**	281520	292073	292581	297401
500-30-28	**303199**	264199	265250	267071	251200
500-30-29	**300572**	286471	288381	288405	289307

Можемо закључити да је најбољи модел GraphSAGE-7-100 и да се у случају три инстанце најбоља вредност решења се поклапају са решењима из рада [24]. Такође, увиђа се да вредност осталих решења не одступа много од најбољих из литературе. Закључујемо да има смисла проширити модел у погледу да се модел прошири у виду слојева, епохе или величине скупа података.

Време извршавања модела GraphSAGE-7-100 за добијање решења по инстанци кретало се у интервалу $[10s, 115s]$, не рачунајући време обуке, док је за методу FEPSS време било знатно дуже и кретало се у интервалу $[996s, 4285s]$. У поређењу са FEPSS, који захтева значајно дуже време за решавање већих инстанци како би гарантовао оптималност на потпроблему инстанце, модел GraphSAGE остварује вишеструко мању рачунску цену. Иако квалитет добијених решења није на нивоу методе FEPSS, предност GraphSAGE-а огледа се у могућности брзог добијања приближних решења, што га чини погодним за хибридизацију са другим хеуристикама, јер обезбеђује квалитетно полазно решење за даљу локалну претрагу.

4.2 Резултати за WMIS

За тренирање три модела на инстанцама за проблем WMIS са четири слоја, дата су поређења на слици 27. Можемо закључити да је модел GraphSAGE имао најбоље перформансе у овом случају.

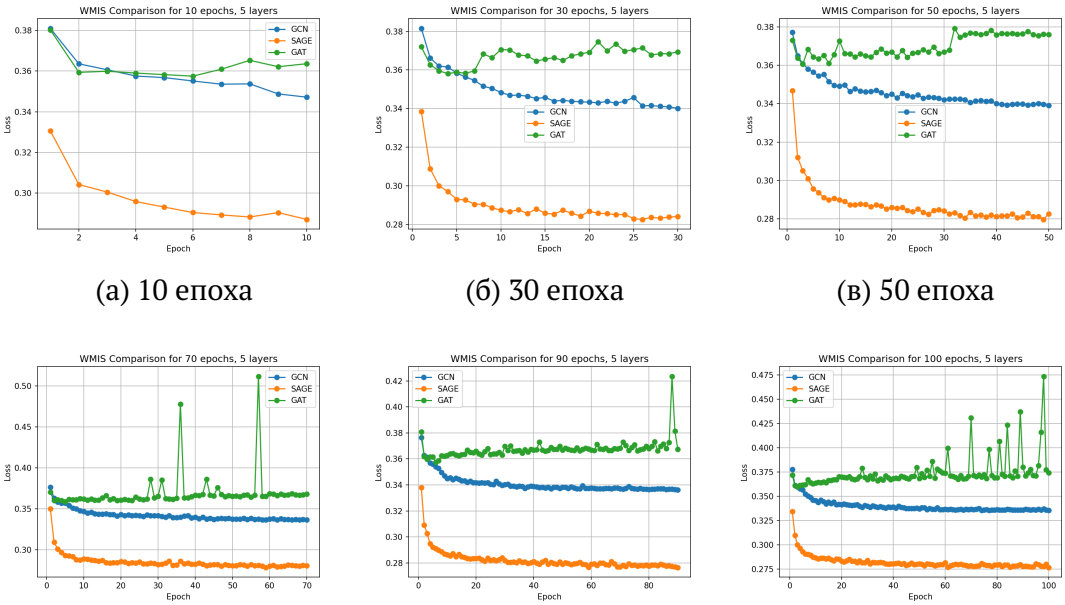


(а) 10 епоха (б) 30 епоха (в) 50 епоха

(г) 70 епоха (д) 90 епоха (ђ) 100 епоха

Слика 27: Поређење резултата за 4 слоја.

У случају са пет слојева, можемо закључити да је модел GraphSAGE, дао најбоље резултате, што можемо видети на основу поређења са слике 28.

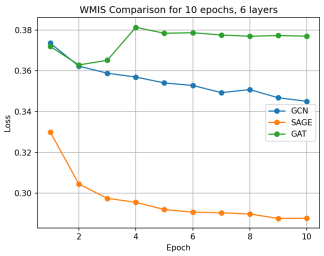


(а) 10 епоха (б) 30 епоха (в) 50 епоха

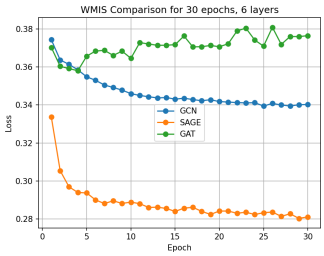
(г) 70 епоха (д) 90 епоха (ђ) 100 епоха

Слика 28: Поређење резултата за 5 слојева.

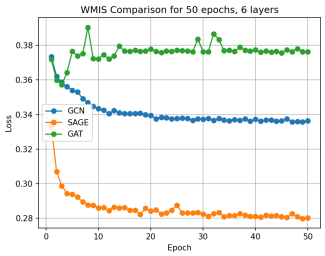
На основу поређења резултата са слике 29, закључујемо да је GraphSAGE најбољи модел за случај са шест слојева.



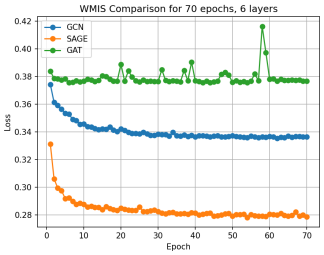
(а) 10 епоха



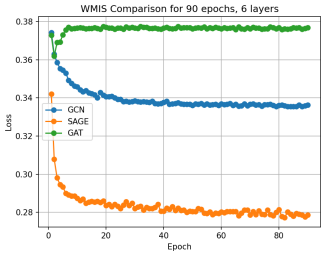
(б) 30 епоха



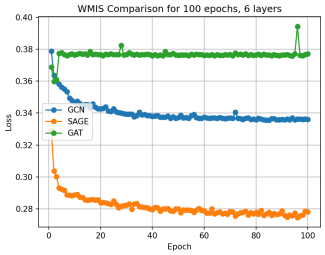
(в) 50 епоха



(г) 70 епоха



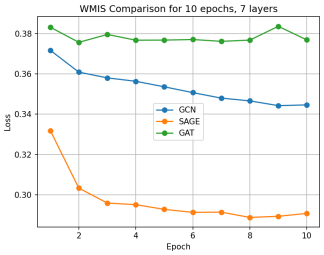
(д) 90 епоха



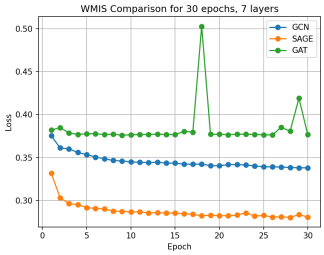
(ђ) 100 епоха

Слика 29: Поређење резултата за 6 слојева.

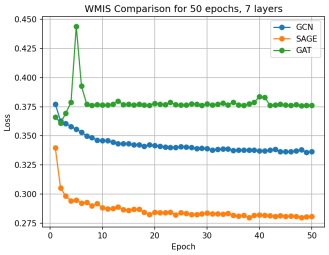
У случају са седам слојева, дата су поређења резултата на слици 30, одакле можемо закључити да се модел GraphSAGE најбоље показао.



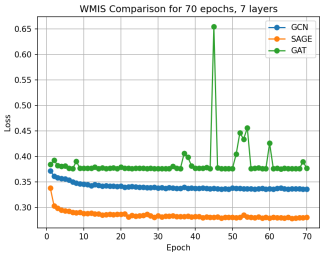
(а) 10 епоха



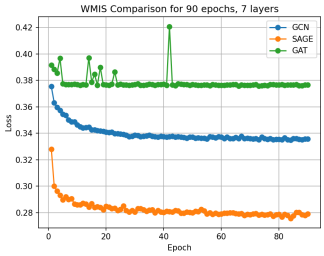
(б) 30 епоха



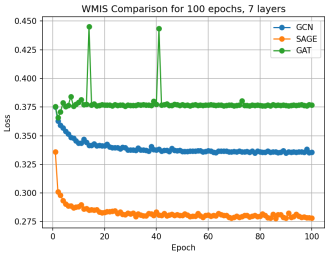
(в) 50 епоха



(г) 70 епоха



(д) 90 епоха



(ђ) 100 епоха

Слика 30: Поређење резултата за 7 слојева.

На основу претходних поређења може се закључити да модел GraphSAGE постиже најбоље резултате. Да би се идентификовао најбољи модел у односу на број слојева и број епоха, из сваке групе формиране према броју слојева издвојени су најперспективнији модели и тестирани на већим инстанцама из реалних примера [28]. Као представници ових група одабрани су модели GraphSAGE-4-70, GraphSAGE-5-70, GraphSAGE-6-90 и GraphSAGE-7-100.

Резултати добијени разматраним моделима на већим инстанцама упоређени су са резултатима из рада [25] који су приказани у табели 2. Најбољи резултати модела су приказани подвученом цртом.

Табела 2: Упоређивање резулта GraphSAGE модела и матхеуристике Red+HILS.

Инстанца	Red+HILS f_{best}	GraphSAGE-4-70	GraphSAGE-5-70	GraphSAGE-6-90	GraphSAGE-7-100
alabama-AM2	**174309**	84317	91209	87310	82114
alabama-AM3	**185744**	107455	113274	112609	118448
district-of-columbia-AM1	**196475**	94301	145807	129420	131644
district-of-columbia-AM2	**209132**	122473	136519	140476	137210
district-of-columbia-AM3	**227598**	155027	146513	148094	149473
florida-AM2	**230595**	99738	130409	139273	137306
florida-AM3	**237333**	163372	159208	165170	169794
georgia-AM3	**222652**	140703	151729	156080	154233
greenland-AM3	**14011**	9002	10358	11037	13704
hawaii-AM2	**125284**	82145	87683	73621	93814
hawaii-AM3	**141011**	92540	105235	107704	109470
idaho-AM3	**77145**	40805	52003	54728	68015
kansas-AM3	**87976**	34086	47205	51520	73702
kentucky-AM2	**97397**	53095	51244	69127	68050
kentucky-AM3	**100486**	58834	64475	62800	84195
louisiana-AM3	**60024**	27103	35660	34705	42583
maryland-AM3	**45496**	22175	24596	26308	33104
massachusetts-AM2	**140095**	83614	95027	93580	127309
massachusetts-AM3	**145866**	51405	84323	97486	114417
mexico-AM3	**97663**	35805	44274	50408	65020
new-hampshire-AM3	**116060**	21591	60385	74205	93179
north-carolina-AM3	**49720**	17357	22093	35726	22414
oregon-AM2	**165047**	73445	91707	90380	141753
oregon-AM3	**175078**	45895	105315	106724	35903
pennsylvania-AM3	**143870**	53775	94837	96150	117424
rhode-island-AM2	**184596**	66294	103841	104451	135069
rhode-island-AM3	**201758**	105240	146728	147041	163824
utah-AM3	**98847**	14078	38305	39140	72582
vermont-AM3	**63302**	11749	41028	37522	48477
virginia-AM2	**295867**	112708	155144	157144	240759
virginia-AM3	**308305**	126539	204148	205652	273010
washington-AM2	**305619**	117305	185247	188031	227460
washington-AM3	**314288**	124381	205703	202850	251026
west-virginia-AM3	**47927**	8386	34788	31405	29730

Резултати указују да нису добијена решења високог квалитета, јер је уочено извесно одступање од тренутно најбољих познатих вредности. До одступања је дошло пре свега услед недовољне разноврсности инстанци коришћених током обуке, што је ограничило способност модела да се успешно прилагоди већим и комплекснијим примерима. У великом броју случајева модел GraphSAGE-7-100 дао је најбоља решења у односу на остале, иако су и други модели остварили добре резултате на појединим инстанцама.

За унапређење модела неопходно је генерисати нове инстанце мањих и средњих димензија, пошто такви скупови не постоје у литератури. Могу се добити редукцијом постојећих великих инстанци, како би се добили подслучајеви са сличним структурним својствима. На тај начин модел би имао разноврснији и прилагођенији скуп података за обуку, што би омогућило већу ефикасност на великим инстанцама. Време извршавања модела GraphSAGE-7-100 за добијања по инстанци припада интервалу $[7s, 44s]$, не укључујући време за обуку, док је за методу Red+HILS време извршавања по инстанци припада интервалу $[0.04s, 974s]$.

5 Закључак

У овом раду дат је детаљан преглед графовских неуронских мрежа, посебно на моделе GCN, GAT и GraphSAGE. Имплементирани су модели машинског учења тако да решавају два позната комбинаторна проблема, MdKP и WMIS. Закључено је да је модел GraphSAGE са 7 слојева и 100 епоха најбољи за проблем MdKP и да је најбољи у већини случаја за WMIS. Када се упореде квалитет решења са резултатима из литературе, закључује се да GraphSAGE даје решења доброг квалитета за проблем MdKP, док то није случај за проблем WMIS. Разлог због ког су резултати били бољи за проблем MdKP је зато што су инстанце из литературе коришћене у тренирању модела графовских неуронских мрежа, док за проблем WMIS постоје само инстанце већих димензија из литературе које нису могле бити употребљене за скуп тренирања.

Наредни корак истраживања може бити да се направи хибридизација са неком S-хеуристиком за проблем MdKP и да се прошири тренутни скуп података како би се креирао модел већих димензија зарад бољих решења. За проблем WMIS, мора да се генерише нови скуп података на основу великих инстанци како би модели налазили квалитетнија решења.

Литература

[1] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems. *University of Michigan Press*.

[2] Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst., Man, Cybern. B* 26(1), 29–41.

[3] Teodorović, D. (2009). Bee Colony Optimization (BCO). Innovations in Swarm Intelligence. *Studies in Computational Intelligence*, vol. 248, Springer, 39–60.

[4] Păun, G. (2000). Computing with membranes. *Journal of Computer and System Sciences* 61(1), 108–143.

[5] Von Neumann, J. (1966). Theory of Self-Reproducing Automata. *University of Illinois Press*.

[6] Adleman, L. M. (1994). Molecular computation of solutions to combinatorial problems. *Science* 266(5187), 1021–1024.

[7] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* 5, 115–133.

[8] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2(4), 303–314.

[9] Kipf, T. N. & Welling, M. (2016). Variational graph auto-encoders. *NeurIPS Workshop on Bayesian Deep Learning*.

[10] Veličković, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., & Hjelm, R. D. (2019). Deep graph infomax. *ICLR*.

[11] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. *NeurIPS*.

[12] Lorie, J. H. & Savage, L. J. (1955). A theoretical approach to portfolio choice. *Journal of Finance*, 10(1), 1–18.

[13] Dantzig, G. B. (1957). Discrete-Variable Extremum Problems. *Operations Research*, 5(2), 266–288.

[14] Weingartner, H. M. & Ness, D. N. (1967). Inclusion of multiple constraints in zero–one programming models. *Operations Research Report ORC* 67-10, Cornell University.

[15] Gavish, B. & Pirkul, H. (1985). Efficient algorithms for solving multiconstraint zero–one knapsack problems to optimality. *Mathematical Programming*, 32(1), 81–106.

[16] Chu, P. C. & Beasley, J. E. (1998). A genetic algorithm for the multi-dimensional knapsack problem. *Journal of Heuristics*, 4(1), 63–86.

[17] Puchinger, J., Raidl, G. R. & Pferschy, U. (2007). The multidimensional knapsack problem: Structure and advanced metaheuristics. *Handbook of Combinatorial Optimization*, Springer, 299–353.

[18] Karp, R. M. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, Plenum, 85–103.

[19] Nemhauser, G. L. & Trotter, L. E. Jr. (1975). Vertex packings: Structural properties and algorithms. *Mathematical Programming* 8, 232–248.

[20] Lamm, S., San Segundo, P. & Schaub, T. (2019). Accelerating local search for the maximum weight independent set problem. *Proc. AAAI 2019*, 2361–2368.

[21] Nogueira, B., Pinheiro, R. C. S. & Subramanian, A. (2018). A hybrid iterated local search heuristic for the maximum weight independent set problem. *Optimization Letters* 12(3), 567–583.

[22] Dong, H., Agostinelli, A., De Armas, J. & Mutzel, P. (2022). A metaheuristic algorithm for large maximum weight independent set problems. *Algorithms* 15(8), 270.

[23] Großmann, J., Lamm, S., Schulz, C. & Strash, D. (2024). Reductions and memetic algorithms for the maximum weight independent set problem. *Journal of Graph Algorithms and Applications* 28(1), 1–33.

[24] Xu, J., Li, H., & Yin, M. (2024). Finding and Exploring Promising Search Space for the 0-1 Multidimensional Knapsack Problem. arXiv:2210.03918v3 [cs.AI]; preprint submitted to *Applied Soft Computing*.

[25] Lamm, S., Schulz, C., Strash, D., Williger, R., & Zhang, H. (2019). Exactly Solving the Maximum Weight Independent Set Problem on Large Real-World Graphs. *Proceedings of the 2019 SIAM Symposium on Algorithm Engineering and Experiments (ALENEX), SIAM*, 144–158.

[26] https://github.com/VladimirJan/MastProject

[27] Beasley, J. E. (1990). OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society* 41(11), 1069–1072. Доступно на: `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

[28] Geofabrik GmbH (n. d.), Geofabrik OpenStreetMap Data Extracts. *Geofabrik*. Доступно на: `https://www.geofabrik.de/data/download.html`