Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Návrh pokročilých algoritmov umelej inteligencie v autonómnej mobilite založenej na využití výpočtov na hrane

Diplomová práca

2023 **Bc. Adam Petík**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Návrh pokročilých algoritmov umelej inteligencie v autonómnej mobilite založenej na využití výpočtov na hrane

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	prof. Ing. Juraj Gazda, PhD.

Košice 2023 **Bc. Adam Petík**

Abstrakt v SJ

S príchodom prepojených autonómnych vozidiel pozorujeme rastúcu potrebu nových riešení alokácie zdrojov v mobilných sieťach. V súčasnosti väčšina riešení prideľovania zdrojov na komunikáciu s vozidlami nezohľadňuje jazdné trasy automobilov a zároveň riešenia dynamického pridelenia zdrojov nezohľadňujú spojito optimalizáciu pridelenia komunikačných aj výpočtových zdrojov alebo ich použitie je značne obmedzené. V tejto práci predstavujeme spojitý výber trasy vozidla a prideľovanie rádiových a výpočtových zdrojov pre prepojené autonómne vozidlá. Navrhovaný prístup je založený na lexikografickom A* algoritme založenom na grafovom vyhľadávaní, ktorý minimalizuje pomer neúspešných úloh pozdĺž celej trasy vozidla vzhľadom na dostupnosť rádiových aj výpočtových zdrojov. Zároveň je navrhnuté riešenie spojitého prideľovania komunikačných aj výpočtových zdrojov, ktoré je založené na hlbokom učení posilňovaním a grafových neurónových sieťach. Efektívnosť navrhovaných prístupov demonštrujú simulácie, ktoré ukazujú, že navrhované algoritmy znižujú pomer nedokončených úloh pred požadovaným termínom až o 60% v porovnaní s existujúcimi najmodernejšími algoritmami.

Kľúčové slová v SJ

prideľovanie zdrojov, viacprístupové výpočty na hrane, bezdrôtové siete, prepojené autonómne vozidlá

Abstrakt v AJ

With the advent of connected autonomous vehicles, we observe a need for new resource allocation solutions in mobile networks. Currently, most resource allocation solutions for connected autonomous vehicles communication do not take into account the driving routes of connected autonomous vehicles, and at the same time, dynamic resource allocation solutions do not take into account the joint optimization of the allocation of communication and computing resources, or their use is significantly limited. In this thesis, we present joint vehicle route selection and radio and computing resource allocation for connected autonomous vehicles. The proposed approach is based on the lexicographic A* algorithm based on graph search, which minimizes the ratio of failed tasks along the entire vehicular route considering the availability of both radio and computing resources. At the same time, we proposed a solution for the continuous allocation

of communication and computing resources, which is based on deep reinforcement learning and graph neural networks. The effectiveness of the proposed approaches is demonstrated by simulations, which show that the proposed algorithms reduce the ratio of failed tasks before the deadline by up to 60% compared to existing state-of-the-art algorithms.

Kľúčové slová v AJ

resource allocation, multi-access edge computing, mobile networks, connected autonomous vehicles

Bibliografická citácia

PETÍK, Adam. *Návrh pokročilých algoritmov umelej inteligencie v autonómnej mobilite založenej na využití výpočtov na hrane*. Košice: Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky, 2023. 73s. Vedúci práce: prof. Ing. Juraj Gazda, PhD.

67699

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Informatika**

Názov práce:

Návrh pokročilých algoritmov umelej inteligencie v autonómnej mobilite založenej na využití výpočtov na hrane

Design of the Advanced Artificial Intelligence Algorithms for the Autonomous Mobility Based on Edge Computing Paradigm

Študent: **Bc. Adam Petík**

Školiteľ: **prof. Ing. Juraj Gazda, PhD.**

Školiace pracovisko: **Katedra počítačov a informatiky**

Konzultant práce:

Pracovisko konzultanta:

Pokyny na vypracovanie diplomovej práce:

1. Vytvoriť komplexný realistický simulátor autonómnej mobility vozidiel s podporou výpočtov na hrane v programovacom prostredí Python.
2. Identifikovať vhodné algoritmy umelej inteligencie pre použite v oblasti navigácie autonómneho vozidla.
3. Vybrané algoritmy vhodne modifikovať pre prípadovú štúdiu navigácie autonómneho vozidla.
4. Porovnať prevádzkové vlastnosti navrhnutých algoritmov s konkurenčnými state-of-the-art prístupmi.

Jazyk, v ktorom sa práca vypracuje: slovenský

Termín pre odovzdanie práce: 21.04.2023

Dátum zadania diplomovej práce: 31.10.2022

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY -12-

v.r.

prof. Ing. Liberios Vokorokos, PhD.

dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 21.4.2023 .....

Vlastnoručný podpis

Poďakovanie

Chcel by som sa touto cestou veľmi pekne poďakovať svojmu vedúcemu práce prof. Ing. Jurajovi Gazdovi, PhD. za jeho odborné vedenie a podporu počas riešenia mojej záverečnej práce. Taktiež by som sa chcel poďakovať všetkým ľuďom z výskumného laboratória IISLab TUKE, ktorí mi poskytli cenné rady, podporu pri realizácii mojej práce a dali mi možnosť podielať sa na výskume v modernej oblasti. Napokon ďakujem svojej rodine a priateľom za neustálu podporu a povzbudenie.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Viacprístupové výpočty na hrane (z angl. Multi-access Edge Computing, MEC) sa ukázali ako sľubná technológia, ktorá umožňuje poskytovanie výpočtových a komunikačných služieb mobilným používateľom s nízkou latenciou a veľkou šírkou pásma vďaka 5G a očakávaných 6G sieťach. MEC môže zohrávať kľúčovú úlohu pri umožňovaní pripojeným autonómnym vozidlám (z angl. Connected Autonomous Vehicles, CAVs) spracovávať a vymieňať si údaje v reálnom čase. Očakáva sa, že autonómne vozidlá budú generovať až minimálne 3 Gb dát za sekundu, a práve MEC môže zaistiť bezpečnosť, efektivitu a lepší zážitok z jazdy umiestnením serverov v blízkosti bázových staníc. CAVs sú momentálne rýchlo sa rozvíjajúcou oblasťou výskumu a vývoja, pričom mnohé zainteresované strany do tejto technológie výrazne investujú.

Integrácia MEC s CAVs predstavuje výzvy, ktoré je potrebné riešiť, najmä pokiaľ ide o prideľovanie MEC zdrojov. Jednou z hlavných výziev je zabezpečiť, aby výpočtové a komunikačné zdroje MEC boli dostupné pre CAVs počas ich cestovania. To si vyžaduje vývoj účinných algoritmov, ktoré dokážu vybrať vhodné trasy pre vozidlá, pričom sa zohľadní dostupnosť zdrojov MEC na trase. Výber vhodnej trasy pre CAV je kľúčový, pretože môže ovplyvniť kvalitu služieb, spoľahlivosť výmeny údajov a celkový výkon systému. Navrhovaný algoritmus by mal brať do úvahy rôzne faktory spolu s dostupnosťou MEC zdrojov.

Okrem toho si integrácia CAVs s MEC vyžaduje riešenie otázky prideľovanie zdrojov v dynamickom prostredí. CAVs si potrebujú vymieňať veľké množstvo údajov s inými vozidlami, cestnou infraštruktúrou a cloudovými službami a spracovávať tieto údaje v reálnom čase. CAVs sa navyše premiestňujú v čase a na rôznych miestach potrebujú rôzne množstvo MEC zdrojov na splnenie ich požiadaviek, čo závysí hlavne od kvality mobilného signálu. Je tak potrebné efektívne prideľovať zdroje ako komunikačné, tak výpočtové, čo môže zaručiť včasné spracovanie údajov a poskytnúť požadovanú kvalitu služieb pre všetky vozidlá využívajúce MEC.

Na riešenie týchto výziev táto práca navrhuje algoritmus, ktorý vyberá vhod-

né trasy pre CAVs, aby sa zabezpečilo, že počas cesty budú mať zdroje MEC k dispozícii. Algoritmus je zároveň navrhnutý tak, aby zohľadnil aj dĺžku trasy. Dodatočne je navrhnuté riešenie na dynamickú alokáciu zdrojov MEC na základe požiadaviek vozidiel v prostredí. Riešenie využíva prístup založený na strojovom učení nazvaný hlboké učenie posilňovaním (z angl. Deep Reinforcement Learing, DRL) na dynamické prideľovanie zdrojov, čím sa zaisťuje čo najväčšia úspešnosť z hľadiska vypočítaných úloh do požadovaného času. Navrhované riešenie by malo byť schopné dynamicky upravovať pridelenie zdrojov na základe aktuálneho dopytu, aby sa zabezpečilo, že systém spĺňa požadované výkonnostné kritériá.

Navrhované riešenia sú vyhodnotené pomocou simulácií. Výsledky ukazujú, že navrhovaný algoritmus a riešenie na dynamickú alokáciu zdrojov MEC môže výrazne zlepšiť úspešnosť vypočítaných úloh CAV a zároveň môžu tiež pomôcť znížiť náklady na nasadenie a prevádzku služieb MEC.

Cieľom tejto práce je prispieť do oblasti MEC a CAV návrhom efektívnych riešení na alokáciu zdrojov, aby sa zabezpečilo spracovanie úloh do požadovaného času. Navrhované algoritmy a riešenia majú mať potenciál byť implementované v reálnych systémoch a prispievať k rozvoju inteligentnejšieho a efektívnejšieho dopravného systému, preto je potrebné ich porovnať so súčasnými metódam.

Výsledky diplomovej práce boli publikované v časopise *IEEE Transactions on Intelligent Transportation Systems* s IF = 9.5 [1].

Formulácia úlohy

V rámci práce budeme riešiť nasledujúce úlohy. V prvom rade je potrebné predstaviť a oboznámiť sa s MEC ako takým. Je potrebné pochopiť, na akom princípe fungujú bázové stanice, a ako prebieha komunikácia medzi používateľom a bázovou stanicou, aby sme vedeli vytvoriť presnú simuláciu. Ďalej je potrebné analyzovať doterajšie riešenia, identifikovať ich nedostatky a na základe ich analýzy vytvoriť a implementovať vlastné riešenia v oblasti plánovania vhodnej trasy a dynamickom rozdeľovaní MEC zdrojov. Ďalšou úlohou je vytvorenie simulačného prostredia v Python programovacom jazyku. Nároky na simulačné prostredie sú také, aby obsahovalo mobilitu používateľov a bolo schopné simulovať odosielanie a výpočty úloh v MEC systéme. Poslednou úlohou je objektívne vyhodnotiť výsledky simulácií v rôznych scenároch a porovnať jednotlivé algoritmy medzi sebou, a ak je to možné, porovnať ich aj so state-of-the-art prístupmi.

1 Viacprístupové výpočty na hrane

MEC je sieťová architektúra definovaná štandardizačnou organizáciou ETSI [2]. Ponúka výpočtové prostriedky na hrane v rámci rádiovo prístupnej siete v blízkosti koncovým používateľom. To znamená, že hlavným cieľom MEC je zníženie latencie, zníženie zaťaženia chrbticovej siete [3] čo v konečnom dôsledku zvyšuje používateľskú skúsenosť a dovoľuje využívať aplikácie, ktoré by na mobilných zariadeniach neboli schopné uspokojiť potreby používateľov. Alternatívou ku MEC je Mobile Cloud Computing (MCC), ktorý ponúka veľké výpočtové a úložiskové prostriedky v dátových centrách ďaleko od konečného používateľa. Porovnanie MEC a MCC je zhrnuté v tabuľke 1.1.

	MCC	**MEC**
Fyzický server	Vysoké výpočtové a úložiskové kapacity lokalizované v centralizovaných dátových centrách	Limitované výpočtové a úložiskové zdroje lokalizované pri bázových staniciach
Vzdialenosť prenosu	Ďaleko od používateľov, desiatky až tisíce kilometrov	Blízko pri používateľoch, desiatky až stovky metrov
Systémová architektúra	Sofistikovaná konfigurácia, vysoko centralizované	Jednoduchšia konfigurácia, decentralizované
Charakteristika aplikácie	Výpočtovo intenzívne a tolerujúce latenciu, napr. sociálne siete	Závislé od nízkej latencie a výpočtovej náročnosti, napr. autonómna doprava

Tabuľka 1.1: Porovnanie MEC a MCC [3]

Z hľadiska výpočtových a úložiskových zdrojov má MCC dostupné oveľa väčšie množstvo zdrojov ako MEC, pretože používa cloud, teda výpočtové centra, v ktorých sú zdroje prakticky neobmedzené. Budovy obsahujúce servery disponujú chladiacimi zariadeniami a odpovedajúcim napájacím zdrojom. Na druhej strane v prípade MEC sa servery nachádzajú v bázových staniciach alebo blízko nich. Musia zaberať menej miesta a nemajú k dispozícii také chladiace a napájacie prostriedky ako v prípade cloud-ového centra. Z toho dôvodu sú výpočtove

a úložiskové zdroje MEC obmedzené. Využitie cloud-u v MCC systémoch má za následok veľkú vzdialenosť od koncového používateľa, ktorá môže byť až v tisíckach kilometrov. V prípade MEC je vzdialenosť od koncových používateľov výrazne nižšia, desiatky až stovky metrov. MCC systémy vo väčšine využívajú cloud-ových poskytovateľov ako Azure alebo Google. MEC systémy sú distribuované a väčšinou spravované telekomunikačnými operátormi, podnikmi alebo komunitami a ich konfigurácia je jednoduchšia ako konfigurácia MCC.

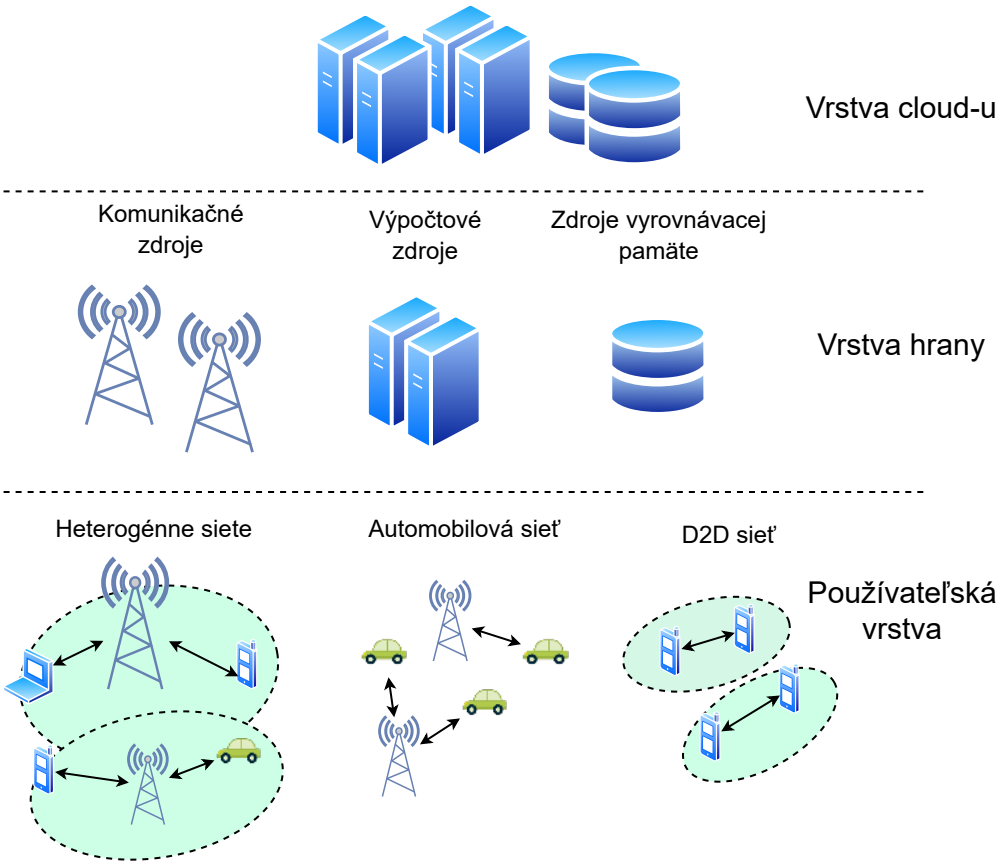
Z uvedených charakteristík vyplýva využitie aplikácii. MCC je prirodzene vhodný na aplikácie, ktoré nevyžadujú nízku latenciu. Tieto aplikácie môžu byť aj výpočtovo náročne, pretože cloud je schopný takúto záťaž zvládnuť. Príkladom takýchto aplikácií môžu byť sociálne siete, napríklad Facebook alebo aplikácia Uber. Pre MEC sa hodia aplikácie vysoko závislé od nízkej latencie. Výpočtová náročnosť aplikácie by nemala presahovať kapacity MEC servera. Avšak tým, že v MEC sú servery distribuované, sa tu otvára možnosť paralelných výpočtov na viacerých serveroch naraz [4], čo môže zvýšiť výpočtovú kapacitu pre konečného používateľa, ale s tým je spojené aj isté zaťaženie komunikačných zdrojov. Vhodnou aplikáciou pre MEC môže byť autonómna doprava, pomocou ktorej autá komunikujú medzi sebou cez MEC alebo odosielajú úlohy na výpočet do MEC siete.

Radi by sme poznamenali, že MEC nie je samostatný systém, ale skôr dopĺňa MCC o zmienené funkcie. Vo veľkej väčšine prípadov stále potrebujeme centralizovaný systém na ukladanie a spracovanie dát, na jednoduchšie riadenie a pod.

1.1 Architektúra MEC

Architektúru MEC na vysokej úrovni môžeme rozdeliť do troch vrstiev[3]: používateľská vrstva, hranová vrstva, vrstva cloud-u, ako je znázornené na obrázku 1.1. Používateľská vrstva v sebe zahŕňa rôznych koncových používateľov a aplikácie. Vrstva sa skladá predovšetkým zo zariadení ako vozidlá, mobilné telefóny, senzory atď. Tieto zariadenia komunikujú cez mobilnú sieť s MEC servermi. Podľa spôsobu komunikácie medzi koncovými zariadeniami a bezdrôtovou infraštruktúrou sa používateľská vrstva môže rozdeliť na:

- Heterogénna sieť: Od nasledujúcich generácií bezdrôtových sietí sa očakáva aj prístup k aplikáciám, ktoré budú vyžadovať vysoký dátový prenos. Jednou z možností ako zvýšiť bezdrôtový dátový prenos je vytvorenie siete s viacerými malými bázovými stanicami [3][5]. Takéto riešenie dokáže zároveň aj znížiť spotrebu energie koncového zariadenia pretože server sa bude nachádzať bližšie, a to znamená nižšiu potrebu energie na prenos in-



Obr. 1.1: Hierarchická architektúra MEC [3]

formácie. Práve takýmto sieťam obsahujúcim rôzne veľké bázové stanice, macrocell-y, femtocell-y, microcell-y, hovoríme heterogénne siete. Obyčajne sú tieto bázové stanice vybavené výpočtovými prostriedkami pripravenými na využite a zdrojovo obmedzené zariadenia môžu tieto zdroje využiť na posielanie úloh na výpočet.

- Sieť vozidiel: Sieť tvoria vozidla spolu s chodcami a bázovými stanicami, môže byť vytvorená kdekoľvek pri cestnej komunikácii, či už ide o mestá alebo diaľnice. Pre komunikáciu s vozidlami a okolím sa používajú technológie IEEE 802.11p alebo Long Term Evolution (LTE) [6]. Komunikácia vozidiel s okolím sa označuje pojmom komunikácia vozidlo-všetko (z angl. vehicle-to-everything) a zahŕňa komunikáciu vozidlo-vozidlo (z angl. vehicle-to-vehicle), komunikáciu vozidlo-chodec (z angl. vehicle-to-pedestrian) a komunikáciu vozidlo-infraštruktúra (z angl. vehicle-to-infrastructure). S príchodom nových technológií sa vytvára priestor pre aplikácie pre sieť vozidiel, od bezpečnosti až po zábavu, ako napríklad strímovanie médií, identifikovanie parkovacieho miesta alebo kontrola mŕtveho uhla vozidla.

V sieti je možné využiť vozidlá na výpočtove zdroje, ak mobilné zariadenie, či už ďalšie vozidlo, alebo mobilný telefón potrebuje viac zdrojov, ako je schopné poskytnúť. Okrem toho sú stále prístupne zdroje na hrane. Vozidlá sa však pohybujú vyššou rýchlosťou, čo spôsobuje zmenu topológie siete, a to je výsledkom dodatočných výziev, ktoré je potrebné riešiť.

- Siete mobil-mobil / zariadenie-zariadenie: Ide o sieť, v ktorej zariadenia komunikujú medzi sebou priamo bez využitia bázových staníc [7]. To umožňuje využiť priamo iné zariadenie na spracovanie úlohy, na distribúciu obsahu alebo na sprostredkovanie komunikácie medzi iným zariadením a bázovou stanicou. To potencionálne dokáže zlepšiť priepustnosť a využitie energie[7].

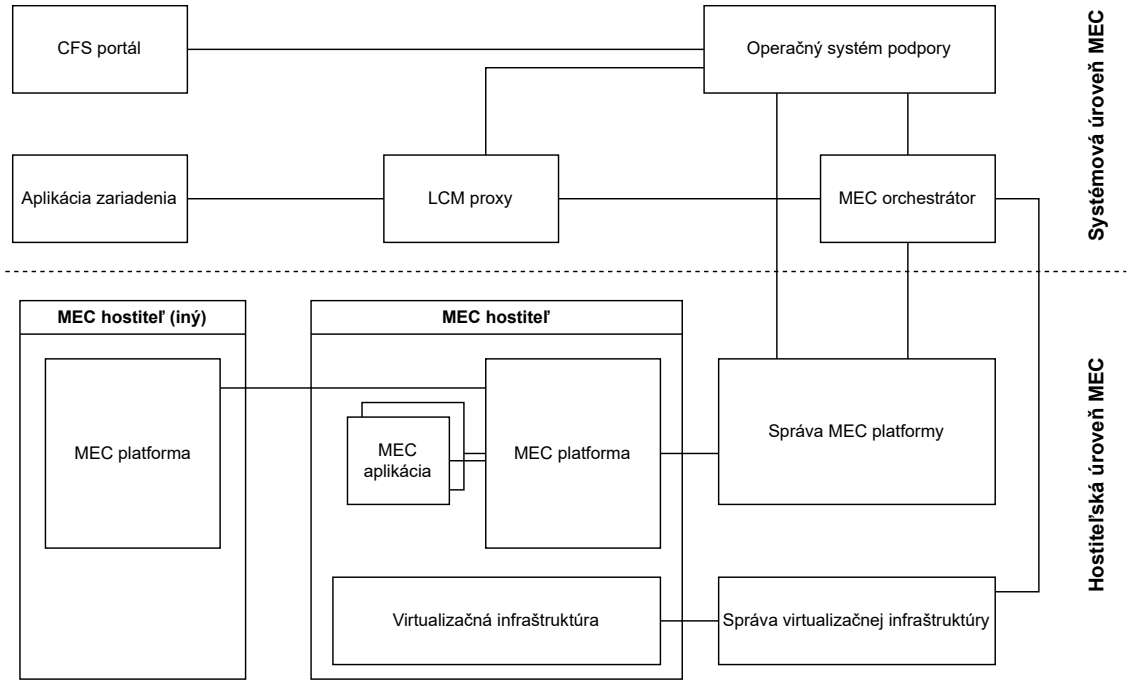
Vrstva hrany sa nachádza v strede v hierarchickej štruktúre. Obsahuje výpočtové servery, ktoré sú umiestnené v bazových staniciach alebo v iných častiach siete.Funkciu MEC servera môžu plniť aj vozidlá, tablety, mobily. Vďaka tomu vrstva poskytuje bezdrôtové distribuované výpočtové a komunikačné zdroje vo frekventovaných miestach – na uliciach, v blízkosti obchodných domov, ciest a pod. Vrstva hrany poskytuje tri typy zdrojov:

- komunikačné zdroje – širka pásma, spektrum a vysielací výkon.
- ukladanie zdrojov do pamäte – predstavuje pamäť MEC servera.
- vypočtové zdroje – v podobe CPU cyklov.

Výpočtové a pamäťové zdroje sú v MEC serveroch zväčša limitované, preto je na efektívne využitie zdrojov potrebné optimalizovať spojito komunikačné, pamäťové a výpočtové zdroje.

Vrstva cloud-u predstavuje vysoko výkonné výpočtové servery s veľmi veľkými kapacitami, či už výpočtovými, komunikačnými alebo úložiskovými. Používajú sa na spracovanie a ukladanie veľkých dát, na manažovanie MEC serverov a na aplikácie, ktoré nie sú náročné na rýchlosť odozvy.

Viac technická a podrobná architektúra MEC je znázornená na obrázku 1.2. Ide o generickú architektúru MEC od organizácie ETSI [8], ktorá definuje MEC štandardy. Obrázok znázorňuje funkčné komponenty MEC, ktoré sú rozdelené do dvoch vrstiev – systémová a hostiteľská vrstva. Správa systémovej vrstvy má dohľad nad celým MEC systémom vďaka MEC orchestrátoru. Správa hostiteľskej vrstvy obsahuje spravu MEC platformy a virtualizačnú infraštruktúru konkrétneho MEC hostiteľa.



Obr. 1.2: MEC systémová referenčná architektúra

Ako vidno z obrázka 1.2, s MEC systémom môže priamo komunikovať aplikácia zariadenia alebo zákazníci tretích strán cez portál zákazníckych služieb (CFS). V oboch prípadoch sa komunikuje s operačným systémom podpory (OSS). OSS spracováva požiadavky na spustenie alebo ukončenie aplikácie a rozhoduje o vyhovení požiadavky. Ak požiadavka vyhovuje, posiela sa MEC orchestrátorovi. MEC orchestrátor má nadhľad nad celým MEC systémom, o tom, ake sú dostupné zdroje, aká je topológia a podobne. Plní nasledujúce funkcie [8][9]:

- Vyberá, na ktorom MEC hostiteľovi sa bude aplikácia vykonávať berúc do úvahy dostupné zdroje.
- Validuje požiadavky a pravidlá aplikácie, a ak je to potrebné, upravuje ich podľa politiky operátora.
- Rozhoduje o inicializácii alebo terminácii aplikácie.
- Rozhoduje o realokácii aplikácie.

Aplikácia zariadenia nekontaktuje priamo OSS, ale medzi nimi vystupuje ešte komponent proxy na správu životného cyklu používateľských aplikácií (LCM proxy). Prijíma požiadavky a autorizuje ich pred tým, ako ich posiela OSS alebo MEC orchestrátorovi.

Systémová MEC vrstva je prepojená s hosťovskou MEC vrstvou cez správu MEC platformy a správu virtualizačnej infraštruktúry. Správa MEC platformy

spravuje životný cyklus aplikácii, autorizáciu, požiadavky aplikácie a pod. Správa virtualizačnej infraštruktúry je zodpovedná za alokáciu, spravovanie a uvoľnenie virtualizovaných zdrojov poskytovaných virtualizačnou infraštruktúrov [9].

Z hľadiska nasadenia MEC serverov sú na výber tri možnosti[9]. Prvou je nasadenie serverov priamo na bázové stanice. Druhou je možnosť nasadiť servery v mieste agregácie viacerých bázových staníc, napríklad na pokrytie obchodného centra. Treťou možnosťou je presunúť servery ďalej od používateľov až na hranu základnej siete operátora. To ktorú možnosť vybrať závisí od požiadaviek na MEC aplikáciu.

1.2 Odosielanie úloh na výpočet

Jedným z využití MEC je odoslanie úloh na výpočet (z angl. computation offloading). Ide o prístup, pri ktorom sa na vzdialene servery pošle úloha na výpočet a výsledok sa odošle naspäť. Dôvody na odoslanie úlohy môžu byť zníženie spotreby energie mobilného zariadenia a zrýchlenie výpočtu úlohy, teda zníženie latencie spracovania úlohy. Pri odosielaní úlohy na výpočet je potrebné vyriešiť, na aký server sa úloha má odoslať, aby sa vypočítala v prípustnom čase a či vôbec odoslanie úlohy na výpočet prináša výhodu oproti výpočtu úlohe v mobilnom zariadení. Podľa charakteru úlohy môže prichádzať do úvahy úlohu rozdeliť a odoslať na výpočet iba časť úlohy. Z toho môžu vzniknúť tri prípady odosielania úlohy do MEC siete [9]:

- Lokálne spracovanie úlohy – celý výpočet je vykonaný lokálne na mobilnom zariadení. To môže nastať z dôvodu nedostupnosti MEC siete alebo zistenia, že odoslanie úlohy a jej spracovanie v MEC by trvalo dlhšie ako lokálny výpočet.
- Plné odoslanie úlohy – celý výpočet je odoslaný do MEC siete.
- Čiastočne odoslanie úlohy – časť úlohy je vypočítaná lokálne v mobilnom zariadení a druhá časť úlohy sa odosiela na výpočet do MEC siete.

Pri odosielaní úlohy na výpočet do MEC siete s cieľom znížiť čas výpočtu je potrebné brať do úvahy tri časy: čas potrebný na odoslanie úlohy T_o, čas potrebný na spracovania úlohy v MEC sieti T_s a čas potrebný na odoslanie výsledku T_v. Celková latencia spracovania úlohy je potom $T_{ot} = T_o + T_s + T_v$. Odoslanie úlohy je vhodné vykonať vtedy, ak čas potrebný na lokálny výpočet úlohy T_l je väčší ako T_{ot}.

1.3 Mobilné siete v MEC

S príchodom 5G a 6G sietí sa rozširujú možnosti MEC využitia [10][11]. 5G je navrhnutá tak, aby podporovala tri hlavne oblasti: vylepšené mobilné širokopásmové pripojenie (eMBB), masívna komunikácia typu stroja (mMTC) a mimoriadne spoľahlivá komunikácia s nízkou latenciou (URLLC). S týmto je spojená aj ďalšia vlastnosť 5G siete nazvaná rozdelenie siete (z angl. network slicing) [12]. Operátor si tak môže rozdeliť sieť na viac častí, kde sú všetky časti od seba oddelené a zvlášť konfigurovateľné. Každá MEC aplikácia alebo služba požaduje rôzne charakteristiky MEC siete a práve rozdelenie siete nám umožňuje sieť využiť efektívne a izolovať aplikácie od seba. 5G vytvorilo priestor pre aplikácie ako napríklad smart home, home office alebo vzdialené vzdelanie, ale aj aplikácie pre sieť vozidiel.

6G siete posúvajú hranicu ešte ďalej. Očakáva sa veľký nárast komunikácie mobilných zariadení, ktorá bude pre 5G siete privysoká[11]. Od 6G sa očakáva podpora nových oblastí, ktoré budú, obrazne povedané, tvorené prienikmi medzi oblasťami z 5G. Prienikom scenárov eMBB a mMTC vznikne uMBB (z angl. ubiquitous mobile broad band). ULBC (z angl. ultra-reliable low-latency broadband communication) bude spĺňať URLLC a aj eMBB. Tretím scenárom je mULC (z angl. massive ultra-reliable low-latency communication) spĺňajúci URLLC a mMTC. Prípady použitia 5G sa v 6G ďalej rozšíria. Očakávajú sa oveľa priateľnejšie podmienky na rozšírenú realitu a technologické dvojčatá, ktoré sa už vyskytujú, ale v obmedzenom režime. Vďaka nižšej latencie a spoľahlivosti oproti 5G sa očakáva aj veľký prínos v inteligentnej doprave a logistike a autonómnych áut.

1.3.1 Bázové stanice

Bázová stanica označuje stanicu poskytujúcu rádiové spojenie. V LTE (4G) sieťach sa označuje ako eNB, v 5G sieťach gNB. V Automobilových komunikačných systémoch sa používa označenie RSU (skratka z angl. roadside unit) pre jednotky komunikujúce s vozidlami. V práci budeme ďalej používať označenie gNB pre jednú bázovú stanicu a označenie gNBs pre množné číslo. V spojitosti s gNB nás zaujíma to, ako sa signál šíri, aby sme to vedeli simulovať. Na prenos signálu potrebujeme výkon označený, ako P_{t}. Vyslaný signál sa postupne zoslabuje. Na vyjadrenie miery zoslabnutia signálu sa používa výraz z angl. path loss. Path loss

zavisí od prostredia, pre voľné prostredie (FSPL) platí nasledujúci vzťah[13]:

$$\mathrm{FSPL} = \frac{P_\mathrm{t}}{P_\mathrm{r}} = \left(\frac{4\pi d}{\lambda}\right)^2, \tag{1.1}$$

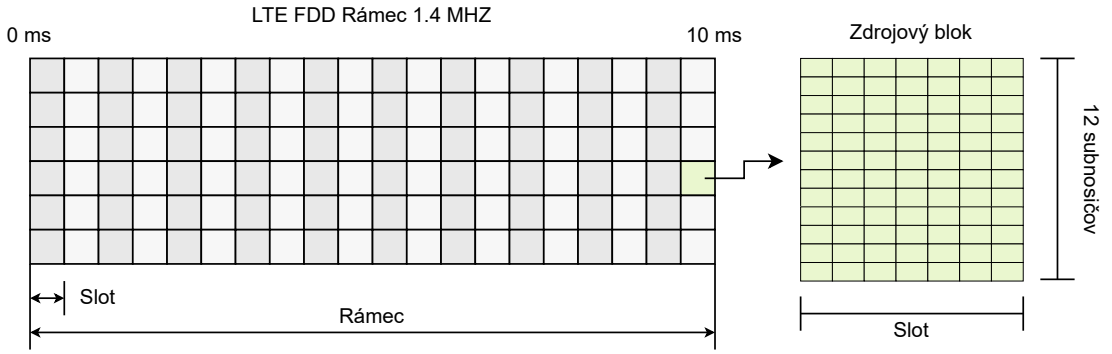
kde P_r je prijatý výkon, d je vzdialenosť od vysielacieho bodu ku prijímaciemu bodu, λ je vlnová dĺžka. Väčšinou sa FSPL udáva v decibeloch, potom vzťah vyzerá nasledovne:

$$\mathrm{FSPL} = 20\log_{10}(d) + 20\log_{10}(f) - 147.55, \tag{1.2}$$

kde f je nosná frekvencia bázovej stanice v Hz a d je vzdialenosť v metroch.

Nosná frekvencia je frekvencia, ktorá sa transformuje pri modulácii za účelom vloženia informácii do signálu[14].

1.3.2 Fyzická vrstva LTE



Obr. 1.3: Príklad fyzickej vrstvy LTE[15]

Na presnejšie implementovanie rádiovej časti v simulácii a na tvorbu riešenia je potrebné pochopiť fyzickú vrstvu LTE siete. Na obrázku 1.3 je znázornení FDD rámec o šírke pásma 1,4 MHz. Šírka pásma predstavuje dostupnú šírku frekvencie na komunikáciu, na obrázku je znázornená zvislou osou. Pozdĺž osi vodorovnej je čas, a vidíme, že čas je rozdelený na sloty o veľkosti 0.5 ms. Slot je ďalej rozdelený v tomto prípade na sedem častí, pričom tieto časti nazývame zdrojovými blokmi (z angl. resource blocks, RB). Zdrojové bloky (RBs) predstavujú komunikačné zdroje, ktoré sa alokujú používateľom. Jeden RB zaberá 180kHz, preto je šírka pásma na obrázku rozdelená do šesť častí. Obyčajne platí, že do 1 MHz šírky pásma sa zmestí päť RBs. Napríklad 10 MHz šírka pásma bude obsahovať 50 RBs.

Zdrojový blok sa ďalej ešte delí na menšie časti volané symboly a subnosiče. Jeden subnosič obsahuje 7 symbolov a vo väčšine kanálov a signálov je 12 subnosičov v jednom RB. Element 1 subnosič $\times$ 1 symbol je považovaný za najmenšiu časť fyzického rámca a reprezentuje dáta z fyzického kanálu alebo signálu.

1.3.3 SNR a SINR

SNR (z angl. signal-to-noise ratio) vyjadruje pomer sily signálu k sile šumu [16], teda

$$SNR = \frac{P_s}{P_N} \tag{1.3}$$

kde P_s je sila signálu a P_N je sila šumu.

Okrem SNR poznáme aj SINR (z angl. signal-to-interference-plus-noise ratio), ktorý rozširuje SNR o interferenciu iných gNBs. SINR sa využíva na meranie kvality signálu v bezdrôtových spojeniach. Na rozdiel od drôtového spojenia v bezdrôtovom vznikajú rôzne iné faktory ovplyvňujúce prenos ako napríklad slabnutie signálu, interferencia. SINR sa snaží tieto aspekty zachytiť. Vzorec SNR sa rozšíril o interferenciu I [17]:

$$SINR = \frac{P_s}{P_N + I} \tag{1.4}$$

SINR sa obyčajne počíta vzhľadom na požívateľa. P_s predstavuje prichádzajúcu silu signálu záujmu, I predstavuje iné signály v sieti, ktoré interferujú so signálom záujmu. P_N predstavuje šum, obyčajne sa pre šum používa v simuláciách konštanta podľa prostredia, v ktorom sa vykonáva simulácia.

2 Použiteľné metódy hlbokého učenia v MEC

V oblasti MEC sa nachádza využitie najmä pre hlboké učenie posilňovaním (DRL). Viacero autorov prišlo s návrhmi, ako sa dá využiť DRL na alokáciu zdrojov na bázových staniciach, alebo pre rozhodovanie, kam sa má odoslať úloha. Okrem DRL sa v poslednom čase aplikujú v MEC aj grafové neurónové siete (z angl. Graph Neural Networks, GNN). GNN je v čase písania tejto práce ešte stále novinkou. Ide o neurónové siete (NN), ktoré spracovávajú dáta v podobe grafu. V tejto sekcii sa zameriame na analýzu GNN a predstavíme si algoritmus DRL, ktorý budeme v práci používať.

2.1 Grafové neurónové siete

V súčasnej dobe sa zvyšuje množstvo prípadov[18], kde sú dáta reprezentované formou grafu. Ide napríklad o oblasti v e-commerce, kde chceme formovať vzťahy medzi zákazníkmi, a tak odporúčať vhodné produkty na predaj. V chémii sa molekuly takisto formujú pomocou grafov a môžeme predpovedať rôzne vlastnosti látok.

Grafy môžu obsahovať rôzny počet vrcholov a hrán rôzne zoradených a grafy môžu byť rôzneho tvaru. To znamená väčšiu komplexnosť dát a vytváranie výziev pre spracovanie takých dát pomocou strojového učenia. Operácie ako konvolúcia sa dajú jednoducho aplikovať na obrázky, ale pri grafoch je to už náročnejšie. Podľa úlohy, ktorú chceme riešiť, poznáme tri úrovne, ako môžeme grafové dáta spracovať:

- na úrovni grafu – chceme určiť vlastnosť vzhľadom na celý graf, napríklad predpovedať vôňu molekuly. Inými slovami sa snažíme reprezentovať graf v kompaktnejšej forme. Preto sa používajú GNN skombinované s *pooling* a *readout* operáciami.

- na úrovni uzlov – chceme určiť vlastnosti pre každy z uzlov – klasifikovať uzly alebo robiť regresiu. Viaceré GNN vrstvy dokážu extrahovať reprezentáciu uzlov za pomoci propagácie informácií zo susedných uzlov.
- na úrovni hrán – chceme určiť vlastnosti pre každú hranu podobne ako v prípade na úrovni uzlov.

Autori v [18] rozdelili GNN do niekoľkých kategórií. Prvou sú rekurentné grafové neurónové siete (RecGNN). Prvé GNN boli práve tohto typu. Využíva sa tu propagovanie informácií od susedných uzlov tak ako pri klasických rekurentných neurónových sieťach. Predpokladá sa, že susedia si vymieňajú informácie dovtedy, dokým nedosiahnu stabilné equilibrium. RecGNN môžu mať problém s natrénovaním pri veľkých grafoch.

Druhou kategóriou sú konvolučné grafové neurónové siete (ConvGNN). V súčasnej dobe naberajú na popularite. Na rozdiel od RecGNN, kde sa používa rekurentná vrstva a s ňou spojený skrytý stav, sa pri ConvGNN používa viacero vrstiev za sebou, a teda skrytý stav sa prenáša z vrstvy na vrstvu a mení sa. ConvGNN sa ešte ďalej dajú rozdeliť na spectral based a spatial based. Konkrétnejšie sa zameriame na spatial based, pretože tie sú preferované z dôvodu efektivity, všeobecnosti a flexibility [18]. Message-passing neurónová sieť (MPNN) sa javí ako základný rámec spatial based sietí [18][19]. MPNN dekomponuje grafovú konvolúciu na dve fázy, fáza agregácie a fáza kombinácie. Fáza agregácie agreguje správy od susedov uzla a fáza kombinácie skombinuje výslednú agregovanú správu s aktuálnym stavom uzla. Tieto operácie môžeme zapísať nasledovne:

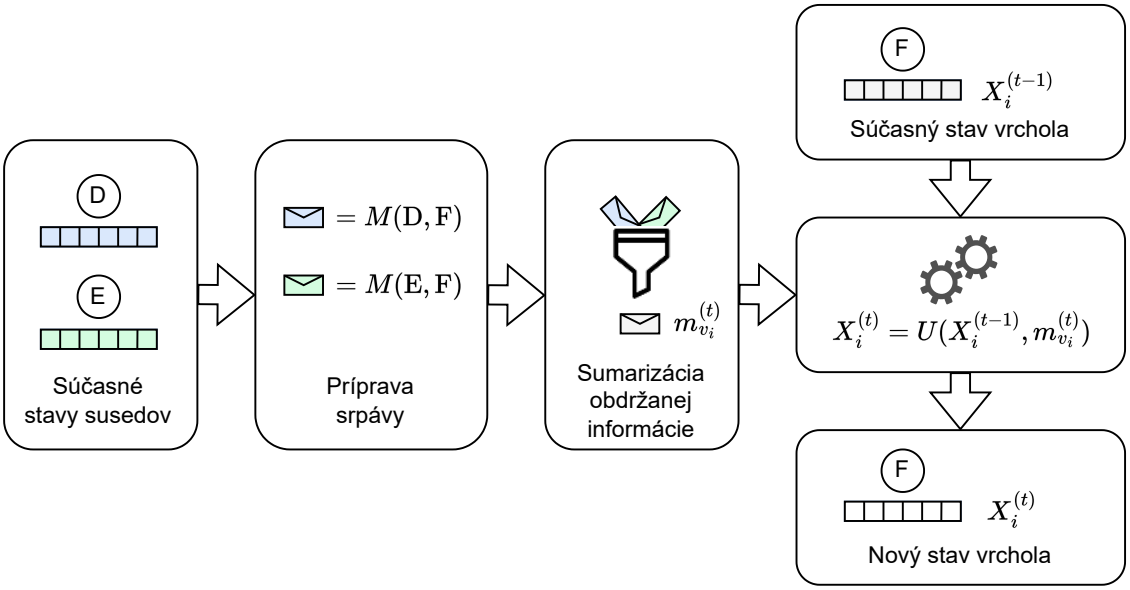
$$m_{v_i}^{(t)} = \sum_{v_j \in N(v_i)} M^{(t)}\Big(X_i^{t-1}, X_j^{t-1}, e_{ij}\Big), \tag{2.1}$$

$$X_i^{(t)} = U^{(t)}\Big(X_i^{t-1}, m_{v_i}^{(t)}\Big), \tag{2.2}$$

kde $M^{(t)}(\cdot)$ je funkcia agregácie správ, $U^{(t)}(\cdot)$ je funkcia kombinácie a $m_{v_i}^{(t)}$ je agregovaná správa pre uzol v_i zostavená z jeho susedov a jeho skrytého stavu. Funkcie $M^{(t)}(\cdot)$ a $U^{(t)}(\cdot)$ obsahujú aj trénovateľné parametre. Tento proces je znázornený aj na obrázku 2.1.

Ďalšou kategóriou, zaujímavou pre nás, sú spatial-temporal grafové neurónové siete (STGNN), ktoré sa využívajú na spracovanie časovo závislých dát. Poznáme metódy založené na RNN a CNN. Ich využitie môže byť zaujímavé v prípade predpovedania využitia komunikačných zdrojov MEC.

Podľa úlohy a v záujme zredukovania dimenzionality môžeme použiť *pooling* operáciu potom, ako GNN vrstvy vytvoria reprezentáciu dát uzlov alebo hrán.

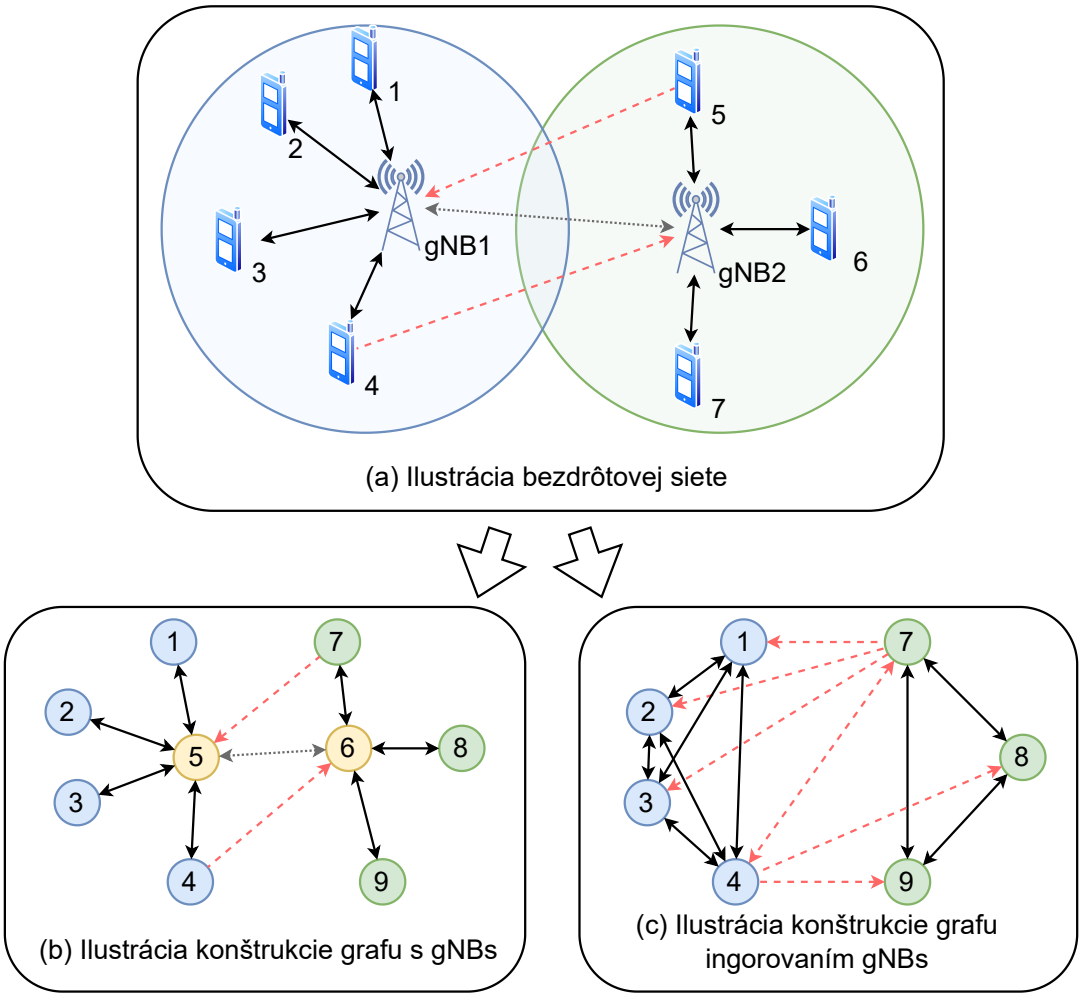


Obr. 2.1: Znázornenie MPNN vrstvy

V odbornej literatúre sa používa pojem *pooling*, keď je cieľom zredukovať dimenzionalitu dát a pojem *readout* pri úlohách na úrovni celého grafu. V súčasnosti sú operácie mean/max/sum efektívne a zároveň rýchle[18]. Pri ich použití treba brať do úvahy to, že sa nepozerá na štruktúru grafu. Aj z toho dôvodu vznikli pokročilejšie metódy zohľadňujúc štruktúru grafu, ako napr. Set2Set, SortPooling, DiffPool, SAGPool a podobne[18].

Aby sme vedeli použiť GNN na riešenie problémov v MEC, je potrebné MEC reprezentovať vhodnou formou grafu. Na obrázku 2.2 sú znázornené dve možnosti vytvorenia grafovej reprezentácie[19]. Pracujeme so scenárom, v ktorom sa na gNB môže napojiť viacero zariadení, ale zariadenia komunikujú najviac s jednou gNB. Konkrétne z obrázka zariadenia 1 až 4 komunikujú s gNB1 a zariadenia 5 až 7 komunikujú s gNB2. Zároveň zariadenie 5 interferuje s gNB1 a zariadenie 4 s gNB2. gNBs sú tiež prepojené linkou. Pri tvorbe grafu buď budeme brať do úvahy gNBs, alebo gNBs do úvahy brať nebudeme.

Obrázok 2.2b znázorňuje graf, v ktorom sme zachovali gNBs. V grafe sa nachádzajú dva typy uzlov, pre gNBs a zariadenia a tri typy hrán, pre priame spojenie alebo interferenciu medzi zariadením a gNB a hrana spájajúca gNBs. Vektor príznakov uzlov zariadení obsahuje informácie o polohe, konfigurácii a podobne. Hrany obsahujú informácie o linke alebo interferencii. Naproti tomu v obrázku 2.2c sa uzly gNBs nenachádzajú, ale vyskytujú sa tam nepriamo v hranách. Napríklad zariadenie 5 interferuje so zariadeniami 1–4. Uzly pripojené na rovnakú gNB sú navzájom poprepájané. Vzniklo tak viac hrán ako v prípade 2.2b, ale v grafe máme iba jeden typ uzlov. To, akú formu grafu zvolíme a aké informácie uložíme



Obr. 2.2: Ilustrácia konštrukcie grafu z mobilnej siete[19]

do uzlov a hrán, zavisí od problému.

2.2 Hlboké učenie posilňovaním

Problémy vznikajúce pri odosielaní úloh do MEC vrstvy sú často formulované ako nekonvexné, NP ťažké problémy. Ich optimalizácia je preto náročná. Učenie posilňovaním (z angl. Reinforcement Learning, RL) sa často využíva na dosiahnutie optimálneho rozhodovania. Dokážeme nájsť plno vedeckej literatúry, v ktorej sa použil RL na riešenie výberu BS alebo priradenie MEC zdrojov.

Ide o formu umelej inteligencie, ktorá sa učí na základe spätnej väzby v podobe odmeny. Pozostáva z piatich základných elementov:

- agent – entita vykonávajúca akcie respektíve rozhodnutia,
- prostredie – priestor, v ktorom agent vykonáva akcie,

- stav – podoba alebo popis prostredia v konkrétnom čase,
- akcia – rozhodnutie agenta,
- odmena – získaná po vykonaní akcie. Vypovedá o tom, ako dobre zvolil agent akciu. Môže byť pozitívna, ale aj negatívna.

Klasický RL sa dá použiť v situáciach, v ktorých je malý počet stavov a akcii. To sa dá vyriešiť práve použitím DRL, ktorý využíva NN na aproximáciu stavov a vykonávanie akcií.

V tejto práci budeme používať algoritmus deep deterministic policy gradient (DDPG)[20], preto si ho podrobnejšie predstavíme. Ide o DRL algoritmus, ktorý je typu herec-kritik. Názov herec-kritik pomenúva dve časti algoritmu, jedna časť vykonáva rozhodnutia (herec), teda akcie a druhá časť hodnotí vykonané akcie (kritik). Obe časti sú tvorené NN. Okrem toho DDPG používa aj takzvané target kópie siete herca a siete kritika pre výpočet target, čo slúži na stabilizovanie učenia, pretože zabraňuje agentovi robiť veľké aktualizácie váh. Ide o o off-policy algoritmus, to znamená, že politika zbierajúca vzorky je rozdielna od politiky pri aktualizácii váh. To umožňuje agentovi použiť aj staré vzorky na učenie, aj keď už je jeho politika výberu akcie iná, ako bola pri získaní vzorky. DDPG výhradne vykonáva spojité akcie čo je v našom prípade potrebné. Algoritmus je naviac deterministický, to znamená, že v tom istom stave vykonáva vždy tú istú akciu.

Sumarizácia DDPG algoritmu je znázornená pseudoalgoritmom 1. Najprv je potrebné inicializovať váhy sietí a zásobnik vzoriek. Potom nasleduje výber akcie a jej vykonanie v prostredí. Tým obdržíme nový stav a odmenu za vykonanú akciu. Získanú vzorku uložíme do zásobníka vzoriek. Ak je splnená podmienka na aktualizáciu váh, vyberieme zo zásobníka náhodné vzorky a vypočítame target hodnotu pomocou target NN. Následne prebehne update kritik NN podľa straty vypočítanej ako stredná štvorcová chyba s target a predpoveďou kritik siete. Následne sa aktualizujú váhy NN herca kde sa urobí gradientný zostup s cieľom maximalizovať hodnotu kritika. Na záver aktualizácie aktualizujeme váhy target sietí podľa váh sieti herca a kritika a podľa parametra rýchlosti učenia. Pri výbere akcie pri trénovaní je odporúčané používať šum na podporu explorácie, napríklad v podobe Gausovskej distribúcie so strednou hodnotou nula a hyperparametrom určujúci rozptyl.

Algorithm 1: DDPG pseudoalgoritmus[20]

Inicializuj parametre herca θ, parametre kritika ϕ, vyprázdni zásobník vzoriek $\mathcal{D}$

Nastav parametre $target$ sietí $\theta_{targ} \leftarrow \theta, \phi_{targ} \leftarrow \phi$,

repeat

Pozoruj stav s a vyber akciu $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, kde $\epsilon \sim \mathcal{N}$

Vykonaj akciu a v prostredí

Pozoruj ďalší stav s', odmenu r, a ukončujúci signál d, ktorý indikuje či je s' konečný stav

Ulož (s, a, r, s', d) do zásobníka $\mathcal{D}$

Ak je s' konečným stavom, resetuj stav prostredia

if *je čas na aktualizáciu* **then**

for *podľa počtu epoch* **do**

Náhodne vyber dávku vzoriek, $B = \{(s, a, r, s', d)\}$ z $\mathcal{D}$

Vypočítaj *target* hodnoty

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{targ}}(s', \mu_{\phi_{targ}}(s')) \tag{2.3}$$

Aktualizuj sieť kritika o jeden krok gradientného zostupu

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$$

Aktualizuj politiku herca o jeden krok gradientného stúpania

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta)$$

Aktualizuj *target* siete podľa

$$\phi_{targ} \leftarrow \rho\phi_{targ} + (1 - \rho)\phi$$
$$\theta_{targ} \leftarrow \rho\theta_{targ} + (1 - \rho)\theta$$

until *agent skonverguje*;

3 Uvažovaný MEC model

V tejto kapitole si popíšeme MEC model, ktorý použijeme pri vypracovaní riešení. V rámci neho zahrnieme popis prostredia, komunikácie, výpočtu na CPU a latencie. Použitá notácia sa nachádza v prílohe A.

Model obsahuje N_{gNB} gNBs, kde súčasťou každej gNB je aj MEC server, a preto budeme uvažovať o každom gNB, ako o entite schopnej výpočtu. V modeli sa nachádza N_{CAV} áut, ktoré odosielajú úlohy na spracovanie. V modeli striktne uvažujeme iba o možnosti výpočtu úlohy na gNB, ku ktorej je príslušné auto pripojené a každú úlohu odosielame do MEC, teda nebudeme sa zaoberať rozhodnutím, či úlohu odoslať alebo neodoslať do MEC siete na výpočet.

Pri probléme odosielania úloh do MEC je podstatný čas spracovania úlohy. Môžeme povedať, že existujú dve požiadavky na spracovanie úlohy. Prvou je zabezpečiť, aby bol výsledok úlohy i prijatý v určitom čase $t_{\mathrm{deadline},i}$ – ide o čas, po ktorého uplynutí už výsledok úlohy nemá využitie. Druhou požiadavkou môže byť minimalizovať latenciu $t_{\mathrm{total},i}$ výpočtu úlohy. V práci sa budeme prevažne sústrediť na úspešný výpočet úlohy v definovanom čase a nebude nás až tak zaujímať minimalizovanie latencie.

Pre čo najrýchlejší výpočet je nutné použiť väčšie množstvo zdrojov. Ak nás zaujíma len to, či latencia úlohy i bola menšia ako $t_{\mathrm{deadline},i}$ sa budeme snažiť reálny čas spracovania úlohy čo najviac priblížiť času $t_{\mathrm{deadline},i}$, ale nepresiahnuť ho. V princípe si môžeme zadefinovať ϵ_i, ako parameter určujúci rezervu v čase spracovania. Jeho hodnota môže byť od 0 po 1. Časová rezerva je potrebná, pretože môže dôjsť ku odchýlke predpovede polohy auta, či ku zmene prostredia, čo môže spôsobiť zhoršenie kvality signálu. Teda potom čas výpočtu, ktorý budeme chcieť dodržať, a podľa ktorého rezervujeme zdroje MEC, je vyjadrený:

$$t_{\mathrm{soft\_deadline},i} = t_{\mathrm{deadline},i} \cdot (1 - \epsilon_i) \tag{3.1}$$

Ako sme uviedli v kapitole 1.2, čas T_{ot} sa skladá z troch časov. To znamená, že v princípe nám treba rozdeliť čas T_{ot} na tri časti a následne vypočítať potrebné zdroje na dodržanie výsledných časov. Pre zjednodušenie môžeme predpokladať, že $T_o = T_v$, potom nám stačí rozdeliť T_{ot} len na dve časti alebo alternatívne,

ako sa to robí aj vo vedeckých článkoch, je zanedbanie času odosielania výsledku T_v z dôvodu, že väčšinou je veľkosť výsledku veľmi nízka s porovnaním s veľkosťou odosielanej úlohy a rýchlosť sťahovania je vyššia ako pri nahrávaní. Keď sa rozhodneme o zanedbanie T_v, potom čas spracovania úlohy i je:

$$t_{\text{total},i} = t_{\text{uplink},i} + t_{\text{process},i}. \tag{3.2}$$

Alokácia zdrojov MEC sa bude odrážať podľa dostupného času na výpočet a komunikáciu. To znamená, že čas $t_{\text{soft\_deadline},i}$ sme si rozdelili na dve časti, na komunikačnú (v našom prípade len nahrávanie) a výpočtovú nasledovne:

$$\begin{aligned} t_{\text{soft\_uplink},i} &= \omega_i \cdot t_{\text{soft\_deadline},i}, \\ t_{\text{soft\_process},i} &= (1-\omega_i) \cdot t_{\text{soft\_deadline},i}, \end{aligned} \tag{3.3}$$

kde ω_i je parameter, ktorý určuje pomer rozdelenia.

V situácii, v ktorej budú komunikačné zdroje pridelené CAV, budeme používať nasledujúci vzorec na výpočet rýchlosti prenosu dát medzi CAV u a gNB s v čase t:

$$R_{u,s}(t) = \beta_u(t)\rho_u(t)f_{\text{s}}(t)n_{\text{RB},u,s}(t), \tag{3.4}$$

kde β_u je počet bitov na symbol, ρ_u je kódová rýchlosť, f_{s} je modulačná rýchlosť a $n_{\text{RB},u,s}(t)$ vyjadruje počet zdrojových blokov alokovaných s-tou gNB pre u-té CAV.

Hodnoty $\beta_u(t)$ a $\rho_u(t)$ sa dajú vypočítať podľa nameranej SINR hodnoty. Použijeme na to tabuľku[1] určujúcu kvalitu spojenia, v ktorej sú mapované hodnoty $\beta_u(t)$ a $\rho_u(t)$ na hodnotu SINR. Hodnota $f_{\text{s}}(t)$, respektíve modulačná rýchlosť predstavuje počet symbolov prenesených za sekundu. Zo sekcie 1.3.2 vieme, že zdrojový blok obsahuje 7×12 zdrojových elementov v čase 0,5 ms. Pri prepočte na jednu sekundu je potom $7 \times 12 \times 2000$ dostupných zdrojových elementov v rámci jedného zdrojového bloku.

Vzorec 3.4 si upravíme tak, aby sme dostali potrebný počet zdrojových blokov $n_{\text{RB},u,s}$ pre dosiahnutie požadovanej rýchlosti prenosu:

$$n_{\text{RB},u,s}(t) = \left\lceil \frac{R_{u,s}(t)}{\beta_u(t)\rho_u(t)f_{\text{s}}(t)} \right\rceil. \tag{3.5}$$

Potrebujeme to preto, lebo čas $t_{\text{soft\_uplink},i}$ je známy a pomocou neho vieme vypočítať požadovanú rýchlosť prenosu nasledovne:

$$R_{u,s}(t) = D_i / t_{\text{soft\_uplink},i}(t), \tag{3.6}$$

[1] Tabuľka je dostupná v [21].

kde D_i je veľkosť úlohy v megabajtoch pri nahrávaní.

Kapacitu CPU zdrojov MEC budeme vyjadrovať v inštrukciách za sekundu (IPS). Potrebné zdroje pre výpočet úlohy na CPU podľa času $t_{\text{soft\_process},i}$ vypočítame nasledovne:

$$I_{\text{CPU},u,s} = I_i / t_{\text{soft\_process},i}, \tag{3.7}$$

kde I_i je počet inštrukcii potrebných pre výpočet i-tej úlohy na CPU.

4 Doterajšie riešenia v oblasti MEC

Vďaka technologickému pokroku v oblasti bezdrôtovej komunikácie sa začína rátať s využitím MEC v oblasti dopravy. Auta začínajú obsahovať viac a viac senzorov, radarov a kamier, obzvlášť v prípade autonómnych áut. Uvádza sa, že v budúcnosti bude jedno autonómne auto produkovať od 3 Gbit/s až do 40 Gbit. Takéto množstvo dát bude potrebné spracovať, pričom určitá časť z týchto dát bude kritická na rýchle spracovanie. Môže ísť o algoritmy umelej inteligencie ako napríklad hlasové spracovanie alebo spracovanie výstupu z kamier. Tieto aplikácie vyžadujú vysoký výpočtový výkon a môžu byť kritické pre bezpečné fungovanie autonómneho auta. MEC sa javí ako dobrý prostriedok na spracovanie takých dát, pretože servery nasadené v blízkosti ciest môžu slúžiť na odosielanie úloh na spracovanie. Využitie MEC siete v oblasti dopravy sa po angl. nazýva Vehicular Edge Computing (VEC). V tejto kapitole je opísaný stav výskumu v oblasti spracovania úloh v MEC systéme. Dve riešenia budú implementované a použité na porovnanie s nami vytvoreným algoritmom.

4.1 Spracovanie úloh

Autori v [22] prišli s návrhom virtuálnej hrany (z angl. virtual edge). Ide o prístup využitia voľných výpočtových zdrojov vozidiel na vytvorenie siete na možnosť odosielania úloh na výpočet. To môže byť užitočné najmä vtedy, ak nie je dostatok zdrojov dostupných vo VEC sieti alebo sa vozidlo nachádza príliš ďaleko od bázovej stanice. Keďže sieť virtuálnej hrany tvoria iné vozidlá, je tento prístup využiteľný prevažne v hustej premávke, teda v mestách, pričom sieť môže byť tvorená aj zo zaparkovaných vozidiel. Najdôležitejšou úlohou bolo vymyslieť algoritmus na vytvorenie virtuálnej hrany. V sieti virtuálnej hrany existuje práve jeden *master* uzol, vozidlo, ktoré odosiela úlohy na výpočet *slave* vozidlám vo virtuálnej hrane. *Master* ma za úlohu aj vytvárať, aj terminovať virtuálnu hranu. Pri návrhu algoritmu autori brali do úvahy a analyzovali iba výpočtove zdroje, nie komunikačné. To sa javí ako veľký nedostatok algoritmu.

Na druhej strane v [23] autori nebrali do úvahy výpočtove zdroje pri rozhodovaní o odosielania úlohy na výpočet do VEC siete a jej následnej analýze. Problém prideľovania komunikačných zdrojov vyriešili v podobe prerozdeľovania zdrojových blokov medzi vozidlá podľa požiadaviek, čím zabezpečili efektívne využívanie komunikačných zdrojov.

V [24] sa autori zaoberajú využitím zaparkovaných aút, z ktorých sa vytvorí zhluk vhodný pre spracovanie úloh, obzvlášť v prípade, kedy MEC server očakáva veľa požiadaviek. Zároveň navrhli ekonomicky model odmeňovania za využitie zdrojov zaparkovaného auta na základe teórie kontraktu a prospektu.

V tejto oblasti existujú aj výskumy zaoberajúce sa rozmiestnením VEC infraštruktúry ako napríklad v [25]. Cieľom autorov je rozmiestniť bázove stanice a servery tak, aby boli dosiahnuté isté požiadavky – pokrytie, dodržaný rozpočet, dodržaná trojvrstvová architektúra siete a pod.

Na riešenie spracovania úloh sa značne využívajú aj DRL algoritmy. V [26] autori spojito riešili problém rozhodnutia, či úlohu odoslať na MEC vrstvu na spracovanie, alebo nie a alokáciu výpočtových zdrojov. Ich MEC model zahŕňal jednú gNB a viacero používateľov. Použili Deep Q-learning algoritmus, ktorého výstup bola akcia pre každého jedného používateľa. Agenta dokázali naučiť, avšak vnímame tu problém z pohľadu škálovateľnosti, pretože popis stavu zahŕňa kriteriálnu funkciu všetkých áut. Bolo by potrebné mať viacero agentov pre rôzny počet používateľov a komunikačné zdroje nie sú prideľované dynamicky, ale sú pridelene rovnomerne každému pripojenému používateľovi.

Dynamické prideľovanie MEC zdrojov s cieľom optimalizovať latenciu a spotrebu energie je riešené v [27]. Autori prideľujú komunikačné aj CPU zdroje ak nastane rozhodnutie odoslať úlohu do MEC siete. Ich riešenie a definícia problému je dosť podobná naším zámerom. Stav obsahuje parametre úlohy vygenerované autami počas definovaného časového úseku, voľné CPU zdroje, ktoré vie priradiť auto, voľné CPU a komunikačné zdroje bázových staníc a polohy áut. Akcia obsahuje rozhodnutie o odoslaní úlohy do MEC, priradenie CPU a komunikačných zdrojov. Za algoritmus zvolili TD3, čo je rozšírenie algoritmu DDPG. Rozhodli sa tak z dôvodu, že TD3 aplikuje techniky na zlepšenie priebehu učenia oproti DDPG. Vo výsledkoch porovnávali algoritmy DQN, DDPG a TD3 kde najlepšie výsledky dosahoval TD3 algoritmus. Avšak, ako aj v predchádzajúcom riešení, aj v tomto prípade je stavový aj akčný priestor definovaný fixne podľa počtu áut v prostredí.

Multiagentove učenie posilňovaním je použité v [28] na optimalizáciu rozhodnutia odoslania úlohy do MEC, pričom uvažujú aj o preposlaní úlohy su-

sednú gNB. Autori neriešia dynamicke pridelovanie zdrojov. Agent je v tomto prípade reprezentovaný gNB. Tým, že gNB môže vykonať akciu odoslania úlohy na susednú gNB, ovplyvňuje akcie susednej gNB a práve z toho dôvodu bolo zvolené multiagentové učenie, konkrétne MADDPG. Pri MADDPG je centralizované iba trénovanie, ale vykonávanie akcii je decentralizované. Pri trénovaní má kritik k dispozícii pozorovania a vykonané akcie ostatných agentov, vďaka čomu môže lepšie ohodnotiť vykonané akcie, a tak zrýchliť proces učenia. Ale opäť sú v tomto scenári akcie aj stavy viazané na istý počet vozidiel.

Podobne aj v ďalších prácach autori uvažujú o fixnom počte akcii podľa počtu áut a takisto o stavovom priestore, ktorý rastie úmerne so zvyšujúcim sa počtom áut. To podľa nášho názoru zabraňuje použitiu algoritmu v reálnych prípadoch, kedy sa počet používateľov respektíve áut rýchlo mení. Všetky spomenuté výskumy poukázali na svoju efektivitu a aj na reálne využitie MEC. Avšak ich riešenia nepoukázali na možnosť výberu vhodnej trasy vozidiel s cieľom zvýšiť úspešnosť výpočtu ich úloh. Mobilita áut je v uvedených výskumoch daná alebo sa nezohľadňuje do veľkej miery.

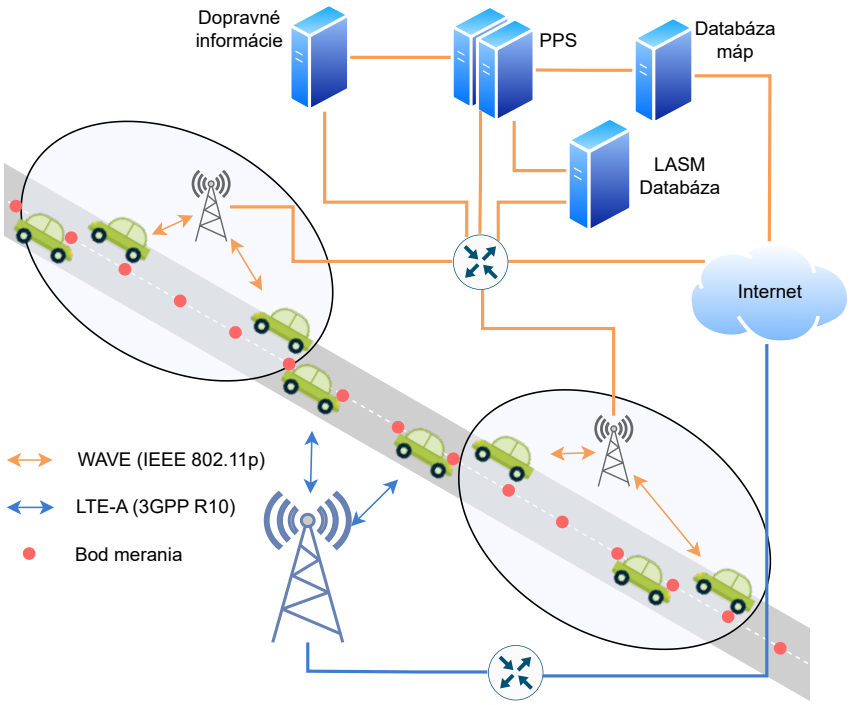
4.2 Plánovanie trasy vozidiel

V ďalšej sekcii sa pozrieme na riešenia, ktoré pracujú s mobilitou a plánovaním cesty. V [29] autori berú do úvahy scenár využívajúc satelity na nízkej obežnej dráhe na pokrytie vozidiel. Plánovanie cesty vozidla vyriešili pomocou Dijkstrovho algoritmu na nájdenie najkratšej cesty spolu s algoritmom optimalizácie mravčej kolónie, ktorý berie spojito do úvahy rozpoloženie satelitov. V práci [30] sa aplikuje hybridný prístup k automatizovanému plánovaniu cesty autonómnych vozidiel. V tomto prípade je cieľom znížiť množstvo zápch na cestách, čo stále môže viesť ku situácií s nedostatkom zdrojov vo VEC pre vozidlá.

Práca autorov v [31] spojito berie do úvahy plánovanie trasy vozidla s informáciou o semaforoch na križovatkách. Problém riešia aplikovaním evolučnej teórie hier. Algoritmus však nerieši problém odosielania úloh na spracovanie, teda nekontroluje dostupné zdroje na bázových staniciach.

Plánovanie cesty s ohľadom na vyťaženosť VEC zdrojov sa rieši v [32]. Autori navrhli dva spolupracujúce algoritmy, jeden plánujúci cestu spolu s rozdelením záťaže na VEC systém a druhý na výber vhodnej RSU a alokáciu zdrojov. Prvý algoritmus využíva algoritmus smerovania protitlaku, zatiaľ čo druhý je postavený na multi-agentovom modely teórii hier.

Nasledujúce dve riešenia budú popísané do detailov, pretože ich použijeme na



Obr. 4.1: Systémová architektúra SQRSA [33]

porovnanie s nami navrhnutým algoritmom. Autori v [33] navrhli prístup podľa kvality SINR. Ich systémová architektúra je zobrazená na obrázku 4.1. Cesta obsahuje meracie body, v ktorých sa meria hodnota SINR. Táto hodnota nameraná vozidlom je odoslaná ako dopravná informácia a uložená do databázy dlhodobého priemeru SINR mapy (z angl. long-term average SINR map, LASM). Takto auta postupne tvoria a aktualizujú SINR hodnoty jednotlivých bodov merania. Rozpoloženie týchto meracích bodov je závislé od prostredia a charakteristiky bázových staníc. Ďalším komponentom je databáza mapy potrebná na účely plánovania cesty spolu s komponentom dopravných informácii. Servery plánovania trasy na obrázku označené ako PSS používajú informácie z ostatných komponentov na výber optimálnej trasy.

Algoritmus výberu cesty prebieha následovne. Najprv vozidlo odošle pozíciu štartovacej a cieľovej destinácie na PSS. Minimálny čas označený ako t_{CD_min} je vypočítaný na základe mapy a informáciach o doprave a odoslaný naspäť vozidlu. Používateľ si vyberie čas t_{TPJ}, ktorý určuje maximálne predĺženie cesty. Na základe t_{TPJ} sa vypočíta celkový maximálny čas cesty $t_{CD\_MAX} = t_{CD\_MIN} + t_{TPJ}$. PSS potom vytvorí množinu Ω_{CD_MAX} možných ciest, ktorých čas nepresahuje t_{CD_MAX}.

$$\Omega_{CD\_MAX} = \{P_k | t_{p_k} < t_{CD\_MAX}\}, \tag{4.1}$$

kde t_{p_k} predstavuje čas cesty P_k. Každá cesta P_k z množiny Ω_{CD_MAX} reprezentuje

množinu SINR meracích bodov označených ako $\tau(x_m, y_m)$:

$$P_k = \{\tau(x_1, y_1), ..., \tau(x_m, y_m), ..., \tau(x_z, y_z)\}, \tag{4.2}$$

kde (x_m, y_m) sú koordinácie bodu merania m a z predstavuje počet meracích bodov cesty P_k. Každý bod $\tau(x_m, y_m)$ predstavuje priemernú SINR hodnotu získanu z LASM databázy. Následne je vytvorená množina E_k nasledovne:

$$E_k = \{\tau(x_m, y_m) \in P_k \in \Omega_{CD\_MAX} | \tau_{MIN} > \tau(x_m, y_m)\}, \tag{4.3}$$

kde τ_{MIN} predstavuje minimálnu akceptovateľnú hodnotu SINR. Teda množina E_k obsahuje body cesty P_k, ktoré neposkytujú dostatočnú kvalitu signálu. Nakoniec je vybratá cesta obsahujúca najmenší počet prvkov v množine E_k ako najlepšia cesta.

Riešenie berie do úvahy kvalitu signálu v podobe SINR hodnoty, ale informácia o tom, ako veľmi je bázová stanica vyťažená, sa v SINR hodnote nenachádza. Z hľadiska odosielania úlohy na výpočet je potrebné zabezpečiť dostatočné komunikačné a výpočtové zdroje. Dostupnosť zdrojov bázovej stanici môže byť obmedzená, o čom uvedený algoritmus nemá informáciu, a to môže viesť ku nasmerovaniu auta horšou trasou. Aj keď SINR hodnoty sú lepšie oproti trase, ktorá síce obsahuje menšie množstvo vyhovujúcich SINR meraných bodov, ale v konečnom dôsledku bázové stanice na tejto alternatívnej trase nemusia byť vôbec vyťažené. Algoritmus má navyše nastavenú len hranicu, ktorá určuje, či je alebo nie je SINR hodnota dostačujúca. Nepozerá sa na to, ako dobrý je daný signál. Cesta obsahujúca SINR hodnoty tesne nad hranicou akceptácie je z pohľadu algoritmu tak isto vhodná ako cesta, ktorá obsahuje oveľa vyššie SINR hodnoty.

Autori v [34] čiastočne vyriešili spomenuté nedostatky SQRSA navrhnutím nového algoritmu VaRSA. Systémová architektúra sa oproti SQRSA zmenila iba trochu. PPS komponent sa zmenil na entitu centralizovaného výberu cesty (z angl. centralized route selection entity, CRSE). LASM databáza obsahuje aj predikcie dostupnej kapacity na daných bázových staniciach. Kapacitu vo VaRSA algoritme chápeme ako rýchlosť prenosu dát, ktorú počítame zo SINR hodnoty podľa vzorca 3.4.

Predikcia kapacity hrá kľúčovú úlohu pri výbere najlepšej cesty. LASM databáza si ukladá priemerné SINR hodnoty najsilnejšieho RSU využívajúce WAVE[1], označené ako γ_i^W a pre najsilnejšej bázovej stanice využívajúce LTE, označené ako γ_i^L. Požadovaná kapacita vozidla v pre WAVE sieť je označená ako c_v^W. Dostupná

[1]WAVE (z angl. Wireless Access in Vehicular Environments) je bezdrôtová sieť pre dopravné prostredie [35].

kapacita x-tého RSU v čase t je označená ako:

$$c_x^W(t) = \widehat{C}_x^W - \sum_{v \in U} c_v^W(t), \tag{4.4}$$

kde $\widehat{C}_x^W$ je maximálna kapacita, ktorú dokáže x-tá RSU poskytnúť, U je množina vozidiel pripojených na x-tú RSU a $c_v^W(t)$ je požadovaná kapacita vozidla v v čase t.

Analogicky pre LTE sieť je dostupná kapacita v čase t vyjadrená nasledovne:

$$c_x^L(t) = \widehat{C}_x^L - \left(\sum_{v \in V} c_v^L(t) \right) - \Psi(t), \tag{4.5}$$

kde $\widehat{C}_x^L$ je maximálna kapacita, ktorú dokáže x-tá bázová stanica poskytnúť, V je množina vozidiel pripojených na x-tú bázovú stanicu, $c_v^L(t)$ je požadovaná kapacita vozidla v v čase t a $\Psi(t)$ predstavuje kapacitu potrebnú pre používateľoch nevyužívajúcich vozidlá v čase t.

Priebeh algoritmu je zo začiatku rovnaký ako v prípade SQRSA. Používateľ si vyberie destináciu, vypočíta sa najkratšia cesta a na základe jeho preferencie sa vytvorí maximálne tolerovaný čas cesty. Potom CRSE dostane všetky možné cesty zodpovedajúce vybranému časovému intervalu. Každá cesta sa podobne skladá z meracích bodov. Avšak, VaRSA sa nebude pozerať na počet vyhovujúcich alebo nevyhovujúcich bodov, ale bude brať do úvahy čas strávený v meracom bode. Čas strávený na ceste j v meracom bode i je označený ako $T_{j,i}$. Celkový čas cesty j sa dá vyjadriť ako

$$T_j^T = \sum_{1}^{n_j} T_{j,i}, \tag{4.6}$$

kde n_j je celkový počet meracích bodov na ceste j. Následne čas strávený pripojením na WAVE na ceste j je vyjadrený pomocou požadovanej priepustnosti a voľnej kapacity ako

$$T_j^W = \sum_{1}^{n_j} T_{j,i}, \{T_{j,i} | c_{j,i}^W > \tau_r\}, \tag{4.7}$$

kde τ_r predstavuje požadovanej priepustnosť používateľa a $c_{j,i}^W$ je dostupná kapacita WAVE v odpovedajúcom meracom bode. Analogicky je čas pre LTE vyjadrený ako

$$T_j^L = \sum_{1}^{n_j} T_{j,i}, \{T_{j,i} | c_{j,i}^L > \tau_r\}, \tag{4.8}$$

kde $c_{j,i}^L$ je dostupná kapacita LTE v odpovedajúcom meracom bode. Následne autori zadefinovali čas strávený autom v nenaplnení požiadaviek auta

$$T_j^O = T_j^T - (T_j^W + T_j^L). \tag{4.9}$$

Z uvedených časov autori vytvorili kriteriálnu funkciu, podľa ktorej sa vyberie najlepšia trasa

$$F_j = \Omega(T_j^T) + \Phi(T_j^T - T_j^W) + \Theta(T_j^O), \tag{4.10}$$

kde $\Omega \in \langle 0, 1\rangle$ reprezentuje váhu preferencie najrýchlejšej cesty, $\Phi \in \langle 0, 1\rangle$ reprezentuje váhu preferencie WAVE spojenia a $\Theta \in \langle 0, 1\rangle$ reprezentuje váhu vyhýbania sa zónam bez pokrytia požiadaviek vozidla. Iba jedna z váh musí mať hodnotu 1, ostatné musia mať 0, teda nie je možne kombinovať váhy. Ak používateľ chce byť co najrýchlejšie na mieste destinácie, váha $\Omega = 1$. Ak chce využiť čo najviac WAVE sieť, váha $\Phi = 1$. Ak chce zabezpečiť čo najlepšiu kvalitu spojenia a splnenie požiadaviek vozidla, váha $\Theta = 1$. Nakoniec je zvolená cesta s najnižšou hodnotou F_j.

Autori vytvorili algoritmus, ktorý podľa preferencie používateľa môže vyberať tri druhy ciest. My sa sústredíme hlavne na vyber cesty s najlepším splnením požiadaviek vozidla. V tomto algoritme už sú zachytené komunikačné zdroje, ale v prípade odosielania úlohy na MEC servery sa môže stať, že príslušný server bude preťažený, a to algoritmus VaRSA stále nezohľadňuje.

4.3 Riešenia využívajúce grafové neurónové siete

Existuje viacero výskumov s využitím GNN v oblasti bezdrôtových sieti, ale len malá časť z nich sa týka MEC. Autori v [36] riešili problém rozhodnutia, či odosielať úlohu na MEC, alebo ju vypočítať lokálne s cieľom dosiahnuť čo najnižšiu latenciu. Na riešenie využili GNN spolu s DRL, konkrétne použili herec-kritik DRL algoritmus. Navrhli rámec, v ktorom MEC pretransformovali do grafovej podoby ako v obrázku 2.2b a pomocou GNN získali vektorovú reprezentáciu vrcholov. Z reprezentácie vrcholov následne zostavili vektorovú reprezentáciu hrán spájajúcich zariadenie a gNB tak, že skombinovali vektorové reprezentácie oboch vrcholov hrany. Následne agent určil pre každú hranu pravdepodobnosť odoslania úlohy do MEC. Vznikne tak jeden súbor akcií pre daný stav. Tu však vstupuje ďalší algoritmus navrhnutý autormi, ktorý z akcií vygeneruje ďalšie akcie. Vznikne viacero scenárov (súborov akcií) na odosielanie úlohy v MEC sieti. Všetky scenáre sú následne ohodnotené a vyberie sa ten s najlepšou hodnotou. Zároveň kritik vybratú akciu ohodnotí a uloží vzorku do pamäte na učenie.

Ďalší zaujímavý prístup priniesli autori v [37]. Nejedná sa o aplikáciu v MEC, ale o prideľovanie spektra. Autori modelujú situáciu, v ktorej sa vyskytuje D2D komunikácia medzi vozidlami, ale aj komunikácia s gNB. Cieľom je prerozdeliť spektrum tak, aby bola dosiahnutá čo najvyššia kapacita prenosu medzi vozid-

lami. Problém riešili opäť s využitím DRL s GNN. Graf bezdrôtovej siete modeloval interferenciu podobne ako na obrázku 2.2c s tým, že uzol reprezentoval D2D pár. Pomocou GNN získali vektorovú reprezentáciu uzlov. Následne sa pre každý D2D pár vytiahol vektor prislúchajúceho uzla a spojil sa s dodatočnou reprezentáciou toho páru. Tento vektor bol stavom daného D2D páru pre agenta. Na základe toho stavu agent vybral akciu, ktorá hovorila o pridelení spektra danému páru.

Existujú aj výskumy, v ktorých je trénovanie GNN robené pomocou učenia s učiteľom. Napríklad autori v [38] navrhli rámec pre prideľovanie zdrojov. Keďže učenie s učiteľom si vyžaduje vzorky na natrénovanie, autori museli využiť generátor vzoriek odvodený zo suboptimálneho *constraint cross-entropy* algoritmu.

Na trénovanie sa dá využiť aj čiastočné učenie s učiteľom (z angl. Semi-Supervised Learning), ako to využili aj autori v [39]. Riešili prideľovanie výkonu a výber gNB v husto prepojených sieťach, teda jedno zariadenie môže byť pripojené na viacero gNB. Pri čiastočnom učení s učiteľom obsahujú niektoré vzorky pridelenie, v tomto prípade niektoré zariadenia mali predom dané pripojenie s gNB.

Súhrn prác v [19] využívajúcich GNN v bezdrôtových sieťach naznačuje vysoký počet prác v problémoch kontroly výkonu, prideľovanie kanálu alebo prideľovanie spektra, ale ako som už spomenul, veľmi málo prác sa dotýka prideľovania výpočtových a komunikačných zdrojov. Preto si myslíme, že práve tu je priestor na prínos v podobe využitia GNN a DRL.

4.4 Zhrnutie

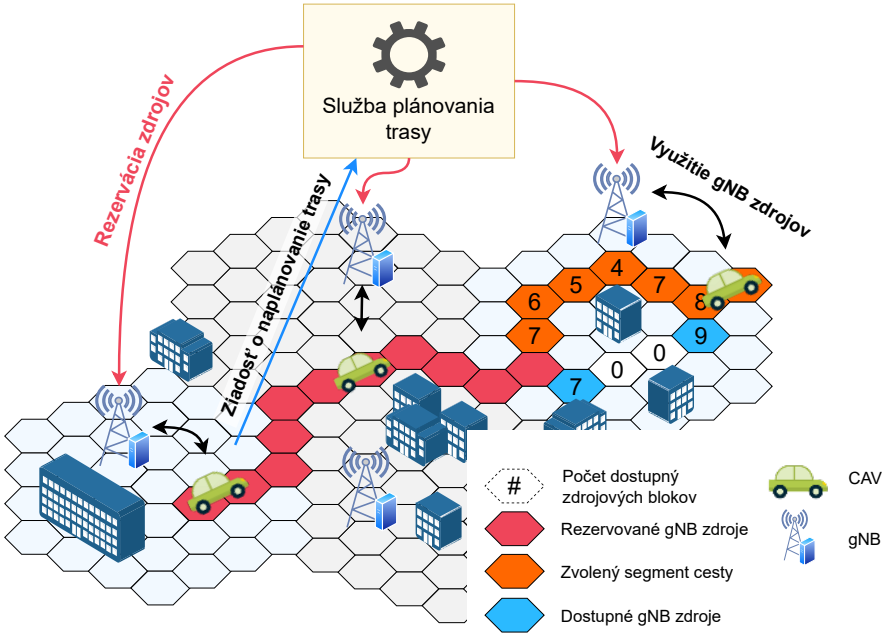
Spracovanie úloh v MEC stále podlieha výskumu. Ako sme si ukázali, existuje veľké množstvo prístupov riešenia tohto problému. Doménu sme si zúžili na využitie MEC v oblasti dopravy a v tejto oblasti sme poukázali na nedostatok v podobe plánovania trasy s cieľom dosiahnuť čo najlepšie využitie komunikačných a výpočtových zdrojov. Jedným cieľom je navrhnúť algoritmus, ktorý by spojito bral do úvahy komunikačné a výpočtové zdroje prí plánovaní trasy ako aj dĺžku trasy. Sľubným scenárom sa javí navrhnutie plánovania trasy pre autonómne autá, ktoré si už zároveň rezervujú zdroje MEC pri plánovaní trasy dopredu.

Zároveň sme urobili analýzu GNN, relatívne novej oblasti, ktorá sa javí ako veľmi vhodná na využitie v MEC, pretože MEC vieme vhodne pretransformovať do grafu. Nenašli sme doposiaľ žiadny prístup využitia GNN na alokáciu výpočtových aj komunikačných zdrojov pre pohybujúcich sa používateľov.

5 Navrhnuté riešenia

Prvé riešenie sa zaoberá plánovaním trasy CAVs s ohľadom na MEC zdroje. V druhom riešení sa snažíme optimálne rozdeľovať zdroje MEC pre pripojené CAVs pričom je trasa vozidla daná spolu s výberom gNB na ktorú sa CAV pripája.

5.1 Plánovanie cesty s alokáciou MEC zdrojov



Obr. 5.1: Systémovy model plánovania trasy s ohľadom na dostupné MEC zdroje

Pri plánovaní trasy budeme rátať s nasledujúcim predpokladom: všetky CAVs budú autonómne, a tým predpokladáme, že ich pozície v budúcnosti nám budú známe, respektíve ich budeme môcť dostatočne presne odhadnúť. To môžeme usúdiť vďaka tomu, že do riadenia auta nebude takmer žiadnym priamym spôsobom zasahovať ľudský faktor. Vďaka tejto skutočnosti sme sa rozhodli nielen určiť cestu, ktorá je vhodná pre CAV a jeho požiadavky, ale zároveň aj rezervovať

zdroje na príslušných gNBs. To by v reálnom scenári malo zabezpečiť dostatočnú kvalitu spojenia a dostatok výpočtových zdrojov pre konkrétne CAV aj v prípade, ak sa nečakane bude chcieť spojiť s gNB veľa zariadení – rezervované zdroje sa pre iné zariadenia nebudú môcť použiť.

Ukážkový scenár plánovania trasy s alokáciou MEC zdrojov je zaznamenaný na obrázku 5.1. Na obrázku je trasa CAV zaznamenaná v za sebou idúcimi segmentami cesty. CAV si bude môcť rezervovať MEC zdroje v rámci daného segmentu. Segmenty označené červenou farbou označujú rezervovaný segment – zvolenú trasu. Číslo v segmentoch reprezentuje počet zdrojových blokov potrebných na splnenie komunikačných požiadaviek pre CAV. Hodnota nula v segmente znamená, že zdroje MEC nie sú dostatočné na splnenie požiadaviek CAV a takýto segment nie je možné zarezervovať. Chceme teda, aby segmenty boli rezervované s ohľadom na požiadavky CAV, ale aj s ohľadom na ostatné CAVs využívajúce MEC zdroje.

5.1.1 Popis problému

Prvým cieľom je možnosť definovať si maximálne predĺženie cesty z hľadiska času. Predpokladáme, že používateľ si bude chcieť určiť, ako najviac je ochotný predĺžiť si cestovanie. Pri návrhu sa čiastočne inšpirujeme doterajšími riešeniami. Vyhľadávanie trasy bude ohraničené podľa času tak, ako to bolo aj v prácach [33][34]. To znamená, že do úvahy budeme brať len trasy, ktorých reálny čas T_{P_k} bude kratší ako definovaný maximálny čas $T_{P,\max}$. Maximálny čas sa vypočíta ako

$$T_{P,\max} = T_{P,\text{fastest}} \cdot \delta_u, \tag{5.1}$$

kde $T_{P,\text{fastest}}$ je čas najkratšie trvajúcej cesty a δ_u je koeficient predĺženia cesty. Označenie P_k označuje cestu k, ktorá sa skladá zo segmentov σ. Množinu ciest spĺňajúce toto kritérium si označíme nasledovne

$$\Omega_u = \{P_k | T_{P_k} \leq T_{P,\max}\}, \tag{5.2}$$

Druhým cieľom je minimalizovať počet nespracovaných úloh, ktoré boli odoslané do MEC na spracovanie. Tento cieľ sa skladá z výberu bázovej stanice a alokácie komunikačných a výpočtových zdrojov. To môže znieť jednoducho, ale treba si uvedomiť, že výber ovplyvňuje aj možnosti rezervácie pre iné CAVs. Náročné na tom je určiť, koľko z komunikačných a koľko z výpočtových zdrojov sa má rezervovať pre dané CAV, pretože ak si CAV alokuje priveľa komunikačných zdrojov, môže sa stať, že hoci výpočtových zdrojov ostane viac, ostatným CAVs nemusia stačiť zostatkové komunikačné zdroje. Platí to aj naopak. V prípade pláno-

vania cesty sme sa rozhodli určiť ω_i rovnakú a fixnú pre všetky CAVs a samotné skúmanie rozdelenia MEC zdrojov bude rozobraté zvlášť v kapitole 5.2.

Výpočet i-tej úlohy budeme považovať za úspešný, ak platí

$$t_{\text{total},i} \leq t_{\text{deadline},i}. \tag{5.3}$$

Vzhľadom na to sme si zadefinovali metriku M_{FT} určujúcu mieru neúspešnosti výpočtu úloh v našom MEC systéme

$$M_{\text{FT}} = \frac{N_{FT}}{N_{GT}}, \tag{5.4}$$

kde N_{FT} je celkový počet úloh, ktoré presiahli latenciu, teda nespĺňajú podmienku v 5.3 a N_{GT} je celkový počet vygenerovaný úloh.

Úlohou je minimalizovať M_{FT} a zároveň minimalizovať čas vybratej cesty. Problém vieme zapísať vo forme multikriterialného optimalizáčneho problému v tvare:

$$P1 : \min_{N_{\text{CAV}},D_i,I_i,\omega_i} \left(M_{\text{FT}}, \sum_{\sigma \in P_k} |\sigma|/v \right) \tag{5.5}$$

s.t.:

$$C1.1 : \sum_u n_{\text{RB},u,s}(t) \ \leq N_{\text{RB},s},$$

$$C1.2 : \sum_u I_{\text{CPU},u,s}(t) \ \leq I_{\text{CPU},s},$$

$$C1.3 : \sum_{\sigma \in P_k} |\sigma|/v \ \leq (\sum_{\sigma \in P_f} |\sigma|/v) * \delta_u,$$

kde P_f je množina za sebou idúcich segmentov najkratšej cesty, v je rýchlosť vozidla, $N_{\text{RB},s}$ a $I_{\text{CPU},s}$ sú počet zdrojových blokov a CPU výkon v MIPS s-tej gNB a P_f je množina segmentov najkratšej trasy.

Podmienka $C1.1$ predstavuje ohraničenie priradenia zdrojových blokov pripojeným CAVs tak, aby sa nepresiahol celkový počet zdrojových blokov gNB v čase t. Podmienka $C1.2$ rovnakým spôsobom ohraničuje priradenie výpočtových zdrojov v čase t tak, aby sa nepresiahla kapacita CPU gNB. Nakoniec podmienka $C1.3$ hovorí o tom, aby sa uvažovalo iba o trasách, ktoré sú kratšie ako definované predĺženie najkratšej trasy.

5.1.2 A* algoritmus

V čase návrhu riešenia sme mali implementované algoritmy SQRSA a VaRSA a pri ich simulovaní sme si zároveň uvedomili, že sú časovo náročné. V princípe tieto algoritmy hodnotili každú možnú trasu, ktorá spĺňala časové kritérium aby mohli

určiť, ktorá z nich je najlepšia. Výber trasy je možné formulovať ako hľadanie cesty v strome, a preto sme sa rozhodli využiť algoritmus A*. Problém definovaný v 5.5 obsahuje dve kritéria rôznej priority. Vyššiu prioritu má minimalizovanie metriky M_{FT} a druhoradé je minimalizovanie času trasy. Vybrali sme si lexikografickú formu A* algoritmu, aby sme sa vyhli nastavovania kompromisu medzi kritériami tým, že prioritu kritérií definujeme predom.

Algoritmus pracuje s grafovou štruktúrou G, ktorá pozostáva z vrcholov grafu $V(G) = v_1, \ldots, v_n$ zodpovedajúce polohám na trase tvoriacich koncové body segmentov a hrán $E(G) = \{\{v_1, v_2\}, \ldots, \{v_{n-1}, v_n\}\}$ reprezentujúce v našom prípade cenu spojovanú so segmentom cesty. Algoritmus funguje na princípe expandovania vrcholov. Aby algoritmus expandoval vrcholy efektívne, je potrebné definovať kriteriálnu funkciu, ktorá odhadne celkové náklady potenciálnej cesty cez graf skombinovaním známych váh prechádzajúcich segmentov a s heuristickým odhadnutím kumulatívnej váhy ešte neobjavených segmentov potenciálnej trasy. Teda kriteriálna funkcia f predstavujúca celkovú odhadadovanú cenu potenciálnej trasy je vo vrchole v_i, cez ktorý trasa prechádza, definovaná ako:

$$f(v_i) = g(v_i) + h(v_i), \tag{5.6}$$

kde $g(v_i)$ je reálna kumulatívna cena potenciálnej trasy z vrcholu v_{orig} (počiatočný bod trasy) do vrcholu v_i a $h(v_i)$ je heuristicka funkcia, ktorá určuje odhad ceny trasy z vrcholu v_i do vrcholu v_{dest} (koncový bod trasy).

Funkcia $g(v_i)$ určuje cenu časti trasy od vrchol v_{orig} po vrchol v_i v podobe usporiadanej trojice:

$$g(v_i) : (CC_{\mathrm{missing}}, CC_{\mathrm{RB}}, D_{\mathrm{O}}), \tag{5.7}$$

kde $CC_{\mathrm{missing}} = \sum_{\sigma \in P_{k\_v_i}} \sigma_{\mathrm{missing}}$ označuje kumulatívnu cenu od v_{orig} po v_i v podobe počtu segmentov, v ktorých nie je dostatok zdrojov pre splnenie požiadaviek CAV, pričom $P_{k\_}v_i$ ozančuje podmnožinu P_k pozostávajúca len zo segmentov od v_{orig} po v_i a $\sigma_{\mathrm{missing}}$ označuje hodnotu pre segment σ, ktorá je rovná 1, ak nie je možné rezervovať MEC zdroje na splnenie výpočtových a komunikačných požiadaviek v segmente σ alebo 0 v opačnom prípade. Druhá časť usporiadanej trojice je vyjadrená $CC_{\mathrm{RB}} = \sum_{\sigma \in P_{k\_v_i}} n_{\mathrm{RB},\sigma}$ a určuje počet zdrojových blokov potrebných pre rezerváciu na splnenie požiadaviek pre odpovedajúcu časť trasy od v_{orig} po v_i, pričom $n_{\mathrm{RB},\sigma}$ vyjadruje počet zdrojových blokov potrebných na rezerváciu v segmente σ. Posledná, tretia časť usporiadanej trojice D_{O} označuje dĺžku časti trasy od v_{orig} po v_i vyjadrenú v metroch.

Hodnota CC_{missing} reprezentuje jednu z kľúčových metrík, o ktorých uvažujeme v multikriteriálnej optimalizácii, a to je dôvodom jej umiestnenia v kumu-

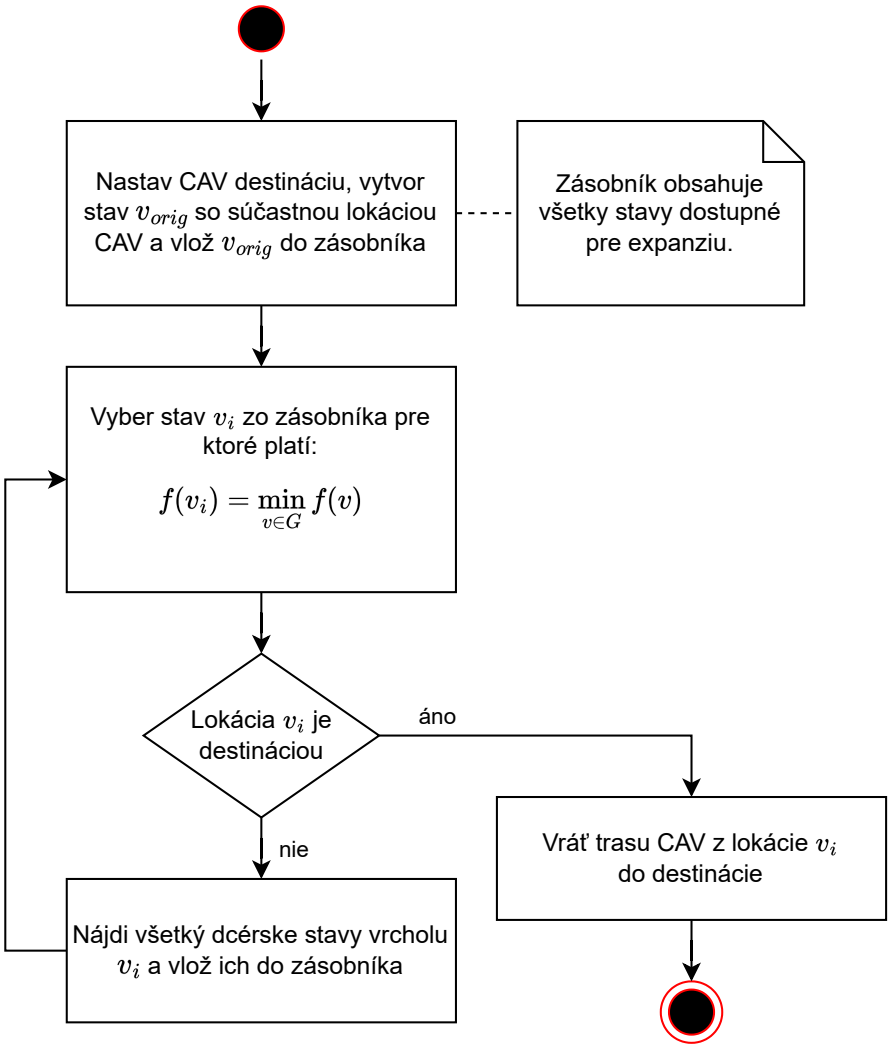
latívnej cene cesty. Dôvodom výberu a zahrnutia CC_{RB} je to, aby sa zdroje gNB využívali efektívne. V tomto prípade je vhodnejšie zvoliť si takú trasu, v ktorej je potrebné si v celkovom počte rezervovať menej zdrojových blokov gNBs na splnenie výpočtu úlohy do času $t_{\mathrm{deadline},i}$ v príslušných segmentoch trasy ako trasu, ktorá vyžaduje viac zdrojových blokov na splnenie tohto kritéria. V prostredí tak ostane viac zdrojových blokov pre ostatné CAVs. Zo začiatku sme CC_{RB} nemali zahrnuté pri výpočte kumulatívnej ceny cesty, ale dospeli sme ku tomu experimentálne, ked sa nám zdalo, že výsledky simulácií neboli podľa očakávaní. D_{O} predstavuje prejdenú dĺžku trasy a štandardne sa táto metrika využíva v A* pri vyhľadávaní najkratšej cesty.

Metriky sú lexikografický usporiadané, a to znamená, že ak majú dve trojice rovnakých n hodnôt zľava, potom hodnota na pozícii $(n + 1)$ zľava určuje, ktorá trojica je väčšia alebo menšia. Zároveň toto pravidlo určuje aj priorite jednotlivých metrík v usporiadanej trojici, v ktorej priorita metrík klesá od ľavej strany smerom doprava. Je logické, že CC_{missing} sa nachádza na prvom mieste zľava usporiadanej trojice. Naším prvoradým cieľom je zabezpečiť čo najviac vypočítaných úloh, a teda vyberať trasu s najnižším počtom CC_{missing}. Pri porovnaní trojíc v prípade rovnakého počtu CC_{missing} sa následne pozeráme na hodnotu CC_{RB}, kde nám ide o rezervovanie čo najnižšieho množstva zdrojových blokov. Až nakoniec sa pozeráme na dĺžku trasy D_{O} v prípade, ak je aj hodnota CC_{missing} rovnaká. Dĺžka trasy tu nie je až taká podstatná, pretože už je ohraničená podmienkou $C1.3$ optimalizačného problému.

Na výpočet kriteriálnej funkcie nám ešte chýba definovať heuristicku funkciu $h(\cdot)$. Obyčajne sa pre A* algoritmus používa relaxácia funkcie $g(\cdot)$ respektíve sa relaxuje samotný problém. Čím viac relaxujeme problém, tým sa heuristický odhad môže vzďaľovať od $g(\cdot)$ čo smeruje k expandovaniu viacerých uzlov. Heuristickú funkciu $h(\cdot)$ sme zadefinovali nasledovne:

$$h(v_i) : (0, 0, D_{\mathrm{H}}(v_i)), \tag{5.8}$$

kde D_{H} je vzdialenosť z v_i po v_{dest}. Heuristická funkcia $h(v_i)$ použitá v A* algoritme musí byť konzistentná. Dôkaz sa nachádza v prílohe B. Hodnoty odpovedajúce CC_{missing} a CC_{RB} sú rovné 0, pretože heuristický ich odhadnúť je nemožné. Vzdialenosť je bežná forma heuristiky v A*. V našom prípade má vzdialenosť najnižšiu prioritu a fakt, že pravdepodobnosť existencie dvoch trás, ktoré vyžadujú rovnaký počet zdrojových blokov na rezerváciu pre splnenie požiadaviek CAV nie je až taká vysoká, vplýva na to, že heuristický odhad nemusí byť vždy veľmi prínosný pri hľadaní trasy.



Obr. 5.2: Diagram akcii A* algoritmu

Na obrázku 5.2 je znázornený diagram reprezentujúci proces fungovania navrhovaného A* algoritmu. V prvom kroku sa vyberie východzia poloha CAV a uloží sa do vrchola v_{orig}, ktorý sa vloží do zásobníka. Zásobník slúži na uchovanie všetkých vrcholov dostupných na expandovanie. V ďalšom kroku sa vyberie stav, respektíve vrchol zo zásobníka, ktorého hodnota kriteriálnej funkcie $f(\cdot)$ je najnižšia. Ak je vybratý stav destináciou, algoritmus vráti trasu. V opačnom prípade sa expandujú priami potomkovia vybratého stavu a vkladajú sa do zásobníka.

Opisaný postup algoritmu A* hovorí o tom, ako dostaneme trasu pre CAV, ale k tomu potrebujeme vedieť určiť v každom segmente cesty najvhodnejšiu gNB a počet zdrojových blokov potrebných na splnenie požiadaviek CAV. Algoritmus 2 znázorňuje proces výberu gNB na rezerváciu MEC zdrojov. Zdroje sa rezervujú pre úlohu i v rámci segmentu σ v časovom intervale $< t_{\text{req}}, t_{\text{rel}} >$. Alokácia výpočtových prostriedkov je jednoduchšia, potrebné výpočtové zdroje sa vypočítajú

Algorithm 2: CAV–gNB združovanie a prideľovanie rádiových a výpočtových zdrojov

Function *gNBAllocation*$(N_{\mathrm{gNB}}, u, i, t_{\mathrm{req}}, t_{\mathrm{rel}}, t_{\mathrm{process},i}, \sigma)$ **is**

```
    /* Výpočet požadovaných výpočtových zdrojov (I_CPU,u,s)      */
1   I_CPU,u,s ← podľa rovnice 3.7
    size ← D_i
    best_gNB ← None

    n_RB,u,s ← 0
    min_RB ← ∞
2   for j ∈ {1 ... N_gNB} do
        t ← t_req
        temp_RB ← podľa rovnice 3.5
        res_avail ← check_resources(gNB_j)
        is_lowest_RB ← temp_RB < min_RB
3       if res_avail and is_lowest_RB then
4           min_RB ← temp_RB
            /* Definitívny počet pridelených rádiových zdrojov
               n_RB,u,s po všetkých iteráciách cyklu              */
5           n_RB,u,s ← min_RB
6           best_gNB ← gNB_j
    return (u, best_gNB, t_req, t_rel, I_CPU,u,s, n_RB,u,s)
```

ako prvé (riadok 1) pomocou vopred určeného času na spracovanie $t_{\text{soft\_process},i}$. Algoritmus následne prechádza cez všetky dostupné gNBs (riadok 2) a snaží sa vybrať gNB, kde je dostatok výpočtových zdrojov, a ktorá je schopná naplniť komunikačné požiadavky s najnižším počtom zdrojových blokov. Do premennej res_avail sa uloží boolovská hodnota, ktorá označuje, či dané gNB má dostatok komunikačných aj výpočtových zdrojov na splnenie požiadaviek v čase t, a ktorá sa následne použije v podmienke na riadku 3. Ak gNB má dostatok zdrojov a zároveň dokáže splniť požiadavky s najnižším počtom zdrojových blokov spomedzi doteraz prejdených gNBs, odpovedajúce gNB sa uloží do best_gNB (riadok 6). Po prejdení všetkých gNBs bude táto premenná obsahovať gNB, ktorá dokáže obslúžiť CAV s najnižším počtom zdrojových blokov. Premenná min_RB obsahuje počet zdrojových blokov odpovedajúcej gNB uloženej v best_gNB (riadok 4). Algoritmus vráti odpovedajúcu gNB a CAV odpovedajúci časový rozsah priradenia, počet zdrojových blokov a výpočtových zdrojov.

Algoritmus ešte priamo nerezervuje zdroje pre gNB, tie sa iba uložia do príslušného segmentu pri plánovaní trasy. Až keď dostaneme výslednú trasu, začne sa vykonávať rezervácia zdrojov na odpovedajúcich gNBs v segmentoch ciest.

5.2 Dynamické prideľovanie MEC zdrojov

Budeme vychádzať zo sekcie 5.1, v ktorej má navrhnuté riešenie nedostatok. Na jednej strane sme dokázali spojito brať do úvahy komunikačné aj výpočtové zdroje, ale alokácia zdrojov, respektíve využitie zdrojov môže byť neoptimálne práve preto, že rozdelenie ω_i je fixné. Lepšie by bolo, ak by sa ω_i menilo podľa prostredia a práve tomu sa budeme venovať.

Scenár si upravíme, nebudeme uvažovať o plánovaní trasy, ale zameriame sa len na vhodné prerozdelenie MEC zdrojov v reálnom čase, pričom pripojenie ku gNB aj trasa CAV budú predom určené iným algoritmom.

5.2.1 Popis problému

Hlavnou prioritou stále ostáva zabezpečiť čo najlepšiu kvalitu pripojenia a výpočtu vzhľadom na požiadavky CAVs. Preto aj naďalej budeme pracovať s metrikou M_{FT}. Okrem nej si zavedieme aj metriku určujúcu priemernú latenciu v systéme vyjadrenú pomerom $t_{\text{total},i}$ ku $t_{\text{deadline},i}$ ako

$$M_{\text{L}} = \frac{1}{N_{CT}} \sum_{i \in \mathcal{T}_{comp}} \frac{t_{\text{total},i}}{t_{\text{deadline},i}}, \tag{5.9}$$

kde N_{CT} predstavuje počet vypočítaných úloh do času $t_{\text{deadline},i}$ a $\mathcal{T}_{comp}$ je množina vypočítaných úloh do času $t_{\text{deadline},i}$. Pri plánovaní trasy sme o znižovaní latencie neuvažovali. Tu budeme skúmať aj zníženie latencie, pretože vo všeobecnosti je lepšie vypočítať úlohu čo najskôr, ak je dostatok zdrojov k dispozícii. Riešenie budeme tvoriť tak, aby prioritou bola metrika M_{FT}.

Problém si opäť zadefinujeme ako multikriteriálnu optimalizáciu, v ktorej sa budeme snažiť minimalizovať metriky M_{FT} a M_{L}:

$$P2 : \min_{N_{\text{CAV}},D_i,I_i} \left(M_{\text{FT}}, M_{\text{L}} \right) \tag{5.10}$$

s.t.:

$$C2.1 : \sum_{u} n_{\text{RB},u,s}(t) \leq N_{\text{RB},s},$$

$$C2.2 : \sum_{u} I_{\text{CPU},u,s}(t) \leq I_{\text{CPU},s},$$

$$C2.3 : 0 < \omega_i < 1$$

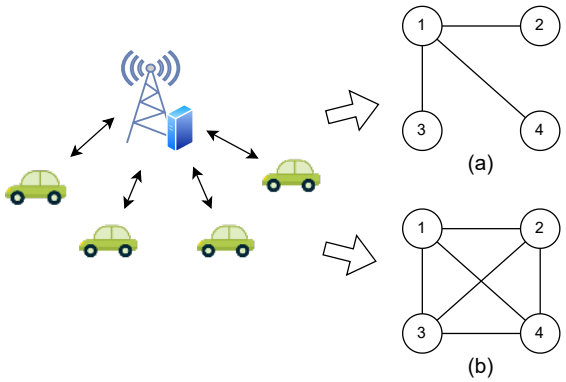
$$C2.4 : 0 < \epsilon_i < 1,$$

kde $C2.1$ definuje ohraničenie pridelenia zdrojových blokov v čase t, aby sa nepresiahol maximálny počet blokov s-tej gNB, $C2.2$ definuje ohraničenie pridelenia CPU zdrojov v čase t tak, aby sa nepresiahli dostupné CPU zdroje s-tej gNB, $C2.3$ ohraničuje nastavenie rozdelenia času ω_i pre i-tu úlohu, teda určuje to, ako sa rozdelia gNB zdroje podľa požiadaviek a $C2.4$ ohraničuje očakávanú latenciu $t_{\text{soft\_deadline},i}$, podľa ktorej sa budú vypočítavať a rozdeľovať zdroje gNB.

Zapísaný problém sa dá jednoducho prispôsobiť situáciám, kedy sa uvažuje iba o jednej z metrík. Pri minimalizácii metriky M_{L} je potrebné ponechať všetky podmienky a pri minimalizácii M_{FT} sa posledná podmienka $C2.4$ odstráni, pretože hodnota ϵ_i bude fixná.

5.2.2 Riešenie

Problém zapísaný v 5.10 sme sa rozhodli riešiť pomocou DRL a GNN. V sekcii 4 sme ukázali existujúce práce, ktoré používajú tieto prvky umelej inteligencie na riešenie veľmi podobných problémov. GNN nám bude slúžiť na vytvorenie reprezentácie prostredia tak, aby sme následne túto reprezentáciu poskytli agentovi na vykonávanie akcií v systéme. Vďaka využitiu DRL nie je potrebné umelo vytvárať dáta na natrénovanie NN a zároveň máme možnosť aktualizovať politiku agenta počas chodu algoritmu. Na to, aby sme problém riešili formou DRL, je potrebné zadefinovať stav, akcie a odmenu pre agenta.



Obr. 5.3: Možnosti vytvorenia grafu zo scenára, (a) hrany sa nachádzajú iba medzi uzlami ak je jeden z nich uzol, ktorý reprezentuje CAV, pre ktoré sa bude určovať akcia, (b) všetky uzly sú prepojené hranami

Prvou vrstvou NN bude GCN, a teda stav agenta bude definovaný formou grafu G_u. Tým, že MEC zdroje sa budu prerozdeľovať v závislosti konkrétnej gNB a CAVs pripojených na túto gNB, bude graf obsahovať iba informácie o ostatných CAVs pripojených na rovnakú gNB. Ako sme ukázali, graf v MEC kontexte sa dá vytvoriť dvojakým spôsobom. Pre jednoduchosť sme sa rozhodli pracovať s formou grafu znázorneného na obrázku 2.2c s tým rozdielom, že hrany interferencie nebudeme používať. Miesto nich priamo zahrnieme SINR hodnoty vo vrcholoch. Vznikne nám úplne homogénny graf, v ktorom bude jeden typ hrany a jeden typ uzlov, ako znázorňuje obrázok 5.3. Pracovať môžeme s dvoma verziami grafov. Stav môžeme reprezentovať tak, ako je znázornené na obrázku 5.3a, kde sú hrany iba medzi uzlom reprezentujúce CAV, pre ktoré vykonávame akciu (vrchol s označením 1) a ostatnými vrcholmi reprezentujúcimi CAVs pripojenými na rovnakú gNB. Druhou možnosťou je uplne prepojiť uzly každý s každým (obrázok 5.3b).

Spracovanie grafu GCN vrstvou, ktorú budeme používať, vyzerá nasledovne:

$$\mathbf{x}_i^{(k)} = \sum_{j \in N(i) \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(j)}} \cdot (\mathbf{W}^t \cdot \mathbf{x}_j^{k-1}) + \mathbf{b}, \tag{5.11}$$

kde $N(i)$ je množina susedov i-ého vrcholu, $\mathbf{W}$ je matica váh, $\mathbf{x}_i^{(k)}$ je vektorová reprezentácia vrchola i v k-tej vrstve, $\mathbf{b}$ je bias a zlomok vyjadruje normalizáciu. Táto GCN vrstva agreguje atribúty všetkých susedných uzlov ako aj atribúty vlastného uzla (výraz $j \in N(i) \cup \{i\}$ pri súčte). To znamená, že ak sú všetky uzly prepojené, vznikne rovnaký vektor pre všetky uzly, a teda všetky CAVs napojené na rovnakú gNB by mali rovnaký stavový vektor (lebo graf je rovnaký bez ohľadu na to, na ktoré CAV sa sústredíme). Preto plne prepojený graf nemôžme priamo použiť na reprezentáciu stavu pre konkrétne CAV. Forma grafu z obrázka 5.3a čelí

tomuto problému iba v prípade, ak je počet vrcholov grafu rovný dvom. Tento problém môžeme vyriešiť pridaním vektora stavu pre dané CAV ku výstupnému vektoru GCN vrstvy, čo bude bližšie popísané neskôr.

Teraz, keď už máme definované možnosti formy grafu, ktorý reprezentuje stav, môžeme sa posunúť o úroveň nižšie a definovať dáta vo vektore $\mathbf{x}_u$ reprezentujúci vektor u-tého CAV:

$$\mathbf{x}_u = (sinr_u, sinr_{\mathrm{old},u}, n_{\mathrm{RB},u,s}, I_{\mathrm{CPU},u,s}), \tag{5.12}$$

kde $sinr_u$ je aktuálne nameraná hodnota SINR u-tého CAV, $sinr_{\mathrm{old},u}$ je SINR hodnota nameraná v predchádzajúcom kroku, $n_{\mathrm{RB},u,s}$ a $I_{\mathrm{CPU},u,s}$ je aktuálny počet rezervovaných zdrojových blokov a aktuálna rezervovaná kapacita CPU v tomto poradí. Rozhodli sme sa nepoužiť hrany určujúce interferenciu, preto je hodnota SINR zahrnutá v $\mathbf{x}$ a zároveň tvorí najdôležitejšiu informáciu. Predchádzajúca hodnota SINR $sinr_{\mathrm{old},u}$, ktorá bola braná do úvahy pri vykonaní poslednej akcie rozdelenia MEC zdrojov, má za účel spolu s $sinr_u$ oznámiť agentovi, či sa kvalita signálu zlepšila alebo nie. Informácie o momentálnom počte zarezervovaných zdrojových blokov ako aj o rezervovaného výkonu CPU v inštrukciách dodá agentovi informáciu najmä o ostatných CAVs, s ktorými zdieľa MEC zdroje.

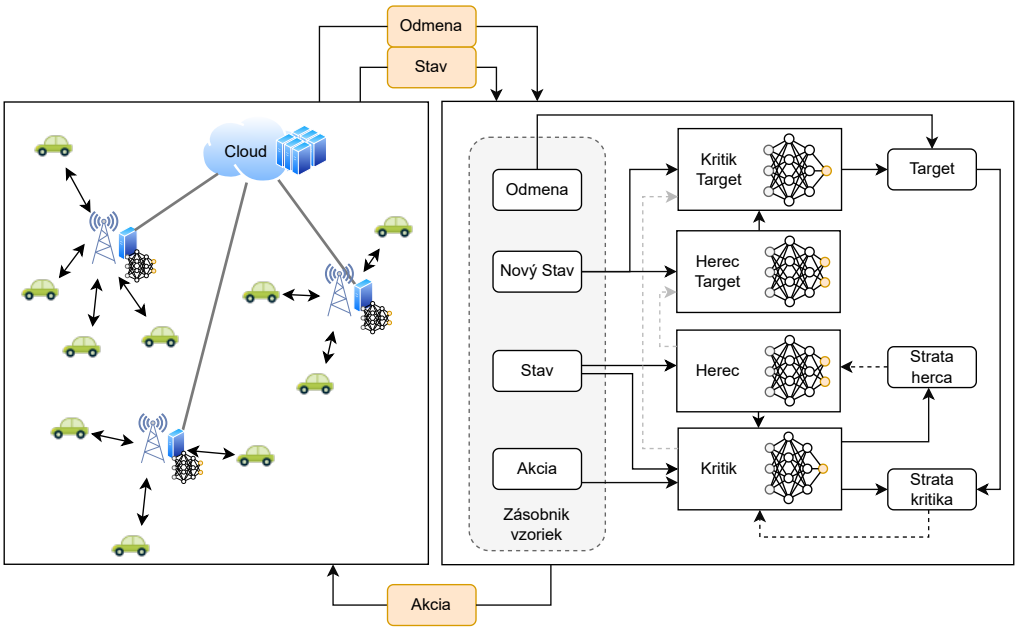
Vektor $\mathbf{x}_u$ sme normalizovali nasledujúcou formou. Hodnoty SINR normalizujeme podľa rozsahu od -7 až 23, pretože tabuľka mapovania SINR na kvalitu signálu obsahuje mapovanie v tomto rozsahu. Počet aktuálne priradených zdrojových blokov a CPU výkonu budu normalizované podľa maximálnej kapacity gNB.

Akciu agenta pre u-té CAV zapíšeme v tvare $a_u = (\epsilon, \omega)$. Agent teda bude určovať časovú rezervu, čo bude ovplyvňovať latenciu a zároveň rozdelenie komunikačných a výpočtových zdrojov gNB.

Odmena pre agenta sa bude odvíjať od definovania optimalizačného problému v 5.10. Agent bude vykonávať akcie, ktoré navzájom ovplyvňujú CAVs v kontexte jednej gNB. Preto aj odmenu budeme počítať v kontexte jednej gNB, a nie pre celé prostredie. Hlavným kritériom je minimalizovať M_{FT}, čo znamená vypočítať čo najviac úloh v definovanom čase. Zároveň je druhým cieľom minimalizovať latenciu. Berúc do úvahy tieto kritériá funkciu odmeny sme navrhli nasledovne:

$$r_u(t) = \begin{cases} N_{CT,s}(t), & \text{ak } N_{FT,s} > 0 \\ N_{CT,s}(t) \cdot (1 + L_s(t)), & \text{inak} \end{cases} \tag{5.13}$$

kde $N_{CT,s}(t)$ je počet všetkých vypočítaných úloh CAVs pripojených na s-tú gNB v časovom okne t a $L_s(t)$ vyjadruje priemerný podiel času výpočtu úlohy ku ča-



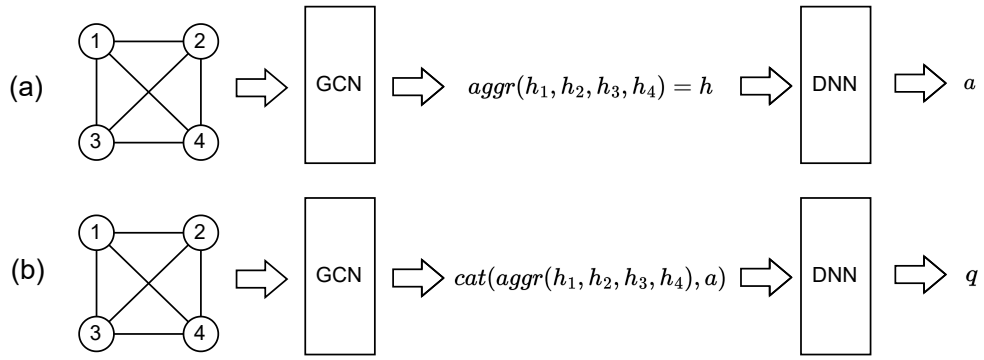
Obr. 5.4: Znázornenie agenta v MEC prostredí

sovému limitu úlohy:

$$L_s(t) = 1 - \frac{1}{N_{CT}(t)} \sum_{i \in \mathcal{T}_{comp}(s,t)} \frac{t_{\text{total},i}}{t_{\text{deadline},i}} \tag{5.14}$$

kde $N_{GT}(t)$ je počet úspešne vypočítaných úloh na s-tej gNb v čase t a $\mathcal{T}_{comp}(s,t)$ je množina úspešne vypočítaných úloh v čase t. Agent tak dostane dodatočnú odmenu za latenciu v prípade, ak všetky úlohy viazané na danú gNB boli úspešne vypočítané, pričom čím nižšiu latenciu dostane, tým sa jeho odmena zvýši. Očakávame, že agent sa v prvom rade bude snažiť dosiahnuť čo najvyšší počet úspešne vypočítaných úloh, pretože bez toho nedostane dodatočnú odmenu podľa latencie.

Zhrnutie použitia agenta pre dynamické rozdeľovanie MEC zdrojov je znázornené na obrázku 5.4. Prostredie pozostáva z viacerých gNB, v ktorých sa u každej nachádza MEC server. V prostredí sú pohybujúce sa CAVs, ktoré sa výhradné pripájajú na gNB s najlepšou SINR hodnotou. gNBs sú prepojené cez cloud, v ktorom sa nachádzajú výkonné servery. Jednotlivé gNB zbierajú vzorky, teda stavy, odmeny a akcie a odosielajú ich do cloud-u. Herec, časť agenta, je rozdistribuovaná na každú gNB, aby sme dosiahli čo najrýchlejšiu reakciu agenta. Navrhnutý stav nie je viazaný na konkrétnu gNB, a preto je možné použiť toho istého agenta pre všetky gNB v prostredí. Samozrejme, nič nebráni tomu aby bol agent viazaný iba ku danej gNB, ale prichádzame tak o možnosť využiť vzorky z iných gNBs. Podľa toho, ako je nastavený algoritmus DDPG, sa v cloud-e z nazbieraných vzoriek vykoná aktualizácia váh herca a kritika a následne sa môže aktualizovaný herec



Obr. 5.5: Znázornenie architektúry sietí agenta na úrovni gNB, (a) herec (b) kritik

rozdistribuovať medzi gNBs.

Vzorky sa ukladajú do zásobníka a použijú sa pri aktualizácii váh. Pomocou nového stavu a Target herca vznikne akcia, ktorá následne spolu s novým stavom putuje do Target kritika, ktorý vyhodnotí akciu od Target herca. Následne sa toto ohodnotenie spolu s odmenou sčíta podľa vzorca 2.3 a vznikne hodnota *target*. Následne kritik ohodnotí reálne vykonanú akciu herca pomocou uloženej vykonanej akcie a stavu, v ktorom herec vykonal akciu. Vznikne nám tak strata, ktorú DDPG alogoritmus maximalizuje na aktualizovanie váh herca. Kritik sa aktualizuje pomocou metódy strednej štvorcovej chyby s vypočítanou *target* hodnotou.

5.2.3 Prideľovanie zdrojov na úrovni gNB

Jednou z možností, ako priraďovať MEC zdroje, je použiť rovnaké rozdelenie na všetky pripojené CAVs. Pre každé CAV sa určí rovnaká hodnota ω_i. Agent bude viazaný na jednu gNB a jeho stav musí vyjadrovať stav gNB a pripojených CAVs. Na to sme použili graf 5.3b na definovanie stavu agenta. Architektúra herca je znázornená na obrázku 5.5a. Stav v podobe grafu vchádza do GCN vrstvy, ktorá vytvorí reprezentáciu uzlov, v tomto príklade $h_1, \ldots, h_4$. Výsledne vektory sa agregujú $aggr$ funkciou a vznikne jeden vektor h, ktorý reprezentuje vnútorný stav gNB a pripojených CAVs. Pri plne prepojenom grafe sú h_1 až h_4 rovnaké po prejdení GCN vrstvy, preto je možné z nich vybrať jeden vektor, ktorý bude zároveň h. Vektor h ďalej vstupuje do plne prepojenej NN, ktorej výstupom je akcia. Architektúra kritika (obrázok 5.5b)je prakticky rovnaká, akurát po agregácii sa pridá akcia ku vzniknutému vektoru (funkcia cat) a spolu vstupujú do plne prepojenej neurónovej siete. GNN tak tvorí extraktor príznakov.

Tým, že agent vykoná rovnaké ω_i pre každé napojené CAV je jasné, že môžu vzniknúť situácie, v ktorých to nebude optimálne kvôli rôznej kvalite signálu.

Očakávame však, že výsledky budu lepšie ako v prípade fixného určenia ω_i a rovnomerného prerozdelenia MEC zdrojov.

5.2.4 Prideľovanie zdrojov na úrovni CAV

Ideálne by bolo vykonávať ω_i zvlášť pre každé CAV. Preto je potrebné, aby stav agenta identifikoval presne stav konkrétneho CAV. Zároveň chceme, aby bol agent použitý na rôzne počty CAVs. Preto náš druhý agent bude vykonávať akcie na úrovni CAV, teda pre každé CAV vyberie akciu zvlášť. Na popis stavu opäť použijeme graf 5.3b, ale len na základe neho nebudeme vedieť rozlíšiť stav CAV, pre ktoré má agent vykonať akciu. To chceme vyriešiť dodatočným pridaním vektora príznakov **x** po agregácii vnútorných reprezentácii uzlov tak, ako to robíme pre kritika pri agentovi na úrovni gNB (operácia cat v obrázku 5.5b), len miesto akcie pridáme **x**. Kritik bude urobený rovnakým spôsobom ako kritik agenta na úrovni gNB s rozdielom, že po agregácii sa ku vektoru pridá vektor **x** spolu s vykonanou akciou.

Očakávame, že keď agent bude mať k dispozícii vnútorný stav gNB a napojených CAVs spolu s vektorom príznakov odpovedajúceho CAV pre ktoré vykonáva akciu, bude sa vedieť rozhodnúť, ako rozdeliť MEC zdroje pre každé CAV zvlášť. To by malo zabezpečiť optimálnejšie rozdelenie zdrojov ako v prípade agenta rozhodujúceho rovnako pre každé napojené CAV.

6 Simulačné prostredie

Simulačné prostredie sme vytvorili v programovacom jazyku Python, pretože neexistuje vhodný simulátor na simulovanie riešení pre MEC problémy. Simulátor môžeme rozdeliť na dve časti, v ktorých jedná časť bude zodpovedná za mobilitu zariadení v priestore a definovanie mapy a druhá bude riešiť MEC ako taký v podobe definovania siete, pripojení a spracovania úloh.

6.1 IISMotion

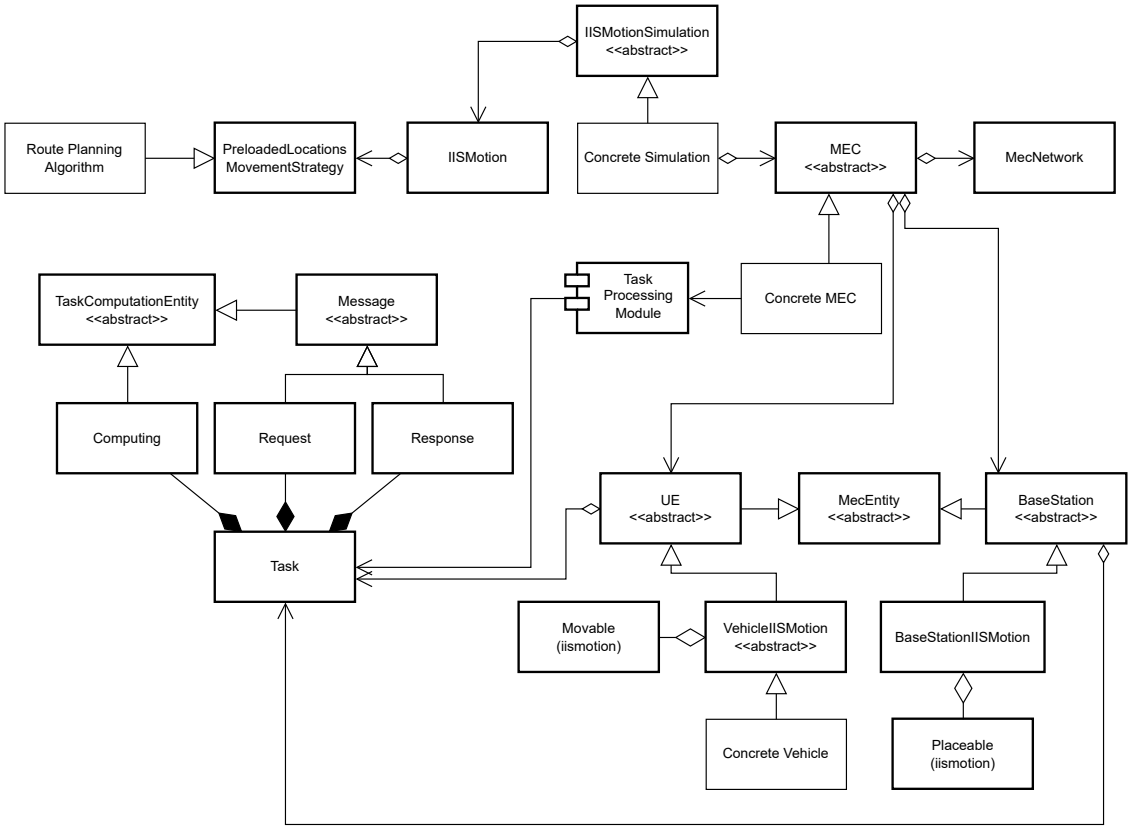
Na simulovanie mobility použijeme projekt *IISMotion*. *IISMotion* je založený na *osmnx* Python balíku, ktorý umožňuje stiahnutie geopriestorových dát z OpenStreatMap ako cesty, chodníky a pracovať s nimi. *IISMotion* predstavuje nadstavbu nad *osmnx* balíkom, v ktorom sú implementované triedy na umiestnenie objektov do mapy a na pohyb na ceste podľa stiahnutej mapy. Mapa cesty je tvorená grafom a každý pohybujúci sa objekt sa pohybuje po hranách tohto grafu podľa jeho rýchlosti. Simulácia neobsahuje semafory ani jazdné pruhy, a teda pohybujúce objekty nie sú na seba závislé, čo znamená, že v simulácii nemôžu vzniknúť zápchy.

Vo všeobecnosti sa budeme snažiť vytvoriť čo najmenšiu závislosť MEC časti od *IISMotion*, aby sa v prípade potreby dala ľahko vymeniť mobilita za viac robustnejšiu.

Jednou z našich úloh je plánovanie trasy, to znamená, že potrebujeme simulovať pohyb áut podľa predom naplánovaných trás. V *IISMotion* sa pohyb vykonáva naraz pre kolekciu objektov, pričom všetký objekty v kolekcii používajú rovnakú stratégiu výberu trasy. *IISMotion* ponúka dve možnosti pre výber trás, náhodný výber bodu na mape, ktorý je destináciou a vyberie sa najkratšia trasa alebo sa pre daný objekt načítajú lokácie s odpovedajúcim časom predom, kde sa počas simulácie použijú lokácie podľa aktuálneho času simulácie. Logicky pre naše plánovanie trasy využijeme druhú možnosť. V prípade druhej časti práce pre dynamické rozdelenie MEC zdrojov použijeme prvú možnosť, teda výber najkratšej trasy ku

náhodne zvolenému bodu na ceste.

6.2 Popis simulačného prostredia



Obr. 6.1: Diagram tried sady nástrojov na simulovanie

Na obrázku 6.1 je znázornený diagram tried zobrazujúci podstatné triedy a vzťahy medzi nimi. Hrubým orámovaním sú označené triedy a moduly, ktoré sú základnou súčasťou vytvorenej sady nástrojov na simulovanie MEC systému. Cieľom je, aby sa pomocou základných objektov dala vyskladať simulácia pre konkrétny prípad použitia či už priamym dedením, alebo kompozíciou.

Základnou abstraktnou triedou simulácie je *IISMotionSimulation*. Ako je na diagrame znázornené, trieda obsahuje *IISMotion* triedu, ktorá zaobaľuje funkcionalitu krokovania pohybu objektov. *IISMotionSimulation* obsahuje hlavnú slučku simulácie a obsahuje potrebnú implementáciu aby sa *IISMotion* mohol jednoducho použiť na krokovanie v podtriedach. Obsahuje abstraktnú metódu, ktorou podtriedy definujú, čo a v akom poradí sa udeje pri jednom kroku simulácie. Konkrétna implementácia triedy *IISMotionSimulation* je v diagrame nazvaná *Concrete Simulation*. Konkrétna implementácia môže mať na starosti aj napríklad genero-

vanie úloh, ukladanie medzivýsledkov alebo iné procesy súvisiace s priebehom simulácie.

Ďalšou podstatnou triedou je abstraktná trieda *MEC*. V diagrame je síce zaznačené, že konkrétna simulácia používa *MEC* triedu, ale to nie je nutné, v budúcnosti sa môže vytvoriť iná variácia triedy *MEC*. Momentálne *MEC* trieda obsahuje všetkých používateľov MEC (trieda *UE*), všetky bázove stanice (trieda *BaseStation*) v prostredí a triedu MECNetwork. Spracovanie MEC časti sme rozdelili na try fázy: nahrávanie úloh, výpočet úloh a sťahovanie výsledkov úloh. Preto *MEC* obsahuje metódu pre každú z týchto fáz. *MEC* obsahuje aj metódu aktualizovania siete MEC. Zámer tejto metódy je, aby sa volala pred spracovávaním úloh. Jej úlohou je nastaviť prenosové rýchlosti, respektíve vypočítať SINR hodnoty a vytvoriť spojenia medzi používateľmi MEC a bázovými stanicami. Potom pri spracovávaní úloh sa použije aktuálne MEC prostredie so zvolenými prenosovými rýchlosťami a topológiou siete. Topológia siete je vyjadrená triedou MECNetwork v podobe grafu, v ktorom sú dva typy uzlov pre *UE* a *BaseStation* a dve typy hrán reprezentujúcich buď rádiové, alebo fyzické spojenie medzi *UE* a *BaseStation* alebo *BaseStation* a *BaseStation*. Do hrán sa ukládajú aj informácie o SINR hodnote alebo kapacity prenosu v prípade fyzického spojenia.

Základ entity MEC systému je definovaný v abstraktnej triede *MECEntity*. Pod MEC entitou rozumieme v tomto prípade objekt s polohou a jedinečným identifikátorom, napríklad vozidla, gNBs, drony. Obsahuje aj dva fronty na úlohy určené na výpočet na danej entite a na úlohy určené na odoslanie inej entite. Z *MECEntity* sme dedením vytvorili *UE* a *BaseStation* predstavujúce základné triedy. *UE* obsahuje abstraktné metódy na prijatie vypočítanej úlohy a metódy spojené s generovaním úloh. Trieda *BaseStation* obsahuje premenné špecifické pre bázové stanice, ako napríklad priemer pokrytia, vysielací výkon, nosná frekvencia a pod. Z *UE* sme vytvorili *VehicleIISMotion*, ktorý používa *Movable* triedu z *IISMotion* projektu. *Movable* tvorí reprezentáciu pohybujúceho sa objektu v mape. Z objektu *Movable* využíva trieda *VehicleIISMotion* iba aktuálnu lokáciu a jedinečný identifikátor. Keď sa aktualizuje lokácia daného *Movable* objektu, zároveň sa to odzrkadlí v lokácii *VehicleIISMotion*. Obdobne je to aj v prípade triedy *BaseStationIISMotion*, kde sa miesto *Movable* použije trieda Placeble, pretože gNB nie je pohybujúci objekt. V prípade, ak bude potrebné v budúcnosti použiť iný systém mobility, malo by ho byť veľmi jednoduché integrovať.

Keďže *MEC* definuje prevažne rozhranie, v konkrétnej implementácii *MEC* je potrebné definovať, ako sa sieť bude aktualizovať, a ako sa budú spracovávať úlohy. To bude závisieť od konkrétneho prípadu použitia. Preto sme sa rozhodli

definovať funkcie spracovania úloh a vytvoriť z nich modul. V našom prípade sa hodí spracovanie úloh spôsobom *round robin*, teda každej úlohe, či už pri odosielaní alebo výpočte, sa priradia zdroje. Priradenie zdrojov je robené pomocou funkcie spätného volania, ktorá sa volá pre každé *UE*, vďaka čomu môžeme ľubovoľným spôsobom definovať, ako sa zdroje prerozdelia podľa vlastníka úlohy.

Úloha (trieda *Task*) v sade nástrojov zaobaľuje tri časti: *Request*, *Response* a *Computing*. Ide o komponenty schopného výpočtu. V prípade *Request* a *Response* máme na mysli výpočet v podobe odosielania úlohy. *Task Procesing Module* spracováva úlohy tak, že odčítava z požadovaného výpočtu (pre *Request* a *Response* je to veľkosť v megabajtoch a pre *Computing* je to počet inštrukcii) podľa výpočtového výkonu (čo je prenosová rýchlosť alebo priradený výkon CPU). *Request* teda označuje úlohu v podobe objektu, ktorý sa odosiela do MEC siete, *Computing* reprezentuje výpočtovú časť úlohy a *Response* reprezentuje úlohu ako výsledok. Základné údaje o úlohe, ako jej veľkosť, vlastník, cieľová gNB sú zaznamenané v *Task* triede, pričom každý komponent má referenciu na prisluchajúci *Task*.

Ak chceme ovplyvniť plánovanie trasy, môžeme tak urobiť zdedením *PreloadedLocationsMovementStrategy* a prepísaním metód na výber prislúchajúcej lokácie v danom časovom úseku.

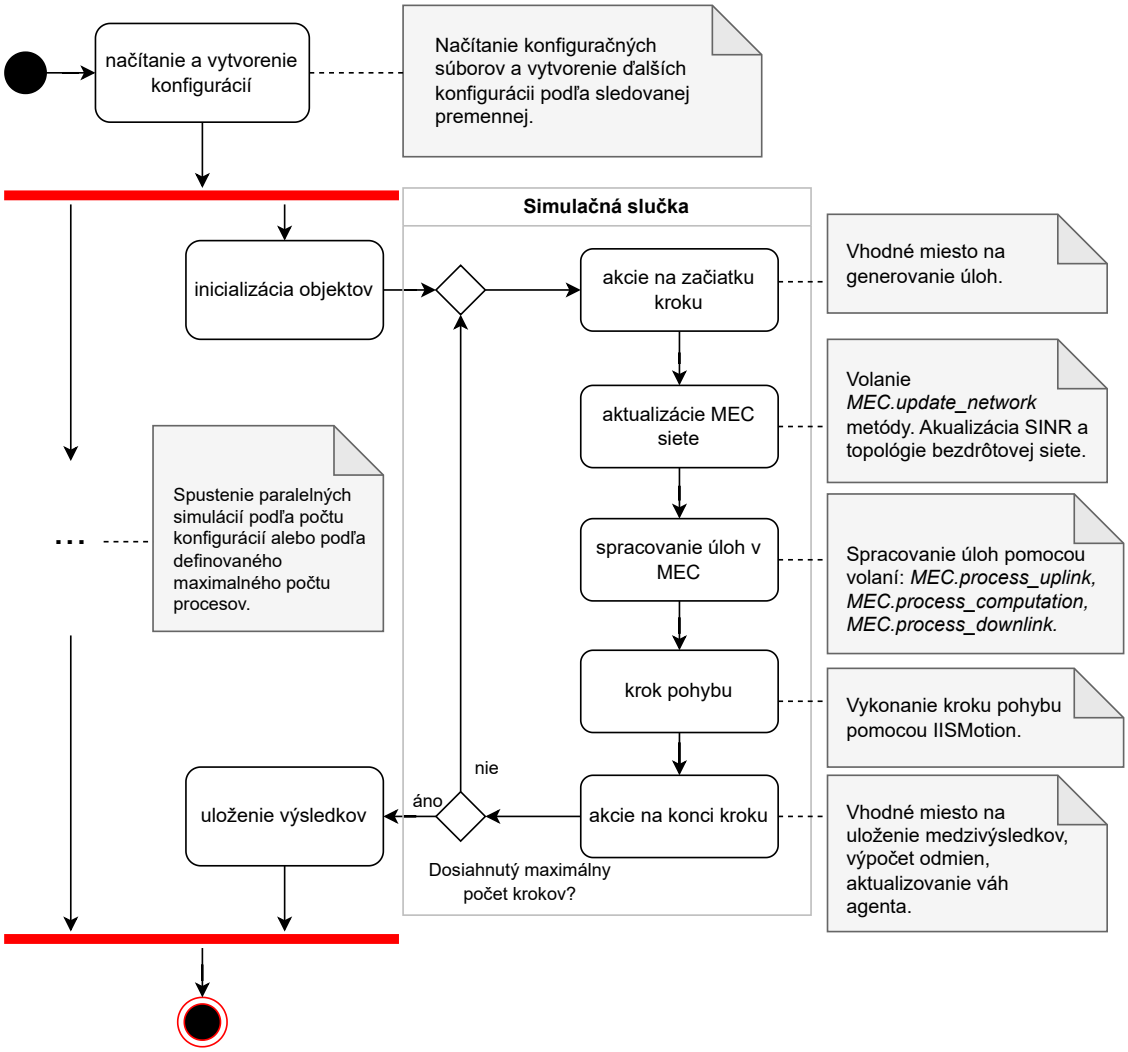
Pre naše potreby simulácie sme urobili viacero tried a funkcií. Vznikli pomocné funkcie pre prácu s trasami, napríklad výpočet budúcich lokácii CAV na trase podľa jeho rýchlosti, výpočet času trasy, vytvorenie bodov na trase, pomocné funkcie pre výpočet SINR a prenosovej rýchlosti podľa vzorcov 1.4 a 3.4, triedy na zber štatistík a priebežne ukladanie výsledkov a priebehu učenia agentov. Implementovali sme aj SINR mapu, ide o mapu, v ktorej je priestor rozdelený na obdĺžnikové oblasti o istej veľkosti, do ktorých sa ukladajú namerané SINR hodnoty, po čase tak vznikne mapa s priemernými SINR hodnotami v oblastiach, ktoré následne môžeme použiť miesto nákladného výpočtu SINR, a tým skrátiť čas simulácie.

V prípade učenia agenta sme vytvorili triedu MECAgent, ktorá zaobaľuje funkcionalitu rátania odmeny, výberu akcie, zber vzoriek a aktualizovanie NN. Použila sa v konkrétnej implementácii *MEC* v metóde pre aktualizáciu siete a v slučke simulácii, kde sa aktualizovali váhy NN na konci simulačného kroku. Akcie agenta sa prepočítali na rezervácie pre jednotlivé *UE* a uložili sa do príslušnej *BaseStation*. Následne sa rezervácie použili pri prerozdelení zdrojov pri výpočte komunikačnej a výpočtovej časti úlohy.

Sada nástrojov obsahuje základne komponenty, ktoré môžu byť rozšírene, upravené a implementované rôznorodým spôsobom, čo sa pri výskumných úlo-

hách hodí, pretože každý výskum môže mať iné požiadavky na simuláciu a nie stále vieme, akým smerom sa simulácia aj samotný výskum vyberie. Negatívum toho je, že je potrebná vysoká znalosť komponentov a potreba programovania a spojenia komponentov, aby sme simuláciu vytvorili podľa potreby.

6.3 Simulačná slučka



Obr. 6.2: Diagram akcíi simulácie

Na obrázku 6.2 je znázornený náhľad dôležitých krokov simulácie v podobe diagramu akcii. Obvykle je pri simuláciách potrebné konfigurovať značné množstvo parametrov, je to tak aj v našom prípade. Preto je prvým krokom načítanie konfiguračného súboru alebo súborov formátu json. V tomto kroku ešte vykonávame jednu dôležitú vec. Pri simuláciách sa tiež väčšinou pozoruje vplyv zmeny jedného parametra na výsledky, napríklad budeme sledovať, ako vplýva

počet CAVs na množstvo zahodených úloh. Aby sme nemuseli vytvárať viacero konfiguračných súborov, ktoré sa budú líšiť v jednom parametri, implementovali sme možnosť definovať priamo v konfiguračnom súbore, ktoré parametre sa majú meniť, a aké hodnoty majú nadobudnúť. Podľa toho sa pri procese načítavania konfigurácie vytvoria, respektíve nakopírujú so zmeneným definovaným parametrom. Bude tak existovať napríklad jeden konfiguračný súbor, kde budú definované rôzne počty CAVs v prostredí.

Po spracovaní konfigurácie je možne paralelizovať simulácie tak, že každá simulácia sa bude vykonávať vo vlastnom procese. Je možné definovať aj maximálny počet procesov, aby sme nepreťažili CPU. Každá simulácia obdrží jednu spracovanú konfiguráciu.

Po paralelizácii prichádza fáza inicializácie objektov. Všetky objekty sa inicializujú podľa konfigurácie. Keďže simulácia obsahuje implementáciu rôznych algoritmov, ktoré sa od seba môžu značne líšiť, je vhodné v tomto bode rozlíšiť algoritmus, ktorý chceme simulovať a oddeliť ho na základe jedného parametra konfigurácie a prípadne vytvoriť rôzne vstupné body pre každý rôzny algoritmus, aby sme udržali čistý kód.

Po inicializácii sa začne vykonávať simulačná slučka. Na začiatku kroku simulácie máme možnosť vykonať prípravné veci. Tu napríklad generujeme nové úlohy na celý simulačný krok a resetujeme použitý čas úloh z minulého kroku. Pri každej úlohe sa zaznamenáva, koľko z celkového času simulačného kroku bola spracovávaná, aby sme náhodou úlohu nespracovávali dlhšie, ako je dĺžka časového kroku.

Druhý krok simulácie je aktualizácia MEC siete pomocou volania *MEC.update_network* metódy. V tomto kroku dochádza ku výpočtu SINR a výberu spojenia medzi gNB a používateľom MEC. Je to vhodné miesto aj na vykonanie akcii agenta, ak sa akcie viažu na výber gNB, kam sa má používateľ napojiť alebo na alokáciu zdrojov, ktorá sa môže uložiť do *MEC* objektu alebo priamo na príslušnú *BaseStation*.

Po aktualizácii je *MEC* pripravený na vykonanie kroku spracovania úlohy. To sa skladá z troch volaní *MEC* metód. *MEC.process_uplink* spracuje komunikáciu nahrávania do MEC siete. Všetky úlohy sa postupne odosielajú, až dokým neubehne celý čas simulačného kroku. Ako sme spomenuli, úlohy majú v sebe uložený použitý čas z časového kroku simulácie. Po uplynutí sa odoslané úlohy vypočítajú pomocou metódy *MEC.process_computation* a spotrebovaný čas úlohy sa opäť zníži. Potom sa metódou *MEC.process_downlink* stiahnu výsledky úloh. Ak počas *MEC.process_uplink* alebo *MEC.process_downlink* dôjde ku vypršaniu limitu

úlohy, úloha sa zahodí.

Po spracovaní úlohy nasleduje krok pohybu objektov v mape pomocou *IISMotion*. Na konci kroku simulácie je vhodné uložiť medzivýsledky alebo vypočítať odmenu pre agenta a aktualizovať jeho váhy. Na konci kroku sa aj skontroluje, či sa už dosiahol maximálny počet krokov simulácie. Ak nie, vraciame sa na začiatok simulačného kroku. V opačnom prípade ukončujeme simulačnú slučku, uložíme finálne výsledky a proces simulácie ukončíme. Ak sa ukončia všetky simulácie, hlavný proces automatický skončí.

7 Vyhodnotenie simulácií

Výsledky sme dosiahli metódou Monte Carlo. Vo všeobecnosti sme skúmali správanie algoritmov pri zmene hodnôt parametrov $N_{\mathrm{CAV}}, N_{\mathrm{gNB}}, D_i, I_i, N_{\mathrm{RB}}$.

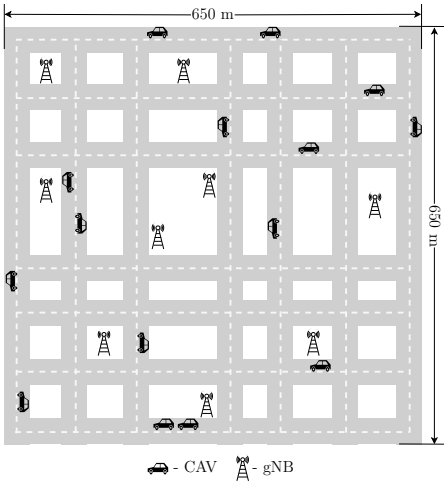
7.1 A* plánovanie trasy

Na porovnanie výkonnosti nášho algoritmu A* sme implementovali najmodernejšie metódy pre výber trás:

- FPR algoritmus: Vyberá pre CAV najrýchlejšiu dostupnú trasu na dostavenie sa do cieľového bodu. Pripomína výber trasy tak, ako to robí navigačný systém pomocou GPS.
- SQRSA [33]: Ide o algoritmus popísaný v sekcii 4.2. Pre zhrnutie, trasa sa pre CAV vyberá na základe SINR hodnôt, ktoré sú dostupné v jednotlivých cestných segmentoch trasy. Kvalita cesty je ohodnotená podľa počtu cestných segmentov s nedostatočnou kvalitou signálu, teda kde je SINR hodnota nízka (SINR hodnota < 5 dB).
- VaRSA [34]: Algoritmus je tiež popísaný v sekcii 4.2. Ide o rozšírenie algoritmu SQRSA, tým, že sa algoritmus pozerá na kapacitu komunikačných zdrojov.

V simulácii používame manhattanovskú mapu znázornenú na obrázku 7.1. Simulovaná oblasť má rozmery 650×650 metrov so siedmimi vodorovnými a horizontálnymi cestami. Vďaka tomu sme mohli použiť manhattanovskú vzdialenosť ako heuristiku $h(\cdot)$. V oblasti sa nachádza fixný počet CAVs a gNBs podľa konfigurácie počas celého chodu simulácie. Ak CAV dorazí do cieľového bodu, je mu náhodné vybratý další cieľový bod a spustí sa tak algoritmus výberu trasy. Pozície gNBs sú náhodne vybraté pri každej simulácii.

Zhrnutie nastavených parametrov simulácii je zobrazené v tabuľke 7.1. Simulácie budú trvať 1200 krokov, pričom jeden simulačný krok reprezentuje 0.5 sekundy. V prostredí budeme uvažovať o rôznom počte CAVs od 30 až po 120, ktoré



Obr. 7.1: Uvažovaná oblasť simulácie pre výber trasy

Parameter	Hodnota
N_{steps} (počet simulačných krokov)	1200
Δt_{T}	0.5 s
N_{CAV} (počet CAVs)	30, 60, 90, 120
N_{gNB}	5, 8, 10, 12, 15, 17, 20
$N_{\text{RB},s}$ (počet RBs na gNB)	2000, 4000
v (rýchlosť CAV)	40 km/h
D_i	0.5, 1, 1.5, 2 MB
I_i	10, 20, 40 MI
$I_{\text{CPU},s}$	$3.6 * 10^9$ IPS
ω_i	0.9 s
rozmiestnenie gNBs	náhodné rozmiestnenie
P_{t}	0.1 W
f_{t}	$2 * 10^9$ Hz

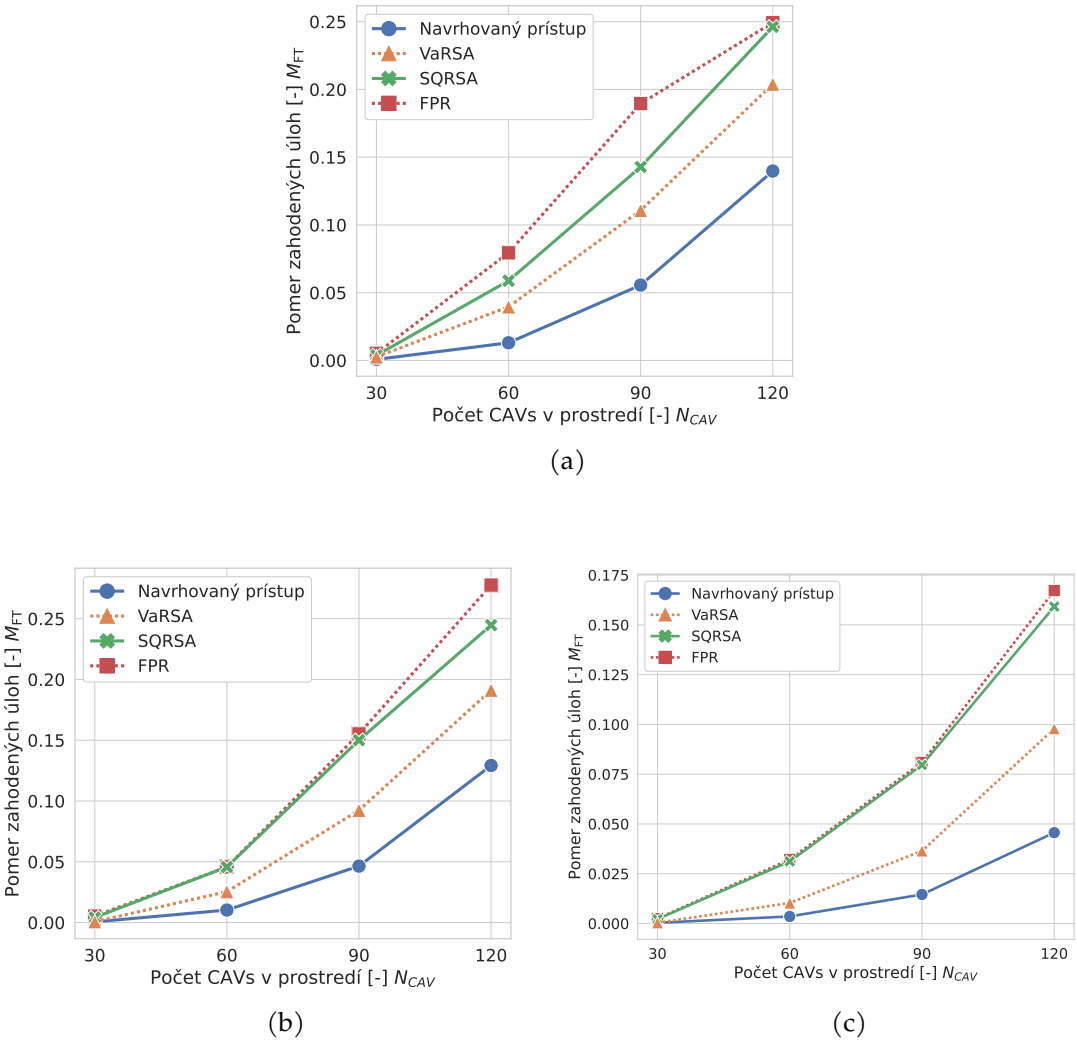
Tabuľka 7.1: Nastavenia simulácie plánovania trasy

sa budú pohybovať rýchlosťou 40 km/h a o rôznom počte gNBs od 5 až po 20. Počet zdrojových blokov budeme tiež obmieňať za 2000 alebo 4000. Výpočtový výkon gNB sme nechali fixne nastavený na 3600 MIPS. Veľkosť úlohy na odoslanie budeme obmieňať od 0.5 do 2 MB a počet inštrukcii úlohy budeme obmieňať od 10 do 40 MI. CAV bude posielať úlohu na odoslanie každý simulačný krok, pričom konečný čas výpočtu je 0.5 sekúnd, rovnako ako čas simulačného kroku.

7.1.1 Pomer zahodených úloh ku celkovému počtu vygenerovaných úloh

V tejto sekcii analyzujeme vplyv zmeny parametrov simulácie na pomer zahodených úloh ku celkovému počtu vygenerovaných úloh vyjadrený metrikou M_{FT}. V grafoch 7.2a–7.2c sme menili počet gNBs a dostupnú šírku pásma, teda počet zdrojových blokov N_{RB} gNBs a sledovali sme, ako sa bude vyvíjať množstvo zahodených úloh pri rôznom počte CAVs v prostredí. So zvyšujúcim sa počtom CAVs vzniká zároveň väčšia zátaž na MEC zdroje, preto majú krivky tendenciu stúpať. Pri 30 CAVs v prostredí sa všetky algoritmy správajú rovnako vo všetkých grafoch skupiny 7.2 – hodnoty metriky M_{FT} sú 0 alebo blízko 0. Pri počte 60 CAVs sa už začínajú prejavovať rozdiely.

V grafe 7.2a je v prostredí 20 gNBs a každá má k dispozícii 2000 zdrojových blokov. FPR prístup dosahuje najhoršie výsledky, pri 10 CAVs dosahuje $M_{\text{FT}} = 0.08$ a pri 90 CAVs až $M_{\text{FT}} = 0.19$. Algoritmus SQRSA dosahuje o niečo lepšie výsledky od FPR, napríklad pri počte 90 CAVs dosiahol $M_{\text{FT}} = 0.14$, čo je o 0.05 lepšie. Avšak pri 120 CAVs je pomer zahodených úloh takmer rovnaký s FPR, a to



Obr. 7.2: Vplyv počtu CAVs na M_{FT} pre $I_i = 40$ MI a (a) 20 gNBs a 2000 RBs dostupnej šírky pásma, (b) 20 gNBs a 4000 RBs dostupnej šírky pásma, a (c) 30 gNBs a 4000 RBs dostupnej šírky pásma

0.25. VARSA prístup dosahuje celkovo nižšie hodnoty metriky. Pri 60 a 90 CAVs je pomer zahodených úloh rovný 0.04 a 0.11, čo je zníženie oproti SQRSA približne o 0.03. Náš prístup v podobe algoritmu A* dosahuje značne nižšie hodnoty metriky M_{FT} oproti ostatným prístupom. Pri 90 CAVs v prostredí dosahuje hodnotu metriky niečo nad 0.05, čo je nižšie ako v prípade FPR a SQRSA pri počte 60 CAVs a takmer rovnako ako VARSA prístup. Pri 90 aj 120 CAVs je v priemere pomer zahodených úloh o 0.05 nižší oproti VARSA a o 0.10 a viac oproti FPR a SQRSA. Vo všeobecnosti sa rozdiel oproti ostatným prístupom a naším prístupom zväčšuje so zvyšovaním sa počtu CAVs

V grafe 7.2b sa zvýšil počet zdrojových blokov z 2000 na 4000. V grafe sa ukazuje približne rovnaký trend ako v 7.2a, vzniká väčší rozdiel medzi naším prístu-

pom a ostatnými algoritmami so zvyšujúcim počtom CAVs. FPR a SQRSA dosahujú takmer identické hodnoty metriky ako v grafe 7.2a. VaRSA a náš prístup sa mierne zlepšili a dosahujú v priemere o 0.01 až 0.02 nižšie hodnoty metriky M_{FT}.

Rozdiel medzi naším a ostatnými prístupmi sa ešte viac zväčší pri zvýšení počtu gNBs v prostredí z 20 na 30 pri postupnom zvyšovaní počtu CAVs, čo je znázornené v grafe 7.2c. FPR a SQRSA vykazujú takmer rovnaké výsledky. VARSA prístup je od nich výrazné lepší, napríklad pri 90 CAVs dosahuje približne $M_{\mathrm{FT}} = 0.04$, pričom FPR a SQRSA dosahujú $M_{\mathrm{FT}} = 0.08$, teda dvakrát toľko. Naproti tomu dosahuje náš prístup oproti VARSA o polovicu nižšiu hodnotu M_{FT} pri 90 CAVs, čo je niečo pod 0.02. Pri 120 CAVs je rozdiel oproti ostatným metódam najvýraznejší, náš prístup dosahuje 0.045, pričom VARSA dosahuje 0.1 a FPR spolu s SQRSA približne 0.16.

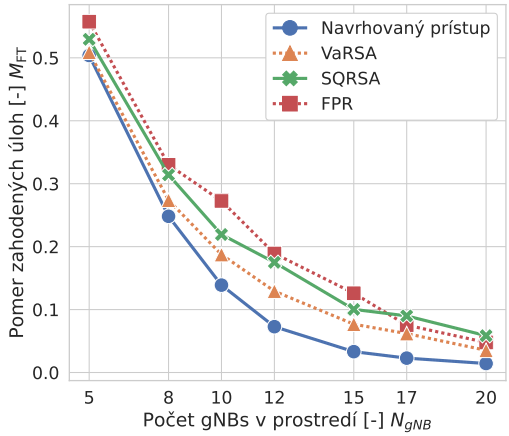
Grafy 7.3a–7.3c znázorňujú vplyv počtu gNBs v prostredí na M_{FT} pri rôznom nastavení parametrov. Hodnota metriky M_{FT} s narastajúcim počtom gNBs klesá, pretože zdroje v MEC sa zvýšia pridávaním gNB. Opäť môžeme vidieť, že náš prístup dosahuje najlepšie výsledky vo všetkých troch grafoch a poradie ostatných metód je rovnaké ako v predchádzajúcej skupine grafov.

Ak sa pozrieme na graf 7.3a, v ktorom uvažujeme o 60 CAVs v prostredí a dostupnej šírke pásma v podobe 2000 zdrojových blokov, zníženie metriky M_{FT} pod úroveň 0.1 je dosiahnutá v našom prístupe pri počte 12 gNBs, zatiaľ čo podobná výkonnosť je pre ostatné prístupy dosiahnutá pri počte až 17 gNBs. To znamená, že pre rovnakú výkonnosť sme dokázali ušetriť 40% z počtu potrebných gNBs.

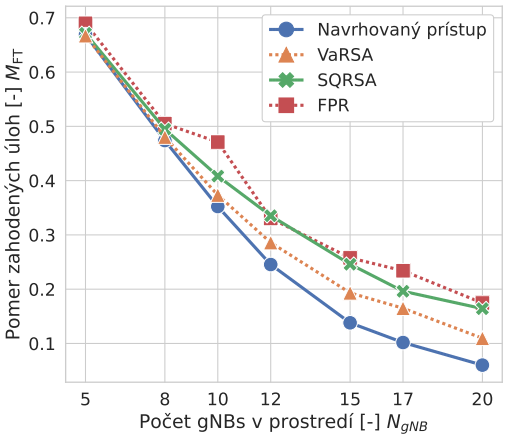
V grafe 7.3b sme zvýšili počet CAVs v prostredí oproti 7.3a o 30, teda celkovo máme v prostredí 90 CAVs. Zvýšením počtom CAVs sme zvýšili nároky na MEC, a preto sú krivky strmšie a hodnoty metriky sú vyššie. Náš prístup dosiahol $M_{\mathrm{FT}} = 0.1$ až pri počte 17 gNBs, ale VaRSA dosiahla rovnakú hodnotu metriky pri počte 20 CAVs, pričom pri rovnakom počte CAVs, SQRSA a FPR dosiahli $M_{\mathrm{FT}} = 0.18$.

Graf 7.3c znázorňuje situáciu s $N_{\mathrm{CAV}} = 90$ s počtom 4000 zdrojových blokov, čo je dvojnásobok oproti grafom 7.3a a 7.3b. Tvar kriviek aj dosahované hodnoty metriky sú o málo lepšie ako v grafe 7.3b. Náš prístup sa priblížil ku $M_{\mathrm{FT}} = 0.1$ už pri 15 gNBs. VaRSA dosahuje rovnakú výkonnosť pri 20 gNBs, čo je o 33% väčší počet. Aj v tomto prípade FPR a SARSA metódy nedosahujú hodnotu metriky 0.1 ani pri 20 gNBs.

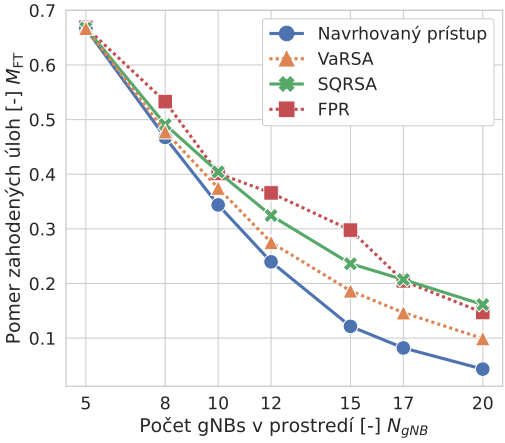
Vo všeobecnosti sa rozdiel medzi naším prístupom a ostatnými metódami viac prehlbuje s narastajúcim počtom gNBs do momentu, kedy náš prístup dosiahne $M_{\mathrm{FT}} = 0.05$. V tom momente v MEC sieti začína byť dostatok zdrojov pre od-



(a)



(b)



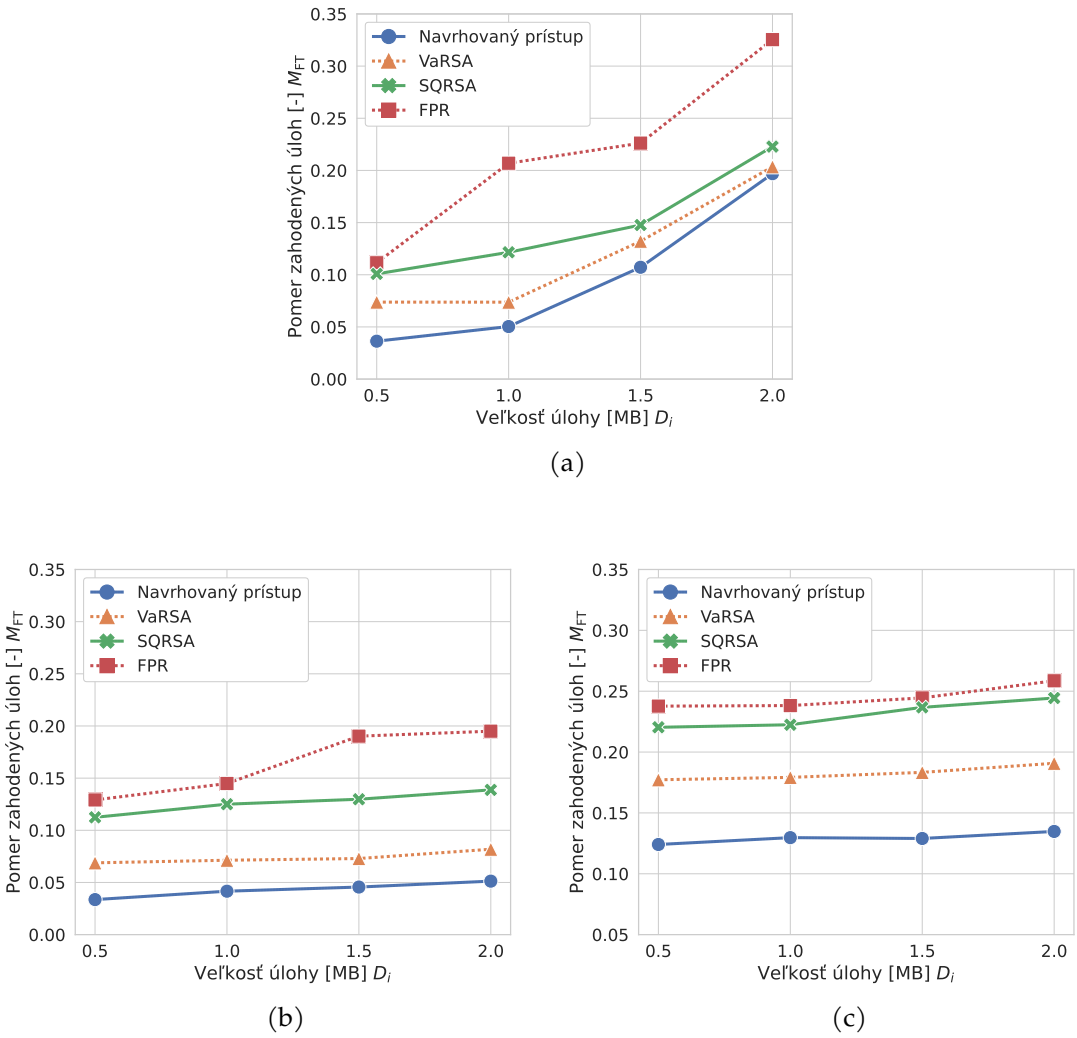
(c)

Obr. 7.3: Vplyv počtu gNB na M_{FT} pre $I_i = 40$ MI a (a) 60 CAVs a 2000 RBs dostupnej šírky pásma, (b) 90 CAVs a 2000 RBs dostupnej šírky pásma, a (c) 90 CAVs a 4000 RBs dostupnej šírky pásma

povedajúci počet používateľov. Potom sa rozdiel postupne zmenšuje, pozri graf 7.3a. Fakt, že na splnenie rovnakej výkonnosti je pri našom prístupe možné použiť menej gNBs, je obzvlášť podstatné pre mobilných operátorov, keďže to dokáže ušetriť kapitálové výdavky aj prevádzkové náklady.

Vplyv veľkosti úlohy D_i a I_i na M_{FT} je zobrazený v grafoch 7.4a–7.4c. M_{FT} sa zvyšuje so zvyšujúcou sa veľkosťou úlohy. Náš prístup dosahuje vo všetkých grafoch nalepší výsledok pre všetky veľkosti úlohy.

V Grafe 7.4a sa výrazne zvýšila hodnota metriky pre veľkosti úloh 1.5 a 2 MB pre všetky metódy. FPR dosahuje výrazne horšiu výkonnosť ako ostatné prístupy. Hoci je náš prístup najlepší, rozdiel oproti SQRSA a VARSA sa znižuje so zvyšovaní veľkosti úlohy, napríklad pri $D_i = 1.5$ dosahujú SQRSA a VaRSA o 0.05



Obr. 7.4: Vplyv veľkosti úlohy na M_{FT} v niekoľkých simulovaných scenároch: (a) 60 CAVs, úlohy s 30 MI a 2000 RBs dostupnej šírky pásma, (b) 60 CAVs, úloh s 30 MI a 4000 RBs dostupnej šírky pásma a (c) 60 CAVs, úlohy so 40 MI a 4000 RBs dostupnej šírky pásma

a 0.025 v tomto poradí vyššiu hodnotu metriky od nášho prístupu a pri $D_i = 2$ je už výkon VARSA takmer totožný s naším prístupom a rozdiel SQRSA sa znížil o polovicu na 0.025. Deje sa to preto, lebo MEC má vysoký nedostatok komunikačných zdrojov, a to výrazne prevyšuje nároky na výpočtové zdroje.

V grafe 7.4b sme zvýšili počet zdrojových blokov na 4000. Krivky zobrazených metód rastú miernejšie ako v grafe 7.4b. Náš prístup si drží približne rovnaký rozdiel od ostatných prístupov počas všetkých veľkostí úloh. V prípade VaRSA a SQRSA je hodnota metriky vyššia o 0.03 a 0.09, čo predstavuje relatívne zvýšenie M_{FT} o 60% a 180% v tomto poradí vzhľadom na náš prístup.

V grafe 7.4c, kde došlo ku zvýšenie počtu inštrukcii úlohy I_i oproti grafu 7.4b

z 30 MI na 40 MI. Náš prístup sa ešte viac vzdialil od ostatných znázornených prístupov a opäť je nárast metriky mierny so zvyšovaním veľkosti úlohy. Opäť môžeme pozorovať zvýšenie M_{FT} až do 0.06, 0.11 a 0.12 (46%, 85% and 92% relatívne vzhľadom na náš prístup), pre VaRSA, SQRSA and FPR v tomto poradí.

7.1.2 Predĺženie trasy

Rozhodli sme sa analyzovať aj priemerné predĺženie cesty oproti najkratšie trvajúcej trasy. Preto sme si zadefinovali metriku predĺženia trasy (M_{RP}) nasledovne:

$$M_{\mathrm{RP}} = (\frac{T_{\mathrm{pr}}}{T_{\mathrm{FPR}}} - 1), \tag{7.1}$$

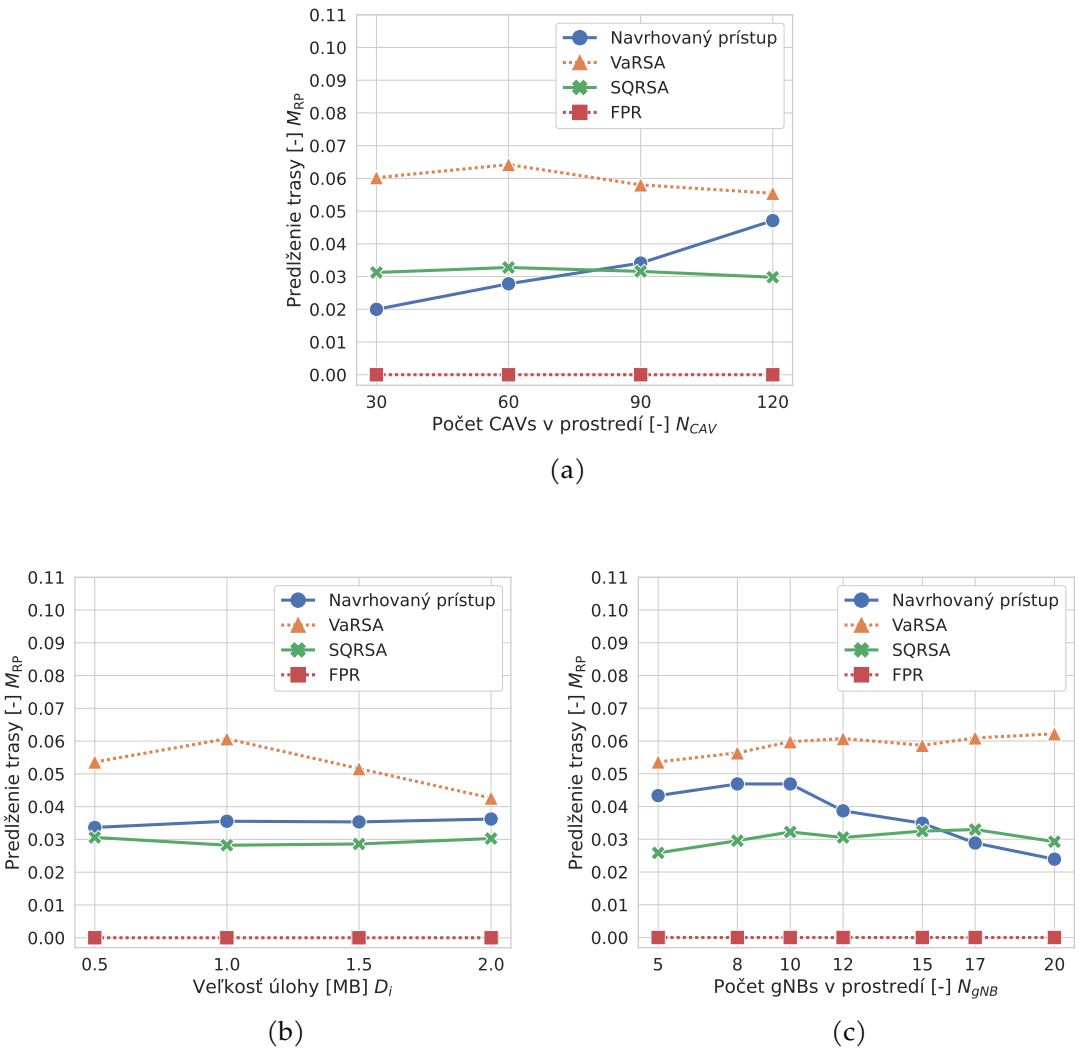
kde T_{pr} predstavuje čas zvolenej cesty a T_{FPR} predstavuje čas najrýchlejšej cesty podľa FPR prístupu.

Predĺženie trás, ku ktorým došlo pri výbere vhodnej trasy podľa prístupov FPR, SQRSA, VaRSA a nášho prístupu, v ktorom sme uvažovali o rôznom počte CAVs a gNBs a rôznej veľkosti úlohy, je zobrazené v grafoch 7.5a–7.5c. Keďže FPR vyberá stále najrýchlejšiu trasu, je samozrejmé, že predĺženie trasy pri FPR nenastane, teda $M_{\mathrm{RP}} = 0$.

Z grafu 7.5a vieme vyčítať, že až do počtu približne 80 CAVs dosahuje náš algoritmus nižšie predĺženie trasy oproti SQRSA a VaRSA. Pri počte 90 a 120 CAVs predĺženie trasy naším prístupom ďalej narastá, až presahuje predĺženie SQRSA. Napriek tomu je predĺženie nižšie ako v prípade VaRSA prístupu, ktoré je najvyššie pre všetky počty CAVs, ktoré graf zobrazuje. Zvyšovanie predĺženia cesty spôsobene naším prístupom je očakávané, pretože čim viac CAVs sa v prostredí nachádza, tým presnejšie prideľovanie zdrojov gNB mierne predlžuje dopravné trasy.

Graf 7.5b zobrazuje vplyv veľkosti úlohy na predĺženie trasy. Náš prístup dosahuje konštantné, ale mierne vyššie predĺženie ako SQRSA s hodnotou metriky $M_{\mathrm{RP}} = 0.035$. VaRSA aj tu dosahuje výrazne horšie výsledky aj pri menších veľkostiach úloh.

Na záver graf 7.5c zobrazuje vplyv počtu gNBs na predĺženie trasy. Predĺženie spôsobené algoritmom SQRSA je opäť konštantné s hodnotou metriky $M_{\mathrm{RP}} = 0.03$. Náš prístup spôsobuje približne konštantné predlženie do počtu 10 gNBs, a to niečo pod 0.05. Pre počty gNBs 12 a viac postupné predĺženie trasy klesá, až pri 16 gNBs dosiahne rovnaké predlženie, ako je predlženie spôsobené prístupom SQRSA. Od toho momentu dosahuje dokonca nižšie predĺženie ako SQRSA, až dosiahne takmer $M_{\mathrm{RP}} = 0.03$ pri 20 gNBs. Predlženie spôsobené algoritmom



Obr. 7.5: Pomer predĺženia dopravnej cesty vo vzťahu k najkratšej dopravnej trase s (a) rôznym počtom CAVs v rovnakej konfigurácii simulácie ako v grafe 7.2b, (b) rôznou veľkosťou úlohy v rovnakej konfigurácii simulácie ako v grafe 7.4b a (c) s rôznym počtom gNBs s $I_i = 40$, 60 CAVs a 4000 RBs dostupnej šírky pásma

VaRSA je opäť najväčšie, a dokonca so zvyšujúcim sa počtom gNBs rastie, až dosiahne hodnotu $M_{\mathrm{RP}} = 0.06$ pri 20 gNBs.

Z porovnania grafov zobrazujúcich predlženie trasy vidíme, že SQRSA dosahuje vo všetkých zobrazených scenároch približne rovnaké konštantné predĺženie. Zmena parametrov, o ktorých sme uvažovali nemá vplyv na hodnotu SINR, čo je hlavný indikátor kvality trasy pre SQRSA, a preto je predĺženie nemenné a konštantné. Náš prístup ukazuje trend, v ktorom je predlženie trasy tým menšie, čím je viac MEC zdrojov k dispozícii, respektíve, tým väčšie, čím je menej MEC zdrojov k dispozícii.

Parameter	Hodnota
Δt_{T}	0.5 s
$t_{\mathrm{deadline},i}$	0.1 s
N_{gNB}	1, 2, 3, 4, 5
$N_{\mathrm{RB},s}$ (počet RBs na gNB)	1000
v (rýchlosť CAV)	$< 35, 50 >$ km/h
D_i	0.1 MB
I_i	10 MI
$I_{\mathrm{CPU},s}$	$3 * 10^9$ IPS
rozmiestnenie gNBs	rovnaké rozmiestnenie
P_{t}	0.01 W
f_{t}	$2 * 10^9$ Hz

Tabuľka 7.2: Základné nastavenia simulácie dynamického pridelenia zdrojov

7.2 Dynamická alokácia

Pre porovnanie dynamickej alokácie zdrojov sme implementovali nasledujúce algoritmy:

- Rovnomerné prerozdelenie (RP): Prístup rovnomerného prerozdelenia rozdelí rovnomerne VEC zdroje medzi všetky CAVs napojené na rovnakú gNB.
- GNN_gNB: DDPG agent vykonávajúci akciu rozdelenia zdrojov na úrovni gNBs s využitím GNN tak, ako je popísané v sekcii 5.2.3. To znamená, že pre každé CAV napojené na gNB obdrží rovnaké rozdelenie VEC zdrojov.
- GNN_CAV: DDPG agent vykonávajúci akciu rozdelenia zdrojov na úrovni CAVs s využitím GNN tak, ako je popísané v sekcii 5.2.4. Akcia sa vykonáva pre každé CAV zvlášť.
- NN_CAV: DDPG agent vykonávajúci akciu rozdelenia zdrojov na úrovni CAVs s bez využitia GNN, podobne ako prístup v [27]. Stav tvoria príznaky všetkých CAVs a výstupom agenta sú akcie pre každé CAV. Tento prístup použijeme iba v niektorých grafoch, pretože je obmedzený na konštantný počet gNBs a CAVs.

Mapa použitá pri simuláciách je v porovnaní s mapou na obrázku 7.1 menšia. V mape sme nechali tri vodorovné a tri zvislé ulice, pričom jej rozmer je 200×200 metrov. V oblasti sa nachádza fixný počet CAVs aj gNBs, pričom CAVs si vyberajú náhodný bod na mape a zvolia si najkratšiu trasu ku nemu. Po dorazení do cieľového bodu si vyberú ďalší bod na mape.

Základné nastavenia simulácie sú zobrazené v tabuľke 7.2. Simulačný krok bude trvať 0.5 sekúnd pričom nové úlohy sa budú generovať každých 0.1 sekúnd. Za jeden časový krok sa tak vygeneruje päť úloh. V prostredí budeme mať rôzny počet gNBs, maximálne päť. Nastavenie parametrov gNB aj úloh sme prispôsobili menšiemu prostrediu a menšiemu počtu áut tak, aby v určitých situáciách stále dochádzalo ku problému s nedostatočnými zdrojmi MEC pre všetky CAVs.

7.2.1 Jedna gNB v prostredí

V tejto sekcii analyzujeme dosiahnuté výsledky pri jednej gNB v prostredí, pričom sa agent bude učiť pri fixnom počte áut. V prílohe C sa nachádzajú základné nastavenia jednotlivých algoritmov. Ak sa pri simulácii použili iné hodnoty parametrov, bude to spomenuté ďalej v texte.

Tabuľka 7.3 zobrazuje dosiahnuté výsledky metriky M_{FT}, kedy agent rozhodoval iba o rozdelení zdrojov pomocou rozdelenia času ω_i. Agenta sme učili pri 7, 12 a 15 CAVs v prostredí. Je jasné, že keď je pri trénovaní v prostredí iba jedna gNB, agent sa pretrénuje na odpovedajúci počet CAVs v prostredí. Napriek tomu nám tieto výsledky dokážu ukázať výkonnosť jednotlivých algoritmov.

Pre 7 CAVs v prostredí je pomer zahodenia úloh veľmi blízko nule pre všetky algoritmy. Algoritmus RP dosiahol hodnotu metriky M_{FT} nula na rozdiel od ostatných algoritmov. GNN_gNB a NN_CAV sú o niečo málo horšie, dosahujú 0.001 M_{FT}, čo je stále veľmi nízky pomer zahodenia úloh. Algoritmus NN_CAV je ešte viac horší, ale stále je M_{FT} veľmi nízka. To, že GNN_gNB, NN_CAV a NN_CAV dosahujú horšie výsledky ako naivný prístup RP, je podľa nášho názoru spôsobené tým, že v prostredí je vysoký dostatok MEC zdrojov a stavy, kedy je potrebné urobiť presnejšie rozhodnutie, sú zriedkavé. Pri 12 CAVs v prostredí sa pomer zahodených úloh rapídne zvýši. Prístup RP dosahuje hodnotu metriky M_{FT} až 0.173. Oproti ostatným algoritmom je to vysoké číslo. GNN_gNB dosiahol $M_{\mathrm{FT}} = 0.026$, čo je stále priateľný výsledok. GNN_CAV dosiahol ešte lepší výsledok, a to 0.021, čo je o 0.005 lepšie ako GNN_gNB a o 0.152 lepšie ako RP. Prístup NN_CAV mierne zaostáva, dosahuje $M_{\mathrm{FT}} = 0.042$, čo je až dvojnásobok toho, čo dosahuje GNN_CAV. 15 CAVs v prostredí zhorší výrazne výsledky aj u ostatných algoritmov, aj keď sa počet CAVs zvýšil len o tri. Najhoršiu hodnotu metriky M_{FT}, čo je takmer 0.4, dosahuje opäť prístup RP. GNN_CAV opäť dosiahol najlepší výsledok $M_{\mathrm{FT}} = 0.2$, čo je relatívne o 13% lepšie ako GNN_gNB, ktorý dosiahol hodnotu metriky 0.23. Rozdiel výsledkov medzi algoritmami umelej inteligencie a RP sa ešte viac prehĺbil.

Z uvedených výsledkov môžeme konštatovať, že v situáciach, v ktorých sú

N_{CAV}	RP	GNN_gNB	GNN_CAV	NN_CAV
7	**0.000**	0.001	0.002	0.001
12	0.170	0.026	**0.021**	0.042
15	0.390	0.230	**0.200**	0.273

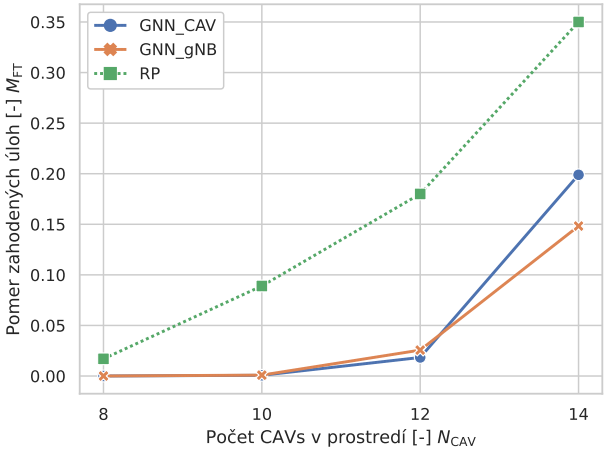
Tabuľka 7.3: Pomer nespracovania úloh (M_{FT}) algoritmov pri rôznom počte CAVs pri vykonávaní akcie rozdelenia zdrojov ω_i

MEC zdroje dostačujúce pre potreby CAVs, je minimálny rozdiel medzi algoritmami, pričom najlepší výsledok dokáže dosiahnuť aj naivný prístup RP (prípad so 7 CAVs). Keď sa však nároky zvýšia, RP prístup je veľmi neoptimálny, pričom ostatné prístupy dosahujú oveľa lepšie výsledky, pričom GNN_CAV dokáže byť najlepší. Prístup NN_CAV dosahuje najhoršie výsledky spomedzi algoritmov využívajúcich strojove učenie. Napriek tomu, že obsahuje dvakrát viac neurónov, sme s nim dosiahli takéto výsledky. Myslíme si, že práve to, že jeho stav tvoria všetky CAVs v prostredí, a to, že vykonáva akcie pre všetky CAVs naraz, si vyžaduje oveľa väčšiu NN a viac krokov na natrénovanie, aby sa výsledkami priblížil navrhnutým prístupom.

Následne sa pozrieme na situáciu, v ktorej agent vykonáva spojito akciu určujúcu latenciu ϵ_i aj rozdelenie zdrojov ω_i. V tabuľke 7.4 sú znázornené výsledky M_{FT} a M_{L} jednotlivých algoritmov. Algoritmus RP očakávane dosahuje rovnaké hodnoty M_{FT} a hodnoty priemernej latencie M_{L} sú najnižšie spomedzi všetkých algoritmov. Ale ako vidíme, je to za cenu vysokého pomeru nespracovania úloh ku spracovaným úlohám. Keď sa pozrieme na výsledky algoritmu GNN_gNB, v prípade 7 CAVs v prostredí dosahuje takmer nulový pomer zahodenia úloh, pričom priemerná latencia je 0.052 sekúnd. Pri viacerých CAVs v prostredí (12 a 15) je latencia 0.09 sekúnd, čo je maximálna priemerná latencia, o ktorej dokáže agent rozhodnúť, pretože si nechávame rezervu 0.01 sekúnd. Naviac sa výsledky pre M_{FT} zhoršili pri 12 CAVs oproti výsledkom zobrazených v tabuľke 7.3, kedy agent vykonával iba jednu akciu. V prípade algoritmu NN_CAV je situácia podobná. Agent dosiahol o trochu lepšie výsledky pri 7 CAVs ako GNN_gNB ale pri 12 a 15 CAVs dosahuje M_{L} okolo 0.009, pričom výsledok pri 12 CAVs je až o 64% horší, ako keď agent vykonával iba akciu pre určenie ω_i. GNN_CAV dosiahol najnižšiu latenciu pri 7 CAVs spomedzi algoritmov využívajúcich strojové učenie, ale zároveň dosiahol aj najhoršiu hodnotu $M_{\text{FT}} = 0.004$, čo je zároveň stále nízka hodnota. Aj tento algoritmus sa zhoršil pri 12, a dokonca aj pri 15 CAVs oproti výsledkom, kedy vykonával iba akciu rozdelenia zdrojov. Pri 12 CAVs je jeho M_{FT} výsledok takmer dvojnásobne horší ako v prípade algoritmu GNN_gNB, pričom

N_{CAV}	RP		GNN_gNB		GNN_CAV		NN_CAV	
	M_{FT}	M_{L}	M_{FT}	M_{L}	M_{FT}	M_{L}	M_{FT}	M_{L}
7	0.000	0.048	0.003	0.052	0.004	0.049	0.002	0.051
12	0.173	0.073	0.028	0.090	0.043	0.084	0.069	0.089
15	0.390	0.082	0.228	0.090	0.213	0.090	0.271	0.090

Tabuľka 7.4: Pomer nespracovania úloh (M_{FT}) a priemernej latencie (M_{L}) pri rôznom počte CAVs pri vykonávaní akcie $a = (\omega_i, \epsilon_i)$



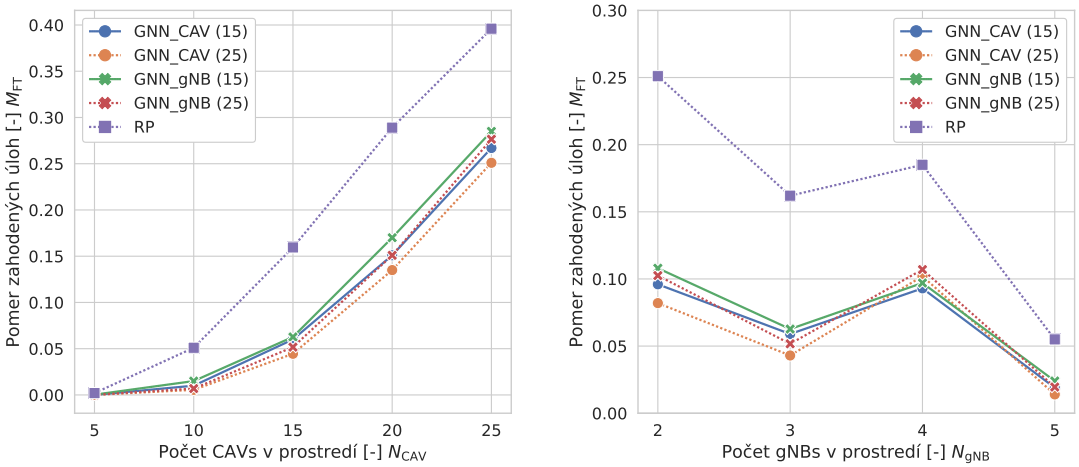
Obr. 7.6: Vplyv počtu CAVs na M_{FT} - agenti natrénovaný pri 12 CAVs a jednou gNB v prostredí

$M_{\mathrm{L}} = 0.084$. Pri 15 CAVs dosiahol GNN_CAV najlepší výsledok zo všetkých algoritmov, $M_{\mathrm{FT}} = 0.213$.

Aj keď sú agenti pretrénovaný, vykonali sme aj testovanie pri rôznych počtoch CAVs, ako pri ktorých boli agenti učení. Na obrázku 7.6 je znázornený príklad dosiahnutých výsledkov GNN_gNB a GNN_CAV, ktorý boli učený pri 12 CAVs. Oproti RP algoritmu dosahuju oveľa lepší výsledok. Prekvapivo pri 8 aj 10 CAVs dosahujú hodnotu metriky M_{FT} blízku nule, zatiaľ čo pre RP pri 10 CAVs je $M_{\mathrm{FT}} = 0.09$. Ďalej môžeme pozorovať, že algoritmus GNN_CAV je pri 12 CAVs v prostredí mierne lepší ako GNN_gNB, pri 14 CAVs má naopak GNN_gNB výraznejšie lepší výsledok, až o 25% nižšiu hodnotu M_{FT} oproti GNN_CAV. Napriek tomu GNN_CAV dosahuje nižšiu hodnotu M_{FT} o 0.15 ako algoritmus RP.

7.2.2 Viac gNBs v prostredí

Agentov sme ďalej trénovali v prostredí s tromi gNBs. V prostredí tak vznikali situácie, kedy sa počet pripojených CAVs na odpovedajúcu gNB s časom menil a agent mal tak možnosť zovšeobecniť svoje rozhodovanie. Agentov sme trénovali



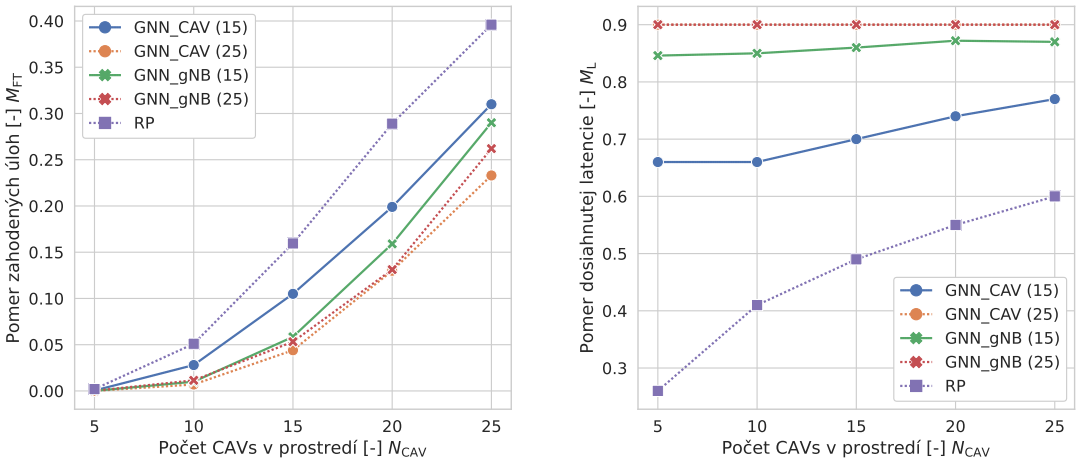
(a) (b)

Obr. 7.7: Dosiahnuté výsledky agentov vykonávajúcich akciu rozdelenia zdrojov $a = \omega$, (a) zobrazuje vplyv počtu CAVs na M_{FT}, (b) zobrazuje vplyv počtu gNBs na M_{FT}

pri 15 a 25 CAVs v prostredí. V tejto situácii sme už netrénovali agenta NN_CAV, pretože je obmedzený na fixný počet CAVs v prostredí. V grafoch budeme porovnávať výsledky toho istého algoritmu, ktorý bol trénovaný pri rôznom počte CAVs. Aby sme ich vedeli od seba odlíšiť, do zátvorky budeme písať za pomenovanie algoritmu číslo, ktoré vyjadruje počet CAVs v prostredí pri trénovaní, napríklad GNN_CAV(15).

Dvojica grafov 7.7a a 7.7b zobrazuje výsledky vplyvu počtu CAVs a gNBs na metriku M_{FT}, kedy agenti vykonávajú iba akciu rozdelenia zdrojov ω_i. V grafe 7.7a vidíme, že algoritmus RP dosahuje výrazne horšie výsledky od ostatných porovnávaných algoritmov, napríklad pri 15 CAVs v prostredí dosahuje M_{FT} viac ako 0.15, zatiaľ čo ostatné algoritmy dosahujú hodnotu metriky približne 0.05. Najlepšie výsledky dosahuje algoritmus GNN_CAV(25), ktorý je pri 20 a 25 CAVs lepší od ostatných algoritmov minimálne o 0.015. Prístup GNN_gNB(15) si spomedzi algoritmov využívajúcich strojové učenie vedie najhoršie, ale stále je výrazne lepší ako RP. Napríklad pri 20 CAVs v prostredí dosahuje GNN_gNB(15) $M_{\mathrm{FT}} = 0.17$, pričom algoritmy GNN_CAV(15) a GNN_gNB(25) dosahujú 0.15 a GNN_CAV(25) dosahuje 0.135, čo je až o 0.035 menej. Ďalším pozorovaním je to, že algoritmus GNN_CAV(15) je horší od GNN_gNB(25) pri menšom počte CAVs v prostredí ako 20, ale pri 25 CAVs dosahuje lepší výsledok.

V grafe 7.7b dosahujú všetky algoritmy vyššiu hodnotu metriky M_{FT} pri štyroch gNBs ako pri troch gNBs v prostredí. Domnievame sa, že z dôvodu roz-



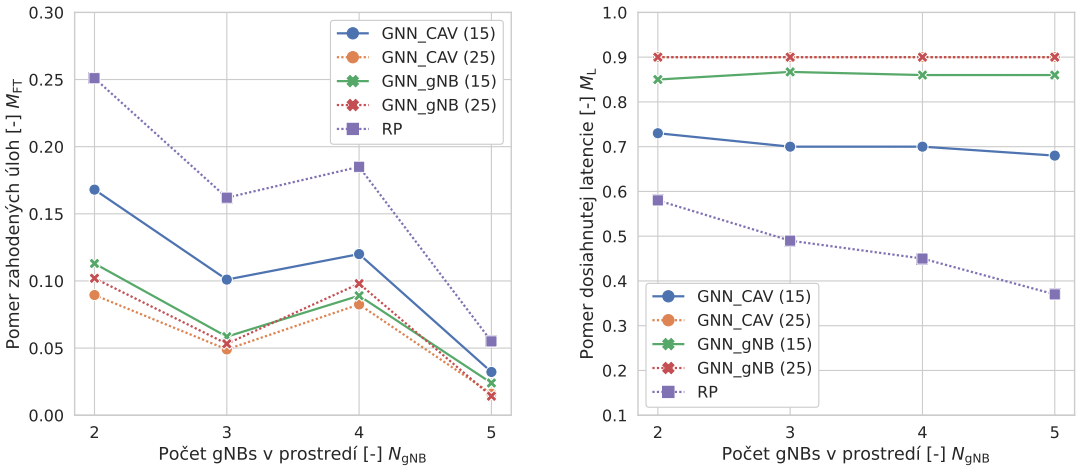
(a) (b)

Obr. 7.8: Dosiahnuté výsledky agentov vykonávajúcich akciu rozdelenia zdrojov a latencie $a = (\omega, \epsilon)$ pre rôzny počet CAVs v prostredí, (a) zobrazuje vplyv počtu CAVs na M_{FT}, (b) zobrazuje vplyv počtu CAVs na M_{L}

miestnenia gNBs, dochádza ku interferencii na úsekoch ciest výrazne viac ako v prípade troch gNBs. Ak budeme ignorovať nárast M_{FT} pri štyroch gNBs v priestore, môžeme povedať, že vo všeobecnosti so zvýšujúcim počtom gNBs klesá dosiahnutá hodnota metriky M_{FT} pri všetkých algoritmoch. Algoritmus RP si vedie opäť výrazne horšie ako ostatné algoritmy, napríklad pri dvoch gNBs v prostredí dosiahol hodnotu metriky M_{FT} 0.25, pričom ostatné algoritmy dosiahli hodnotu v okolí 0.1, čo je približne o 0.15 lepšie. Najlepší výsledok opäť dosiahol algoritmus GNN_CAV(25), akurát pri štyroch gNBs dosiahol horší výsledok ako GNN_CAV(15) a GNN_gNB(15). Výsledky ostatných algoritmov sa prelínajú, ale rozdiel medzi nimi sa postupne znižuje so zvyšujúcim sa počtom gNBs.

Ďalej sa pozrieme na situácie, kedy agenti vykonávali akciu rozdelenia zdrojov v podobe určenia rozdelenia času ω spolu s určením rezervy ϵ teda latencie. Na grafe 7.8a je zobrazená závislosť počtu CAVs na metriku M_{FT}. Výsledky sú veľmi podobné výsledkom v grafe 7.7a, ale aj tak vieme identifikovať rozdiely. V provom rade sa výsledky algoritmu GNN_CAV(15) zhoršili, napríklad pri 15 CAVs dosahuje $M_{\mathrm{FT}} = 0.10$, pričom ostatné algoritmy využívajúce strojové učenie dosahujú hodnotu metriky približne 0.05. Pri 20 CAVs v prostredí je hodnota metriky pre algoritmus GNN_gNB(25) takmer totožná s hodnotou, ktorú dosiahol algoritmus GNN_CAV(25). Algoritmus GNN_CAV(25) dosahuje najlepšie výsledky spomedzi porovnávaných algoritmov.

Graf 7.8b znázorňuje dosiahnuté hodnoty metriky M_{L} pri výsledkoch z grafu



(a) (b)

Obr. 7.9: Dosiahnuté výsledky agentov vykonávajúcich akciu rozdelenia zdrojov a latencie $a = (\omega, \epsilon)$ pre rôzny počet gNBs v prostredí, (a) zobrazuje vplyv počtu gNBs na M_{FT}, (b) zobrazuje vplyv počtu gNBs na M_{L}

7.8a. Vo všeobecnosti hodnota M_{L} rastie so zvyšujúcim sa počtom CAVs v prostredí. Najnižšie hodnoty metriky dosiahol algoritmus RP, od 0.26 pri 5 CAVs až po 0.6 pri 25 CAVs. Spomedzi algoritmov využívajúcich strojové učenie dosiahol prístup GNN_CAV(15) najnižšiu hodnotu metriky M_{L}. Treba brať do úvahy, že jeho výsledky metriky M_{FT} sú najhoršie. Vyzerá to tak, že agent sa nevedel vhodne naučiť prerozdeľovať zdroje a určovať čo najnižšiu latenciu. V prípade algoritmu GNN_gNB(15) sa hodnota metriky M_{L} pohybuje okolo 0.85, čo je o 0.05 menej ako najväčší možný $t_{\mathrm{soft\_deadline},i}$. Algoritmy GNN_CAV(25) a GNN_gNB(25) dosahujú hodnotu $M_{\mathrm{L}} = 0.9$, čo je totožné s maximálnou hodnotou pre $t_{\mathrm{soft\_deadline},i}$. To aj vysvetľuje, prečo sú tieto algoritmy lepšie oproti ostatným. Myslíme si, že pri 25 CAVs v prostredí agent nemal veľmi možnosť sa naučiť rozhodovať o latencii, pretože toľko CAVs spôsobilo to, že sa veľmi často strácali úlohy a nevznikali situácie, v ktorých by bolo dostatok zdrojov pre všetky pripojené CAVs na gNB, a tak agent nedostával odmenu podľa latencie.

V grafoch 7.9a a 7.9b je zobrazená závislosť metrík M_{FT} a M_{L} od počtu gNBs v prostredí. V grafe 7.9a nastáva rovnaká situácia pri štyroch gNBs ako v grafe 7.7b. Algoritmus GNN_CAV(15) dosahuje výraznejšie horšie výsledky od ostatných algoritmov využívajúcich strojové učenie, napríklad pri dvoch gNBs v prostredí dosahuje GNN_CAV(15) v priemere o 0.07 horší výsledok. Algoritmus GNN_CAV(25) dosahuje najlepšie výsledky pri dvoch až štyroch gNBs, akurát pri počte päť gNBs dosahuje algoritmus GNN_gNB(25) rovnakú hodnotu met-

riky M_{FT}.

Z grafu 7.9b vidno klesanie metriky M_{L} s narastajúcim počtom gNBs pre prístupy RP a GNN_CAV(15). Opäť najnižšiu hodnotu metriky M_{L} dosahuje RP a za ním nasleduje GNN_CAV(15), ktorý dosahuje M_{L} okolo 0.7. Pre algoritmus GNN_gNB(15) je M_{L} stabilná na hodnote 0.86, zatiaľ čo pre algoritmy GNN_gNB(25) a GNN_CAV(25) nastáva rovnaká situácia ako v grafe 7.8b kedy ich hodnoty metriky M_{L} sú rovné 0.9 pre všetky uvažované počty gNBs.

Vo všeobecnosti dosahujú lepší výsledok algoritmy, ktoré boli učené pri 25 CAVs v prostredí. Dokonca dosahujú lepší výsledok pri 15 CAVs v prostredí ako algoritmy, ktoré boli pri 15 CAVs v prostredí učený. Domnievame sa, že pri 25 CAVs dokáže vzniknúť viac rôzných stavov v prostredí, a tak je naučený agent robustnejší a dokáže lepšie zovšeobecniť svoje správanie. Relatívne zlý výsledok agenta GNN_CAV(15) v prípadoch, kedy sa rozhodoval aj o určení ϵ, môže mať za následok aj nevydarené trénovanie agenta. Vo všeobecnosti učenie GNN_CAV bolo viac nestabilné ako učenie GNN_gNB. Úspešné zníženie latencie bez vplyvu na zhoršenie M_{FT} sa nám podarilo dosiahnuť iba v prípade algoritmu GNN_gNB(15), pričom sme latenciu znížili v priemere o 5%. Algoritmy učené pri 25 CAVs sa nenaučili pracovať s latenciou. Ako posledné môžeme poznamenať to, že maximálny rozdiel M_{FT} medzi GNN_CAV(25) a GNN_gNB(25) je 0.025 v prospech GNN_CAV(25), ktorý ako sme ukázali, dosahuje najlepšie výsledky metriky M_{FT} v rôznych simulovaných prípadoch.

8 Záver

V práci sme analyzovali momentálny stav MEC prevažne v spojitosti s CAVs a v oblasti rozdeľovania komunikačných a výpočtových zdrojov v MEC a identifikovali nedostatky súčasných prístupov a algoritmov. V prvom rade sme objavili nedostatok pri plánovaní trasy CAV v podobe nebrania do úvahy výpočtové zdroje MEC pri výbere trasy. Vážny nedostatok sme adresovali lexikografickým A* algoritmom, ktoré spojito berú do úvahy výpočtové aj komunikačné zdroje MEC pri výbere vhodnej trasy. Zároveň sme brali do úvahy aj dĺžku trasy pri výbere, ale s menšou prioritou ako dodržanie úspešnosti výpočtu úlohy do požadovaného času.

Identifikovali sme aj nedostatky pri prideľovaní zdrojov MEC medzi napojených používateľov. Buď riešenia opäť nebrali do úvahy spojito komunikačné aj výpočtové zdroje, alebo bol scenár vzdialený od reálneho použitia, ako napríklad riešenia pre fixný počet CAVs a gNBs v prostredí. Rozhodli sme sa použiť GNN pre extrakciu príznakov a DDPG algoritmus na prideľovanie zdrojov MEC na riešenie týchto nedostatkov.

Na to, aby sme overili naše algoritmy, sme vytvorili simulačné prostredie, respektíve sadu nástrojov pomocou IISMotion projektu v Python programovacom jazyku. Sada nástrojov je robená tak, aby sa dala použiť pri rôznych scenároch a dorobiť vlastnú funkcionalitu podľa potreby. Následne sme algoritmy implementovali a vyhodnocovali ich v rôznych scenároch.

Náš prístup plánovania trasy A* algoritmom sme porovnali so state-of-the-art prístupmi, v ktorých sme ukázali, že náš prístup je vo viacerých prípadoch a nastavení parametrov simulácie najlepší z hľadiska úspešnosti výpočtu úloh do ich požadovaného času. V niektorých prípadoch náš prístup dosiahol až o približne 60% lepšiu úspešnosť výpočtu úloh ako ostatné algoritmy. Zároveň sme ukázali, že rovnakú úspešnosť ako pri ostatných prístupoch dokážeme dosiahnuť pri nižšom počte gNBs, čo znamená ušetrenie nákladov pre poskytovateľov MEC služieb.

Algoritmy dynamického prideľovania MEC zdrojov (GNN_gNB

a GNN_CAV) sme porovnali s naivným prístupom rovnomerného prerozdelenia zdrojov (RP) a prístupom využívajúcim veľkú neurónovú sieť (NN_CAV), ktorej veľkosť vstupu je viazaná na fixný počet CAVs v priestore. Ukázali sme, že oba naše prístupy GNN_gNB aj GNN_CAV dosiahli lepšie výsledky v prípade s jednou gNB v simulačnom prostredí, pričom GNN_CAV dosiahol vo všeobecnosti najlepší výsledok. Ďalej sme porovnávali algoritmy GNN_CAV a GNN_gNB, ktoré boli trénované v prostredí s tromi gNBs a 15 alebo 25 CAVs. Algoritmy dosahovali oveľa lepší výsledok v úspešnosti vypočítaných úloh, v niektorých prípadoch sme dokázali znížiť množstvo zahodených úloh až o 69%. Ukázalo sa, že najlepšie si viedli algoritmy trénované pri 25 CAVs v prostredí, pričom vo všetkých prípadoch dosahoval algoritmus GNN_CAV učený pri 25 CAVs v prostredí najlepší výsledok v metrike M_{FT}. Na druhej strane v prípadoch, kedy sa agent mal naučiť aj minimalizovať latenciu spracovania úlohy, sme nedokázali zabezpečiť zníženie latencie bez vplyvu na úspešnosť výpočtu úloh. Buď agent neznižoval latenciu, alebo latenciu znížil, ale zároveň sa znížila aj úspešnosť výpočtu úloh.

Oba algoritmy dokážu ušetriť náklady spojené s infraštruktúrou a zároveň dosiahnuť požadovanú kvalitu služieb pre CAV. Obzvlášť riešenie dynamického priradenia zdrojov sa dá využiť aj pre iných používateľov, ako sú CAVs. Simulačné prostredie, respektíve sada nástrojov, sa dá ďalej rozširovať a používať pri výskumoch v oblastí MEC.

Pri navrhnutých algoritmoch tu vzniká potenciál na ich rozšírenie. Pri prístupe plánovania trasy sme rátali s dokonalou predpoveďou pozície vozidla v budúcnosti. Vozidlo sa však môže stretnúť s neočakávanými udalosťami, napríklad prebehnutie chodcu pred vozidlom, čo spôsobí opozdenie vozidla. Náš algoritmus neberie do úvahy takéto udalosti a nerieši pripadnú zmenu predpovede pozícií vozidla v budúcnosti. Algoritmus by sa tak mohol rozšíriť o možnosť realokácie zdrojov v prípade, ak predpovede vozidla nebudú dodržané. Zároveň sme uvažovali o fixnom určení výpočtových zdrojov (fixne určené ω_i) pri rezervácii MEC zdrojov, a to nemusí byť vždy optimálne. V prípade dynamického rozdeľovania MEC zdrojov sa nám nepodarilo minimalizovať latenciu bez vplyvu na úspešnosť výpočtu úloh, čo môže byť predmetom pre budúce práce. Zároveň sme nevyužili úplný potenciál GNN. Ich využitie nám iba umožnilo použiť riešenie na rôzny počet CAVs a gNBs. Zaujímavým sa javí myšlienka využitia GNN na spojitý výber gNB, na ktorú sa CAV pripojí, výber gNB na ktorom sa úloha bude počítať (gNB môže preposlať úlohu na menej vyťaženú gNB cez chrbticovú sieť) a určenie rozdelením zdrojov MEC. V týchto prípadoch by graf obsahoval vzťahy aj s CAVs napojených na susedné gNBs a využitie GNN by tak bolo významnej-

šie. V neposlednom rade by bolo vhodné porovnať navrhnuté algoritmy dynamického prideľovania zdrojov s viac sofistikovanými súčasnými metódami ako je algoritmus rovnomerného prerozdelenia.

Literatúra

1. VOLOŠIN, Marcel; ŠLAPAK, Eugen; BECVAR, Zdenek; MAKSYMYUK, Taras; PETÍK, Adam; LIYANAGE, Madhusanka; GAZDA, Juraj. Blockchain-Based Route Selection With Allocation of Radio and Computing Resources for Connected Autonomous Vehicles. *IEEE Transactions on Intelligent Transportation Systems*. 2023, s. 1–14. Dostupné z DOI: 10.1109/TITS.2023.3255301.
2. *Mobile-edge computing – Introductory Technical White Paper*. Sophia Antipolis, France, 2014. Dostupné tiež z: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf.
3. ZHANG, Yan. *Mobile Edge Computing*. 1. vyd. Springer Cham, 2021. ISBN 978-3-030-83944-4. Dostupné z DOI: https://doi.org/10.1007/978-3-030-83944-4.
4. MAO, Yuyi; YOU, Changsheng; ZHANG, Jun; HUANG, Kaibin; LETAIEF, Khaled B. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials*. 2017, roč. 19, č. 4, s. 2322–2358. Dostupné z DOI: 10.1109/COMST.2017.2745201.
5. ARSHAD, Rabe; ELSAWY, Hesham; SOROUR, Sameh; AL-NAFFOURI, Tareq Y.; ALOUINI, Mohamed-Slim. Handover Management in 5G and Beyond: A Topology Aware Skipping Approach. *IEEE Access*. 2016, roč. 4, s. 9073–9081. Dostupné z DOI: 10.1109/ACCESS.2016.2642538.
6. ZHOU, Sheng; SUN, Yuxuan; JIANG, Zhiyuan; NIU, Zhisheng. Exploiting Moving Intelligence: Delay-Optimized Computation Offloading in Vehicular Fog Networks. *CoRR*. 2019, roč. abs/1902.09401. Dostupné z arXiv: 1902.09401.
7. ASADI, Arash; WANG, Qing; MANCUSO, Vincenzo. A Survey on Device-to-Device Communication in Cellular Networks. *IEEE Communications Sur-*

veys & Tutorials. 2014, roč. 16, č. 4, s. 1801–1819. Dostupné z DOI: 10.1109/COMST.2014.2319555.

8. *Multi-access Edge Computing (MEC); Framework and Reference Architecture*. 2022. Tech. spr. ETSI.
9. MACH, Pavel; BECVAR, Zdenek. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys & Tutorials*. 2017, roč. 19, č. 3, s. 1628–1656. Dostupné z DOI: 10.1109/COMST.2017.2682318.
10. BLANCO, Bego; FAJARDO, Jose Oscar; GIANNOULAKIS, Ioannis; KAFETZAKIS, Emmanouil; PENG, Shuping; PÉREZ-ROMERO, Jordi; TRAJKOVSKA, Irena; KHODASHENAS, Pouria S.; GORATTI, Leonardo; PAOLINO, Michele; SFAKIANAKIS, Evangelos; LIBERAL, Fidel; XILOURIS, George. Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN. *Computer Standards & Interfaces*. 2017, roč. 54, s. 216–228. ISSN 0920-5489. Dostupné z DOI: https://doi.org/10.1016/j.csi.2016.12.007. SI: Standardization SDN&NFV.
11. JIANG, Wei; HAN, Bin; HABIBI, Mohammad Asif; SCHOTTEN, Hans Dieter. The Road Towards 6G: A Comprehensive Survey. *IEEE Open Journal of the Communications Society*. 2021, roč. 2, s. 334–366. Dostupné z DOI: 10.1109/OJCOMS.2021.3057679.
12. WIJETHILAKA, Shalitha; LIYANAGE, Madhusanka. Survey on Network Slicing for Internet of Things Realization in 5G Networks. *IEEE Communications Surveys & Tutorials*. 2021, roč. 23, č. 2, s. 957–994. Dostupné z DOI: 10.1109/COMST.2021.3067807.
13. WIKIPEDIE, Prispievatelia. *Free-space path loss — Wikipedia* [https://en.wikipedia.org/w/index.php?title=Free-space_path_loss&oldid=1080664589]. 2022. cit. 2022-10-1.
14. GLADIŠOVÁ, Iveta; MIHALÍK, Ján. *Modulované signály*. 1. vyd. Technická univerzita v Košiciach, 2016. ISBN 978-80-553-2442-5.
15. MOUSAVI, H.; AMIRI, Iraj S.; MOSTAFAVI, M.A.; CHOON, C.Y. LTE physical layer: Performance analysis and evaluation. *Applied Computing and Informatics*. 2019, roč. 15, č. 1, s. 34–44. ISSN 2210-8327. Dostupné z DOI: https://doi.org/10.1016/j.aci.2017.09.008.

16. PRICE, John; GOBLE, Terry. 10 - Signals and noise. In: MAZDA, Fraidoon (ed.). *Telecommunications Engineer's Reference Book*. Butterworth-Heinemann, 1993, s. 10-1-10–15. ISBN 978-0-7506-1162-6. Dostupné z DOI: https://doi.org/10.1016/B978-0-7506-1162-6.50016-2.

17. WIKIPEDIE, Prispievatelia. *Signal-to-interference-plus-noise ratio — Wikipedia* [https://en.wikipedia.org/wiki/Signal-to-interference-plus-noise_ratio)]. 2022. cit. 2022-10-1.

18. WU, Zonghan; PAN, Shirui; CHEN, Fengwen; LONG, Guodong; ZHANG, Chengqi; YU, Philip S. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*. 2021, roč. 32, č. 1, s. 4–24. Dostupné z DOI: 10.1109/TNNLS.2020.2978386.

19. HE, Shiwen; XIONG, Shaowen; OU, Yeyu; ZHANG, Jian; WANG, Jiaheng; HUANG, Yongming; ZHANG, Yaoxue. An Overview on the Application of Graph Neural Networks in Wireless Networks. *IEEE Open Journal of the Communications Society*. 2021, roč. 2, s. 2547–2565. Dostupné z DOI: 10.1109/OJCOMS.2021.3128637.

20. LILLICRAP, Timothy P.; HUNT, Jonathan J.; PRITZEL, Alexander; HEESS, Nicolas; EREZ, Tom; TASSA, Yuval; SILVER, David; WIERSTRA, Daan. *Continuous control with deep reinforcement learning*. 2019. Dostupné z arXiv: 1509.02971 [cs.LG].

21. CHIUMENTO, Alessandro; BENNIS, Mehdi; DESSET, Claude; PERRE, Liesbet van der; POLLIN, S. Adaptive CSI and feedback estimation in LTE and beyond: a Gaussian process regression approach. *EURASIP Journal on Wireless Communications and Networking*. 2015, roč. 2015. Dostupné z DOI: 10.1186/s13638-015-0388-0.

22. CHA, Narisu; WU, Celimuge; YOSHINAGA, Tsutomu; JI, Yusheng; YAU, Kok-Lim Alvin. Virtual Edge: Exploring Computation Offloading in Collaborative Vehicular Edge Computing. *IEEE Access*. 2021, roč. 9, s. 37739–37751. Dostupné z DOI: 10.1109/ACCESS.2021.3063246.

23. LI, Xin; DANG, Yifan; AAZAM, Mohammad; PENG, Xia; CHEN, Tefang; CHEN, Chunyang. Energy-Efficient Computation Offloading in Vehicular Edge Cloud Computing. *IEEE Access*. 2020, roč. 8, s. 37632–37644. Dostupné z DOI: 10.1109/ACCESS.2020.2975310.

24. HUANG, Xumin; YU, Rong; YE, Dongdong; SHU, Lei; XIE, Shengli. Efficient Workload Allocation and User-Centric Utility Maximization for Task Scheduling in Collaborative Vehicular Edge Computing. *IEEE Transactions on Vehicular Technology*. 2021, roč. 70, č. 4, s. 3773–3787. Dostupné z DOI: 10.1109/TVT.2021.3064426.

25. LIN, Bin; ZHOU, Xian; DUAN, Jianli. Dimensioning and Layout Planning of 5G-Based Vehicular Edge Computing Networks Towards Intelligent Transportation. *IEEE Open Journal of Vehicular Technology*. 2020, roč. 1, s. 146–155. Dostupné z DOI: 10.1109/OJVT.2020.2988645.

26. LI, Ji; GAO, Hui; LV, Tiejun; LU, Yueming. Deep reinforcement learning based computation offloading and resource allocation for MEC. In: *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. 2018, s. 1–6. Dostupné z DOI: 10.1109/WCNC.2018.8377343.

27. YAO, Liang; XU, Xiaolong; BILAL, Muhammad; WANG, Huihui. Dynamic Edge Computation Offloading for Internet of Vehicles With Deep Reinforcement Learning. *IEEE Transactions on Intelligent Transportation Systems*. 2022, s. 1–9. Dostupné z DOI: 10.1109/TITS.2022.3178759.

28. LU, Yanfei; HAN, Dengyu; WANG, Xiaoxuan; GAO, Qinghe. Distributed Task Offloading for Large-Scale VEC Systems: A Multi-agent Deep Reinforcement Learning Method. In: *2022 14th International Conference on Communication Software and Networks (ICCSN)*. 2022, s. 161–165. Dostupné z DOI: 10.1109/ICCSN55126.2022.9817573.

29. WANG, Zhihao; ZHANG, Weijiong; LIU, Bo; JIANG, Dingde; WANG, Feng; ZHANG, Jingyang. A joint and dynamic routing approach to connected vehicles via LEO constellation satellite networks. *Wireless Networks*. 2021. Dostupné z DOI: 10.1007/s11276-021-02712-0.

30. VALLATI, Mauro; SCALA, Enrico; CHRPA, Lukáš. *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. A Hybrid Automated Planning Approach for Urban Real-time Routing of Connected Vehicles. 2021. Dostupné z DOI: 10.1109/ITSC48978.2021.9564838.

31. CHEN, Bo; YUAN, Quan; LI, Jinglin; LU, Jiawei; ZHU, Bichuan. Joint Route Planning and Traffic Signal Timing for Connected Vehicles: An Edge Cloud Enabled Multi-Agent Game Method. In: *2020 International Conference on Space-Air-Ground Computing (SAGC)*. 2020, s. 1–6. Dostupné z DOI: 10.1109/SAGC50777.2020.00012.

32. YUAN, Quan; CHEN, Bo; LUO, Guiyang; LI, Jinglin; YANG, Fangchun. Integrated route planning and resource allocation for connected vehicles. *China Communications*. 2021, roč. 18, č. 3, s. 226–239. Dostupné z DOI: 10.23919/JCC.2021.03.018.

33. VONDRA, Michal; DJAHEL, Soufiene; MURPHY, John. VANETs signal quality-based route selection in smart cities. 2014, s. 1–8. Dostupné z DOI: 10.1109/WD.2014.7020812.

34. VONDRA, Michal; BECVAR, Zdenek; MACH, Pavel. Vehicular Network-Aware Route Selection Considering Communication Requirements of Users for ITS. *IEEE Systems Journal*. 2018, roč. 12, č. 2, s. 1239–1250. Dostupné z DOI: 10.1109/JSYST.2016.2623762.

35. IEEE Standard for Wireless Access in Vehicular Environments (WAVE) – Multi-Channel Operation. *IEEE Std 1609.4-2016 (Revision of IEEE Std 1609.4-2010)*. 2016, s. 1–94. Dostupné z DOI: 10.1109/IEEESTD.2016.7435228.

36. SUN, Zhenchuan; MO, Yijun; YU, Chen. Graph Reinforcement Learning based Task offloading for Multi-access Edge Computing. *IEEE Internet of Things Journal*. 2021, s. 1–1. Dostupné z DOI: 10.1109/JIOT.2021.3123822.

37. HE, Ziyan; WANG, Liang; YE, Hao; LI, Geoffrey Ye; JUANG, Biing-Hwang Fred. Resource Allocation based on Graph Neural Networks in Vehicular Communications. In: *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. 2020, s. 1–5. Dostupné z DOI: 10.1109/GLOBECOM42002.2020.9322537.

38. CHEN, Tianrui; ZHANG, Xinruo; YOU, Minglei; ZHENG, Gan; LAMBOTHARAN, Sangarapillai. A GNN-Based Supervised Learning Framework for Resource Allocation in Wireless IoT Networks. *IEEE Internet of Things Journal*. 2022, roč. 9, č. 3, s. 1712–1724. Dostupné z DOI: 10.1109/JIOT.2021.3091551.

39. ZHANG, Xiangyu; ZHANG, Zhengming; YANG, Luxi. Joint User Association and Power Allocation in Heterogeneous Ultra Dense Network via Semi-Supervised Representation Learning. *CoRR*. 2021, roč. abs/2103.15367. Dostupné z arXiv: 2103.15367.

40. RUSSELL, Stuart J.; NORVIG, Peter. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2002. ISBN 0137903952.

Zoznam skratiek

CAV Prepojené autonómne vozidlo (z angl. Connected Autonomous Vehicle).

DDPG Deep Deterministic Policy Gradient.

DQN Deep Q Network.

DRL Hlboké učenie posilňovaním (z angl. Deep Reinforcement Learning.

GCN Grafová konvolučná sieť (z angl. Graph Convolutional Network).

GNN Grafová neurónová sieť (z angl. Graph Neural Network).

LTE Long Term Evolution.

MADDPG Multi-agent Deep Deterministic Policy Gradient.

MCC Mobile Cloud Computing.

MEC Viacprístupové výpočty na hrane (z angl. Multi-access Edge Computing).

MPNN Message-passing Neural Network.

NN Neurónová sieť (z angl. Neural Network).

RB Zdrojové bloky (z angl. Resource Blocks).

RL Učenie posilňovaním (z angl. Reinforcement Learning.

TD3 Twin Delayed Deep Deterministic Policy Gradient.

VEC Vehicular Edge Computing.

Zoznam príloh

A Použitá notácia

u	u-té CAV
v	rýchlosť CAV
σ	segment trasy
task_i	i-tá úloha
D_i	veľkosť požiadavky úlohy i
I_{i}	počet inštrukcií úlohy i v MI
$I_{\mathrm{CPU},s}$	výpočtový výkon CPU s-tej gNB v IPS
$N_{\mathrm{RB},s}$	počet zdrojových blokov s-tej gNB v IPS
$I_{\mathrm{CPU},u,s}$	priradený výkon CPU v IPS s-tou gNB u-tému CAV
$n_{\mathrm{RB},u,s}$	priradený počet zdrojových blokov s-tou gNB u-tému CAV
P_{t}	vysielací výkon
f_{t}	nosná frekvencia
$t_{\mathrm{deadline},i}$	maximálna požadovaná latencia úlohy i
$t_{\mathrm{soft\_deadline},i}$	latencia úlohy i, ktorú sa snažíme dodržať
$t_{\mathrm{uplink},i}$	komunikačný čas nahrávania úlohy i
$t_{\mathrm{process},i}$	čas spracovávania úlohy i na CPU
$t_{\mathrm{soft\_uplink},i}$	komunikačný čas nahrávania úlohy i, ktorý sa snažíme dodržať
$t_{\mathrm{soft\_process},i}$	čas spracovávania úlohy i na CPU, ktorý sa snažíme dodržať
$t_{\mathrm{total},i}$	celkový čas spotrebovaný úlohou i
ω_i	faktor rozdelenia času úlohy na komunikačný a výpočtový čas
ϵ_i	faktor časovej rezervy
$t_{\mathrm{total},i}$	celkový čas spotrebovaný úlohou i
Δt_{T}	čas simulačného kroku
t_{req}	čas zarezervovania zdrojov
t_{rel}	čas uvoľnenia rezervovaných zdrojov
v_i	i-tý vrchol grafu
v_{destin}	cieľový vrchol grafu (koniec trasy)
v_{orig}	počiatočný vrchol grafu (začiatok trasy)

P_k	k-tá trasa
P_f	najrýchlešia trasa
Ω_u	množina trás zvolená pre výber
$T_{P,\text{fastest}}$	čas najkratšej trasy
δ_u	faktor určujúci maximálny požadovaný čas trasy
$T_{P,\max}$	maximálny požadovaný čas trasy
T_{pr}	čas vybratej trasy
T_{FPR}	čas vybratej trasy podľa FPR prístupu
N_{FT}	počet zahodených úloh
N_{GT}	počet vygenerovaných úloh
N_{CT}	počet vypočítaných úloh
$\mathcal{T}_{comp}$	množina vypočítaných úloh
M_{FT}	pomer zahodených úloh
M_{RP}	pomer predĺženia trasy
M_{L}	pomer skrátenia latencie
$\beta_u(t)$	počet bitov na symbol pre CAV u
$\rho_u(t)$	rýchlosť kódovania pre CAV u
$f_{\text{s}}(t)$	modulačná rýchlosť
N_{CAV}	počet CAVs v simulovanom prostredí
N_{gNB}	počet gNBs v simulovanom prostredí
D_{O}	dĺžka trasy od jej začiatku po daný vrchol v_i
D_{H}	vzdialenosť z vrchola v_i do v_{dest}
CC_{missing}	kumulatívna cena chýbajúcich segmentov
CC_{RB}	kumulatívna cena rezervovaných zdrojových blokov
$\mathbf{x}_u$	vektor príznakov u-tého CAV
$sinr_u$	nameraná SINR hodnota pre u-té CAV
$sinr_{\text{old},u}$	predchádzajúca nameraná SINR hodnota pre u-té CAV
G_u	graf reprezentujúci stav u-tého CAV
a_u	akcia agenta pre u-té CAV
$r_u(t)$	odmena agenta vzhľadom na u-té CAV v čase t
$L_s(t)$	priemerný pomer dosiahnutej latencie na s-tej gNB v čase t
ρ	faktor aktualizácie *target* sietí
γ	faktor zľavnenia odmeny agenta
σ	štandardná odchýlka akcie pri trénovani

B Dôkaz konzistentnosti metriky A*

Na dokázanie toho, že heuristická funkcia navrhnutého A* algoritmu hľadania cesty je konzistentná, je potrebné ukázať vlastnosť trojuholníkovej nerovnosti a funkcia sa vyhodnotí ako 0 v cieľovom vrchole $v\mathrm{dest}$ [40], ako je vyjadrené v (B.1):

$$\begin{aligned} h(v_i) &\leq c(v_i, v_{\mathrm{desc}}) + h(v_{\mathrm{desc}}) \\ h(v_{\mathrm{dest}}) &= 0, \end{aligned} \tag{B.1}$$

kde v_{desc} je akýkoľvek vrchol nasledujúci za v_i, h je heuristická funkcia z 5.8 a $c(v_i, v_{\mathrm{desc}})$ je cena za dosiahnutie v_{desc} z vrchola v_i vyjadrené formou usporiadanej trojice $(CC_{\mathrm{missing}}, CC_{\mathrm{RB}}, distance)$.

Hodnoty CC_{missing} a CC_{RB} sú vždy rovne 0, preto tieto komponenty usporiadanej trojice určujúce cenu, spĺňajú vlastnosť trojuholníkovej nerovnosti. Zostáva tak ukázať, že posledný člen $distance$ tiež spĺňa túto podmienku. Na to je potrebné vypočítať vzdialenosť medzi v_i a v_{destin}. V našom prípade, v ktorom uvažujeme o manhattanovskej mape, je zároveň aj Manhattanovská vzdialenosť najkratšou možnou vzdialenosťou medzi dvoma vrcholmi a jej odhad (v podobe manhattanovskej vzdialenosti) je tým pádom stále $\leq c(\cdot)$. Preto komponent vzdialenosti tiež spĺňa rovnicu B.1 vo forme $h(v_i) - h(v_{\mathrm{desc}}) \leq c(v_i, v_{\mathrm{desc}})$.

C Konfigurácia DRL agentov

Parameter	Hodnota
počet krokov trénovania	100000
aktualizácia váh	po každom kroku
GCN vrstva	128 (neurónov)
NN vrstvy	[512, 512] (neurónov)
aktivačná funkcia v skrytých vrstvách	ReLU
optimalizátor	Adam
rýchlosť učenia	0.0005
veľkosť zásobníka	20 000
minimálny počet vzoriek	4 000
veľkosť dávky	32
počet epoch	1
ρ	0.9
γ	0.9
počiatočná hodnota σ	0.1
minimálna hodnota σ	0.05
faktor klesania σ	0.9999

Tabuľka C.1: Základné nastavenia algoritmu GNN_gNB

Parameter	Hodnota
počet krokov trénovania	100000
aktualizácia váh	po každom kroku
GCN vrstva	128 (neurónov)
NN vrstvy	[512, 512] (neurónov)
aktivačná funkcia v skrytých vrstvách	ReLU
optimalizátor	Adam
rýchlosť učenia	0.0005
veľkosť zásobníka	20 000
minimálny počet vzoriek	4 000
veľkosť dávky	32
počet epoch	1
ρ	0.9
γ	0.9
počiatočná hodnota σ	0.1
minimálna hodnota σ	0.05
faktor klesania σ	0.9999

Tabuľka C.2: Základné nastavenia algoritmu GNN_CAV

Parameter	Hodnota
počet krokov trénovania	100000
aktualizácia váh	po každom kroku
NN vrstvy	[1024, 1024] (neurónov)
aktivačná funkcia v skrytých vrstvách	ReLU
optimalizátor	Adam
rýchlosť učenia	0.0005
veľkosť zásobníka	20 000
minimálny počet vzoriek	1 000
veľkosť dávky	128
počet epoch	1
ρ	0.9
γ	0.9
počiatočná hodnota σ	0.1
minimálna hodnota σ	0.05
faktor klesania σ	0.99995

Tabuľka C.3: Základné nastavenia algoritmu NN_CAV