FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Optimalizace vyvažování systémů sdílených kol
Student:	Kristián Kuľka
Vedoucí:	doc. Ing. Michal Jakob, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Vyvažování systémů sdílených kol představuje cílené přesunování jízdních kol v rámci systému tak, aby byla zajištěna dostupnost jízdních kol v místech a časech, kde je po nich poptávka. Efektivní vyvažování systémů sdílených kol má významný vliv na kvalitu a náklady poskytované služby sdílených kol. Cílem bakalářské práce je vytvořit a implementovat algoritmus, který bude optimalizovat přesuny jízdních kol vzhledem k zadaným hodnotícím kritériím služby sdílených kol. V rámci zpracování práce proveďte následující:
1) Seznamte se s problematikou vyvažování systémů sdílených kol (dále vyvažování)
2) Namodelujte vyvažování jako optimalizační problém s definovanými hodnotícími kritérii
3) Navrhněte algoritmus pro řešení definovaného optimalizačního problému vyvažování
4) Navržený algoritmus pro optimalizaci vyvažování implementujte
5) Implementovaný algoritmus vyhodnoťte na realistických testovacích instancích problému vyvažování

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 11. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Optimalizace vyvažování systémů sdílených kol

Kristián Kuľka

Katedra softwarového inženýrství
Vedúci práce: doc. Ing. Michal Jakob, Ph.D.

16. mája 2019

Poďakovanie

V prvom rade by som sa chcel poďakovať vedúcemu práce, doc. Ing. Michalovi Jakobovi, Ph.D., za zaujímavú tému, pomoc a príkladné vedenie. Ďalej Monike Maďarovej, svojej sestre, Lívii Kuľkovej a rodine za podporu vo všetkých smeroch počas celého štúdia a v neposlednom rade taktiež všetkým tým, ktorí mi pri písaní práce pomohli prekonať rôzne prekážky.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 16. mája 2019 .....

Odkaz na túto prácu

Kuľka, Kristián. *Optimalizace vyvažování systémů sdílených kol.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Systémy zdieľaných bicyklov sa stávajú stále viac dôležitejšou súčasťou verejnej dopravy. Flexibita, ktorú tieto systémy ponúkajú, je spojená s nepravidelnými a nesymetrickými jazdami, ktoré spôsobujú nevyváženie systému. Vo výsledku teda v niektorých oblastiach mesta chýbajú bicykle a naopak iné oblasti sú preplnené.

Táto bakalárska práca sa zaoberá problémom vyvažovania tzv. voľne plavúceho systému zdieľaných bicyklov, ktorý rieši pomocou odmeňovacích stratégií, s cieľom motivovať užívateľov, aby na konci svojej jazdy zaparkovali bicykle do vyťažených oblastí.

Cieľom práce je definovať problém vyvažovania, navrhnúť odmeňovacie stratégie, ktoré budú tento problém riešiť, implementovať tieto stratégie a vyhodnotiť ich efektivitu na reálnych instanciách problémov vyvažovania.

Výsledkom práce je návrh a implementácia dvoch odmeňovacích stratégií a taktiež plne konfigurovateľný a funkčný simulačný systém, na ktorom boli obe stratégie vyhodnotené a ktorý taktiež dokáže graficky zvizualizovať priebeh simulácie. Na záver prebehla analýza výsledkov odmeňovacích stratégií a diskusia k možnému zlepšeniu.

Kľúčové slová systém zdieľaných bicyklov, vyvažovanie, odmeňovanie, voľne plavúci, zdieľaná mobilita

Abstract

Bicycle-sharing systems are becoming more important part of public transportation. Flexibility, which they offer, comes with irregular and asymmetric flow patterns of the bikes which causes imbalance problems. As a result of these problems some zones of the city are nearly without any bicycles and the other ones are overfilled with them.

This bachelor thesis deals with the problem of rebalancing free-floating bicycle-sharing systems. It solves this problem using incentive strategies with the aim of motivating users to change their target destination so that they park bicycles in the busy or empty zones.

The goal of this thesis is to define the problem of rebalancing, design and implement incentive strategies which will solve this problem and evaluate effectivity of these strategies based on the realistic instances of rebalancing problems.

The result of the thesis is the design and implementation of 2 incentive strategies, fully functional and configurable simulation system which can evaluate incentive strategies and graphically visualise process of simulation. At the end, the results of effectivity of incentive strategies were analysed and there was also discussion about possible improvements.

Keywords bicycle-sharing system, rebalancing, incentivization, free-floating, shared mobility

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Systém zdieľaných bicyklov (BSS) je služba ponúkajúca voľný prenájom bicyklov v mestách a to najčastejšie pomocou mobilnej aplikácie. Takéto systémy ponúkajú flexibilnú a ekologickú alternatívu ku klasickej verejnej doprave zajisťovanej veľkými a energeticky náročnými vozidlami podľa pravidelného cestovného poriadku. V posledných rokoch narásta obľúba BSS v mestách po celom svete [3]. Najčastejšie si užívatelia prenajímajú bicykle na dopravu na krátku vzdialenosť alebo za účelom relaxácie/turizmu. Výhoda BSS je obrovská flexibilita v používaní bicyklov bez nutnosti rezervácie a plánovania cesty.

Existujú dva druhy BSS. Stanicovo orientovaný (SBSS) a tzv. voľne plávúci (FBSS). V prípade SBSS si môže zákazník vypožičiať a vrátiť bicykel len do staníc, ktoré sú rozmiestnené po meste. Bicykel si zväčša odomkne prostredníctvom mobilnej aplikácie a pri vrátení bicykla do stanice zaplatí poplatok. FBSS ponúka flexiblinejší prístup. Bicykle sú voľne rozmiestnené po meste, pripútané napríklad o klasické stojany na bicykle, stĺpy alebo sú voľne opreté o stojan.

Udržovanie takého systému sa spája s viacero výzvami. Jednou z nich je udržovať rozumné rozdelenie bicyklov po meste aby nedochádzalo k situáciam kedy užívateľ nemá možnosť vrátiť bicykel do stanice, lebo je plná alebo opačný prípad, kedy sa v užívateľovom blízkom okolí nenachádza žiaden voľný bicykel kvôli čomu sa rozhodne nevyužiť služby BSS. FBSS má oproti SBSS výhodu v tom, že viacmenej nemusí riešiť problém nemožnosti vrátenia bicykla, keďže mesto sa prakticky nikdy nenaplní.

Problém nevyváženosti systému vzniká kvôli nesymetrickému dopytu po bicykloch a ich nesymetrickom pohybe počas celého dňa, ktorý je spôsobený napríklad rôznymi udalosťami v meste, veľkým prevýšením niektorých oblastí alebo vplyvom počasia. Taktiež za spomenutie stojí situácia ráno, kedy väčšinu ľudí prichádza do práce, škôl atp., čo znamená, že bicykle sa budú hromadiť v okolí budov týchto inštitúcií alebo situácia v podvečer, kedy bude znova

veľký dopyt po bicykloch na ceste domov.

V prípade, že by sa systém rozhodol neriešiť tento problém, počet zákazníkov, ktorí by využili služby BSS, by mohol klesnúť pod prijateľné minimum; taktiež medzi najväčšie náklady BSS patria práve náklady na vyvažovanie. Z týchto dôvodov je dôležité pre BSS riešiť tento problém efektívne a lacno [4].

Vyváženie systému sa dá dosiahnúť pomocou rôznych stratégií. Operátor systému môže použiť dodávky, ktoré budú jazdiť po meste, pričom budú zbierať bicykle z menej vyťažených oblastí a vykladať ich tam, kde to je potrebné.

Ďalší ekologickejší a lacnejší prístup je ponúkať užívateľom odmenu za to, že si na svoju jazdu vyberú bicykel z menej vyťažených oblastí alebo za to, že na konci svojej jazdy zaparkujú bicykel v oblasti, kde je málo voľných bicyklov. Na základe týchto odmien sa môže užívateľ rozhodnúť zaparkovať bicykel do inej než cieľovej destinácie alebo vypožičiať iný, než najbližší bicykel za cenu extra chôdze.

Cieľ práce

V tejto práci sa venujem problému vyvažovania FBSS, ktorý riešim pomocou odmeňovania užívateľov za odvoz bicyklov do vyťažených oblastí. Medzi hlavné ciele tejto práce patrí nájsť vhodné stratégie, ktoré budú dynamicky určovať výšku ponúkanej odmeny za odvoz bicykla do vyťažených oblastí a taktiež vyhodnotiť ich efektivitu na realistických testovacích instanciách problémov vyvažovania.

Medzi čiastkové ciele patrí vytvoriť simulačný systém, ktorý bude simulovať chod FBSS, na základe nejakej konfigurácie a taktiež vytvoriť grafickú aplikáciu, ktorá dokáže zvizualizovať simuláciu. Pomocou výsledkov simulácie budem vyhodnocovať efektivitu implementovaných stratégií odmeňovania.

Aby systém dokázal určiť vhodnú výšku odmeny pre rôzne akcie užívateľa, je nevyhnutné predikovať budúci dopyt po bicykloch v rôznych oblastiach. Avšak tomuto problému sa v rámci tejto práce nevenujem.

Štruktúra textu

V prvej kapitole popíšem rôzne kategórie problému vyvažovania a existujúce riešenia.

V druhej kapitole predstavím model FBSS, ktorý budem v rámci práce vyvažovať a následne formálne definujem problém vyvažovania. Na konci kapitoly sa venujem analýze odmeňovacích stratégií a požiadavkov kladených na systém.

V tretej kapitole popíšem návrh odmeňovacích stratégií a simulačného systému.

Vo štvrtej kapitole predstavím ako som postupoval pri implementácií odmeňovacích stratégií a simulačného systému.

V poslednej kapitole vyhodnotím efekt odmeňovacích stratégií na realistických instanciách problémov vyvažovania a prevediem rozbor týchto výsledkov.

Kapitola 1

Problém vyvažovania

V tejto kapitole priblížim rôzne kategórie problému vyvažovania BSS a popíšem existujúce riešenia, spolu s ich výhodami a nevýhodami.

1.1 Kategórie problému vyvažovania

Na problém vyvažovania sa dá pozerať zo statického alebo dynamického hľadiska. V statickom sa predpokladá, že systém sa nebude nijak meniť, inými slovami, do aktuálneho stavu systému nijak nezasahuje zákazník. Naopak dynamický pohľad rieši problém vyvažovania v aktívnom meniacom sa systéme.

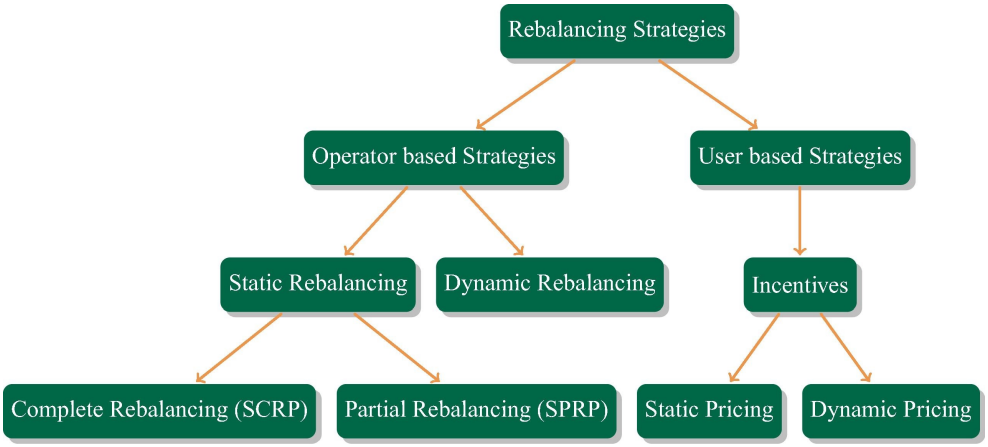
Statický problém riešia operátori BSS v noci, kedy je dopyt po bicykloch minimálny alebo v prípade, že BSS je „vypnutý“, to znamená, že neponúka služby užívateľom. Týmto systém pripravia na ďalší deň, pričom bicykle rozmiestňujú podľa očakaváneho dopytu. Na tento druh problému si musí operátor alokovať manuálnu silu, napr. dodávky alebo jazdcov.

Avšak, vyvažovať BSS v noci nie je častokrát postačujúce, preto je nevyhnutné pre operátora riešiť taktiež dynamický problém vyvažovania, keďže počas dňa sa bicykle nerovnomerne roznášajú po celom meste.

Ako bolo spomenuté v úvode, niektoré problémy vyvažovania sa dajú riešiť pomocou vozidiel alebo odmien pre užívateľov systému, poprípade aj kombináciou týchto možností. Jeden z možných prístupov pri kombinácií týchto stratégií je, že zóny, ktoré sú príliš neatraktívne pre bežného užívateľa napr. kvôli veľkému prevýšeniu by mohli vyvažovať dodávky, pričom ostatné zóny by vyvažovali užívatelia pomocou odmien.

Systém sa taktiež môže snažiť o celkové vyváženie alebo len čiastočné, napríklad nejakej oblasti v blízkosti centra, v prípade kedy nemá dostatok prostriedkov na vyváženie celého systému. Obrázok 1.1 ponúka celkový prehľad možných prístupov.

Pri FBSS sa mesto rozdelí do niekoľkých neprekrývajúcich sa zón. Čas dňa sa rozdelí na niekoľko časových úsekov. Vo väčšine literatúry sa následne vyvažujú jednotlivé zóny práve v týchto časových úsekoch. To znamená, že



Obr. 1.1: Stratégie vyvažovania [1]

vyvážený systém je definovaný ako systém, ktorý má v každej zóne pre každý časový úsek dostatočný počet bicyklov aby pokryl dopyt.

1.2 Prehľad existujúcich riešení

Kvôli veľkému výskytu SBSS sa väčšina literatúry týka tejto problematiky. Vďaka tomu, že zónu FBSS je možné vnímať ako stanicu SBSS, sú častokrát tieto riešenia kompatibilné aj s FBSS.

V nasledujúcich podsekciách popíšem rôzne existujúce riešenia, ktoré riešia problém vyvažovania BSS.

1.2.1 Statické vyvažovanie pomocou dodávok

Na problém vyvažovania BSS pomocou flotily vozidiel sa dá pozerať ako na variantu vehicle routing problem.

Výpočetná komplexita tohoto problému pri FBSS je väčšia než pri SBSS. Na ilustráciu tohoto faktu zvážme následujúci príklad. Nech existuje BSS, ktorý má 100 staníc (alebo stojanov na bicykle) a 200 bicyklov. V prípade SBSS, maximálny počet navštívených miest bude menší alebo rovný počtu staníc. Avšak, pri FBSS to môže byť podstatne viac, keďže každý bicykel môže byť na inom mieste. Na ilustráciu uvažujme, že operátor chce mať v každom stojane práve 2 bicykle. Počet miest, ktoré bude musieť flotila navštíviť, môže byť maximálne rovný súčtu počtu bicyklov a počtu stojanov[1] alebo dvojnásobnému počtu bicyklov, v prípade kedy nie je v systéme dostatok bicyklov aby pokryli žiadúci stav pre každý stojan [1].

[1]Flotila v tomto prípade bude musiet zobrať každý bicykel a navštíviť všetky stanice, čiže kompletný počet miest, ktoré navštívi je rovný $300 = 200 + 100$.

V literatúre sa vyskytujú rôzne objektívne funkcie, ktoré merajú efekt statického úplneho vyvažovania. Niektoré sú zamerané skôr na spokojnosť zákazníka, iné sa naopak snažia primárne minimalizovať náklady operátora BSS. Medzi najvýznamnejšie patrí [1]:

1. cestovné náklady,
2. celkové náklady (cestovné + nakladanie a vykladanie) na prerozdelenie bicyklov,
3. celková absolútna odchýlka od cieľového počtu bicyklov,
4. celkový počet neuspokojených zákazníkov,
5. celkový operačný čas vyvažovacej flotily vozidiel.

V práci [5] autori použili zmiešané celočíselné lineárne programovanie (MILP) a problém SCRP vyriešili optimálne. Avšak v práci uvažujú len jednu dodávku a výpočetná náročnosť, pre 60 navštívených miest, je na štandardnom pc 2 h. Tento prístup je teda priveľmi neefektívny už pre malé alebo stredne veľké FBSS.

V nasledujúcom texte detajlnejšie popíšem prístup [1] ku riešeniu problému statického vyvažovania FBSS pomocou viacerých dodávok, ktorý autori riešia pomocou heuristického algoritmu.

Ako objektívnu funkciu zvolili celkový operačný čas vyvažovacej flotily vozidiel. Týmto zaručujú, že pracovná záťaž pre jednotlivé vozidlá bude rovnomerne rozložená naprieč celou flotilou. V tejto práci nepovoľovali dočasné vykladanie bicyklov do voľných staníc, keďže to zvyšuje komplexitu problému bez výrazneho zlepšenia výsledku.

Každé miesto, ktoré má flotila dodávok navštíviť vnímaju ako uzol v grafe s nejakou mierou nevyváženosti. Ak je rovná kladnému číslu x znamená to, že tam je x prebytok bicyklov. Analogicky, ak je záporná, znamená to, že tam chýba x bicyklov. Každý takýto uzol dekomponujú do x uzlov ktoré majú jednotkovú mieru nevyváženosti, pričom zdieľajú rovnakú polohou. Keďže každý uzol ma jednotkovú nevyváženosť, stačí ak ho flotila dodávok navštívi iba raz. Tento problém sa dá následne označiť ako m-TSP[2] s dodatočnými obmedzeniami. Potom čo nájdu riešenie pre dekomponovaný systém, nahradia každý rozpadnutý uzol pôvodným a tým dostávajú riešenie pre pôvodný problém.

Tento problém následne formulovali ako MILP. Avšak, aj pre malé instancie tohoto problému je táto formulácia výpočetne neriešiteľná. Z tohoto dôvodu navrhli heuristický algoritmus, ktorý je hybridom metód, ktoré sa v anglickej literatúre označujú ako Nested Large Neighborhood Search a Variable Neighborhood Descent.

[2]Bližšie informácia o tomto probléme je možné nájsť v článku [6].

Výhoda tohoto riešenia je, že algoritmus je dostatočne rýchly aj pri väčších instanciách problémov a taktiež dosahuje rozumné hodnoty objektívnej funkcie. Konkrétne pre BSS v meste Chicago, ktorý má 3000 bicyklov, bola hodnota objektívnej funkcie v priemere cca 15 000 sekúnd s 30 vozidlami, ktorých kapacita bola 10 bicyklov.

1.2.1.1 Stručný prehľad ďalších riešení

V práci [7] autori formulujú vyvažovanie ako jednovozidlový kapacitne obmedzený problém vyzdivhnutia a doručenia [8]. Stanice predstavujú uzly grafu, pričom každá stanica má počiatočné naplnenie a cieľové naplnenie. Hranám grafu je pridelená cena, ktorá spĺňa trojuholníkovu nerovnosť. Výsledok algoritmu je trasa spolu s rôznymi akciami nabratia a vyloženia bicyklov. Navrhnutý model je prakticky neriešiteľný problém preto použili relaxáciu a výsledný problém, ktorý sa ukázal ako celočíselne programovanie s exponencionálnym množstvom obmedzení vyriešili pomocou branch-and-cut algoritmu, pomocou ktorého dosahujú suboptimálne výsledky. Avšak, efektivita algoritmu je nepostačujúca v prípade, kedy systém obsahuje viac než 60 staníc.

V práci [9] autori taktiež formulujú vyvažovanie ako jednovozidlový kapacitne obmedzený problém vyzdivhnutia a doručenia. Podobne ako v predošlom prípade, každá stanica má nejaké počiatočné a cieľové naplnenie. Tento problém riešia pomocou iterovaného lokálneho prehľadávania, pričom taktiež používajú nimi navrhnutú heuristiku. Výsledky testovania ukázali, že nimi navrhnutá metóda našla častokrát optimálne riešenia.

1.2.2 Dynamické vyvažovanie pomocou odmien

Pri odmeňovaní užívateľa je dôležité ponúkať vhodne vysokú odmenu za jednotlivé akcie a to tak aby výška odmeny neprekročila predpokladaný zisk, ktorý by akcia systému priniesla.

Medzi najdôležitejšie akcie, za ktoré systém môže ponúkať odmenu patrí: zaparkovanie bicykla do vyťaženej zóny a vypožičanie bicykla z preplnenej zóny namiesto zóny s malým počtom bicyklov.

Výška odmeny závisí od viacerých faktorov a to najmä na predikovanom stave zóny[3] v nejakom časovom úseku, predpokladanom zisku a na odhadnutom užívateľovom modeli, ktorý udáva reakciu pre rôzne vysoké odmeny za rôzne akcie.

V práci [2] autori navrhli systém odmeňovania zákazníkov pomocou odmien za vypožičanie/odvoz bicykla do susedných problémových staníc. Problemové stanice označili ako tie, ktoré sú buď prázdne alebo plné. Systém užívateľovi ponúkne stále len jednu odmenu a to za najbližšiu problémovú stanicu, ktorá je k nemu najbližšie. Zákazník má možnosť požiadať systém

[3]Stav zóny v časovom úseku vypočítam ako počet bicyklov na začiatku časového úseku + počet zaparkovaní bicykla v časovom úseku - dopyt.

o jednu z dvoch akcií: vypožičať alebo vrátiť bicykel. Užívateľa vnímajú ako strategického agenta, ktorý môže klamlivo žiadať o vrátenie bicykla napr. 100 m pred jeho cieľovou destináciou s cieľom získať nejakú odmenu. Celý model vylepšujú pomocou online učenia, pomocou ktorého sa snažia zachytiť trendy v užívateľovom správaní, čo im vo výsledku pomáha lepšie oceniť nejaké stanice; napr. v prípade, že je veľka pravdepodobnosť, že užívateľ vráti bicykel do problémovej stanice A nezávislé na výške odmeny (napr. preto, že tam býva alebo častokrát tam jazdí), výsledná odmena pre túto stanicu bude menšia.

Pri odmeňovaní berú v úvahu rozpočet na vyvažovanie, ktorý má operátor BSS k dispozícii, pričom sa s ním snažia ekonomicky pracovať. Výška výslednej odmeny by mala byť dostatočne vysoká, aby motivovala zákazníka a zároveň nebola príliš veľká vzhľadom k dostupnému rozpočtu a predpokladanému zvyšnému počtu odmien, ktoré ešte len budú ponúknuté v časovom úseku, pre ktorý operátor vystavil tento rozpočet. To zachytáva následujúca rovnica, kde $p \in \{p_0, \ldots, p_K\}$ označuje množinu ponúkaných odmien, B je výška rozpočtu, N je zvyšný počet odmien, ktoré ešte len budú ponúknuté a *F(p)* je funkcia údavajúca pravdepodobnosť, že užívateľ príjme odmenu o danej výške.

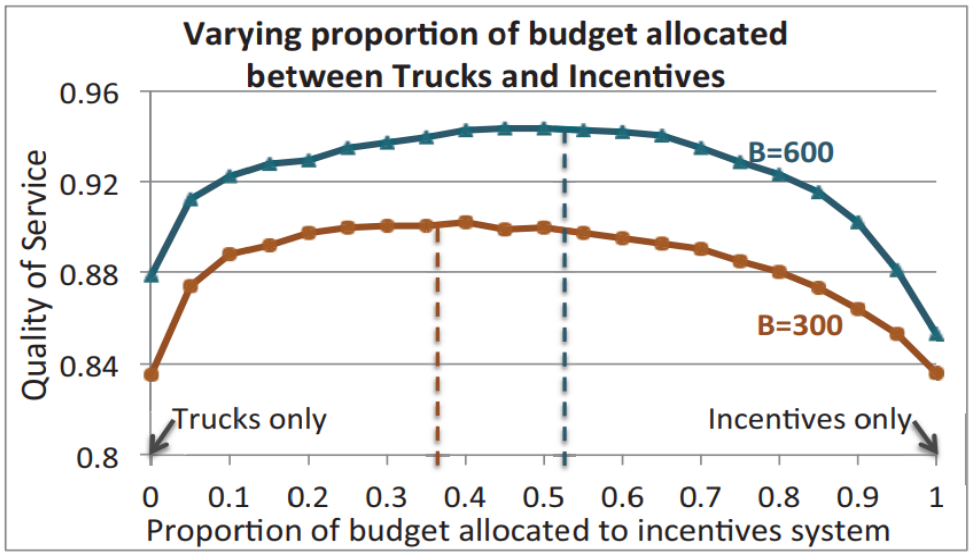
$$p^{OPT} = \arg\max \min_p \left\{ F(p), \frac{B}{N * p} \right\} \; s.t. \; p \in \{p_0, \ldots, p_K\} \tag{1.1}$$

Objektívnu funkciu volili tak, aby výsledný počet udalostí, kedy zákazník nenájde vo svojom okolí voľný bicykel alebo v jeho cieľovej stanici nenájde voľné parkovacie miesto, bol čo najmenší.

Oriešok, s ktorým si museli poradiť je odhadnúť funkciu *F(p)*. Odhad *F(p)* vylepšujú pomocou online učenia za behu aplikácie. FBSS teda môže za rovnakú akciu, rovnakému zákazníkovi ponúknuť rôznu odmenu v inom čase. Popri online učení sa systém musí rozhodnúť, či použije odmenu, o ktorej vie, že ju užívateľ s veľkou pravdepodobnosťou príjme a zároveň nie je priveľká alebo vyskúša novú odmenu a na základe toho si pozmení odhad *F(p)*. V anglickej literatúre sa tento problém označuje ako „explore-exploit trade-off“ [10]. Problém odhadu *F(p)* prirodzene formulujú a rešia ako poupravený Multi-Armed Bandits problém [11], čo je klasický problém spätnoväzobného učenia.

Výsledok tejto práce ukazuje, že stratégia vyvažovania pomocou odmien dokáže prekonať vyvažovanie pomocou flotily vozidiel už pri 20% užívateľskej participácii pri vyvažovaní. Tento prístup vyvažovania je navyše šetrnejší k prírode. Z obrázka B, je vidno, že najviac sa systému darilo, ak rozpočet rovnomerne rozdelili na vyvažovanie pomocou odmien a vyvažovanie pomocou flotily vozidiel.

Podstatný aspekt, ktorý prispel ku výslednej efektivite stratégie je, že užívateľom nikdy neponúkajú odmenu za vypožičanie bicykla zo stanice, ktorá je pri spustení aplikácie ku nim najbližšie a taktiež nikdy za zaparkovanie bi-



Obr. 1.2: Rozpočtový kompromis [2]

cykla do stanice, ktorá je k nim najbližšie v momente, kedy žiadajú o vrátenie bicykla, keďže je veľká pravdepodobnosť, že užívateľ by tam išiel tak či tak, nezávislé na odmene. V texte uvádzajú, že s použitím tohoto pravidla mala simulácia, na ktorej tento systém testovali, o 50 % menšie náklady na vyvažovanie.

Možná nevýhoda tohoto prístupu je vtom, že pri parkovaní bicykla je užívateľovi ponúknutá odmena až na konci jazdy a taktiež to, že systém ponúka iba jednu odmenu. V prípade, že by systém ponúkal odmeny pre všetky zóny už na začiatku, užívateľ BSS by mal možnosť si lepšie naplánovať cestu, čo by vo výsledku mohlo znamenať, že dostane nejakú odmenu a zároveň ušetrí nejaký čas, keďže problémove stanice by mohol mať po ceste. Ak by mu systém ponúkol odmenu až po príchode do svojej cieľovej stanice, musel by sa vraciať späť.

Ďalšia možná nevýhoda je, že odmeňovacia stratégia sa nijak nesnaží vyvažovať systém rovnomerne. Ak by sa nachádzali 2 problémové stanice v skoro rovnakej vzdialenosti od užívateľa, pričom v stanici, ktorá je o trošku bližšie, chýba 5 bicyklov, aby pokryla predpokladaný dopyt, a v druhej chýba 10, pôvodná stratégia by vybrala vždy prvú z nich. Lepší prístup je vybrať druhú stanicu, keďže sa tým zvýši rovnomerne vyváženie naprieč celým mestom.

1.2.2.1 Stručný prehľad ďalších riešení

V práci [4] autori riešia dynamický problém vyvažovania pomocou odmien pre užívateľov. Tieto odmeny ponúkajú na konci užívateľovej jazdy pre blízke

okolité stanice. Pre každý časový úsek v nejakom časovom horizonte, počnúc prítomnosťou, si pre všetky stanice udržiavajú výšky odmien, ktoré budú ponúknuté užívateľom na konci ich jázd. Tieto ceny sa aktualizujú v každom nadchadzajúcom časovom úseku a užívateľ dostáva odmenu len z aktuálneho časového úseku. Výsledný problém formulujú ako problém kvadratického programovania. Ako objektívnu funkciu volili počet udalostí, kedy si užívateľ nemohol vypožičať alebo zaparkovať bicykel, ku počtu všetkých udalostí. Výsledky práce ukazujú, že odmeňovacia stratégia dokaže udržať hodnotu objektívnej funkcie v BSS v meste Londýn nad 87% počas víkendov. Avšak, v rušných hodinách, počas pracovného dňa, samostatná odmeňovacia stratégia nedokázala podstatne navýšiť hodnotu oobjektívnej funkcie.

V práci [12] autori ukázali, že v prípade, že pri parkovaní bicykla užívateľ zakaždým označí 2 stanice, kde by bol ochotný bicykel zaparkovať a následne mu systém z týchto 2 staníc odporučí menej plnú, tak v prípade, že aj malé percento užívateľov nakoniec odnesie bicykel do odporúčanej menej plnej stanice, počet problémových staníc v celom systéme sa dramaticky zníži.

V ďalšej práci [13] autori riešia dynamický problém vyvažovania pomocou odmien pre užívateľov. V prípade, kedy užívateľ nenájde vo svojom blízkom okolí voľný bicykel, systém užívateľovi ponúkne odmenu za extra chôdzu pre vzdialenejší bicykel s cieľom maximalizovať počet uspokojených požiadavok zákazníkov. Tento problém formulujú ako markovský rozhodovací proces [14]. Odmeňovacia stratégia dokázala s nízkym rozpočtom, znížiť počet neuspokojených požiadavok užívateľa o 43 %-63 %.

1.3 Zhodnotenie existujúcich riešení

V predošlej sekcii som predstavil niekoľko možných prístupov k vyvažovaniu. Vyvažovanie pomocou odmeňovacích stratégií ponúka sebestačný, ekologický a efektívny prístup k problému vyváženosti BSS, ktorý má menšie operačné náklady, než prístup pomocou flotily vozidiel. Avšak, ukazuje sa, že často krát bola kombinácia práve oboch prístupov najefektívnejšia.

Kapitola 2

Analýza

V tejto kapitole predstavím môj model FBSS, ktorý budem v rámci tejto práci vyvažovať, formálne definujem problém vyvažovania, detailne rozpíšem odmeňovacie stratégie, ktoré som v rámci tejto práce navrhol a nakoniec popíšem požiadavky kladené na odmeňovacie stratégie a simulačný systém, ktorý použijem na vyhodnotenie týchto odmeňovacích stratégií.

2.1 Prehľad systému

Keďže FBSS žiadne stanice neobsahuje, je potrebné si položiť otázku, čo budem vlastne vyvažovať. K tomuto problému pristupujem následovne. Mesto rozdelím do n rovnako veľkých neprekrývajúcich sa zón, ktoré majú štvorcový tvar. Vo všeobecnosti to môže byť akýkoľovek polygón ale v rámci tejto práce pre jednoduchosť uvažujem, že sa jedná o štvorec. Táto diskretizácia mi dovolí pozerať sa na zóny FBSS ako na stanice SBSS, ktoré následne môžem vyvažovať.

Deň rozdelím do 144 časových úsekov, každý o dĺžke 10 minút. Pomocou tohoto rozdelenia dokážem následne definovať vyvážený systém ako systém, ktorý má pre každý časový úsek dostatok bicyklov vo všetkých zónach aby pokryl dopyt užívateľov. V prípade, že všetky zóny budú v každom časovom úseku vyvážené a veľkosť jednotlivých zón nebude príliš veľká, je rozumné očakávať, že každý užívateľ nájde vo svojej blízkosti voľný bicykel, čo je vlastne cieľom vyvažovania.

Keďže žiaden model, nedokáže dokonale predikovať dopyt po bicykloch, je rozumné aby bol pre všetky časové úseky a v každej zóne aspoň nejaký minimálny počet bicyklov, ktorý určí operátor BSS. To znamená, že ak si v nejakom časovom úseku a v nejakej zóne budú chcieť vypožičiat bicykel 3 užívatelia, tak súčet sum voľných bicyklov, ktoré sa náchadzali na začiatku časového úseku v danej zóne a bicyklov, ktoré užívatelia v tomto časovom úseku do danej zóny zaparkovali musí vo vyváženom BSS spĺňať $sum \geq 3 + minimum$.

Keďže budem vyvažovať zóny v nejakom časovom úseku, môže dôjsť aj pri zdanlivo dokonalom vyvážení k situácií, kedy bude v nejakej zóne chýbať voľný bicykel po skoro celú dĺžku časového úseku. Preto je vhodnejšie voliť kratší časový úsek. Ako príklad uvediem situáciu, v ktorej viem, že v zóne z a na začiatku časového úseku h si bude chcieť 1 človek vypožičať bicykel. Predpokladám, že systému sa podarí nejakého užívateľa namotivovať aby tam počas h nejaký bicykel zaparkoval. Užívateľ ho tam doručí až na konci časového úseku h. Dopyt v z počas h bol 1, počet zaparkovaných bicyklov v z počas h bol 1 ale predsa došlo k situácií, kedy si užívateľ nemohol vypožičať bicykel zo z počas celej dĺžky h.

Pri vyvažovaní FBSS je teda potrebné riešiť, okrem ponúkania vhodne vysokých odmien, ešte 2 podproblémy a to zhlukovanie mesta do zón a ako som už spomenul v úvode, predikovanie stavu zóny pre nejaký časový úsek.

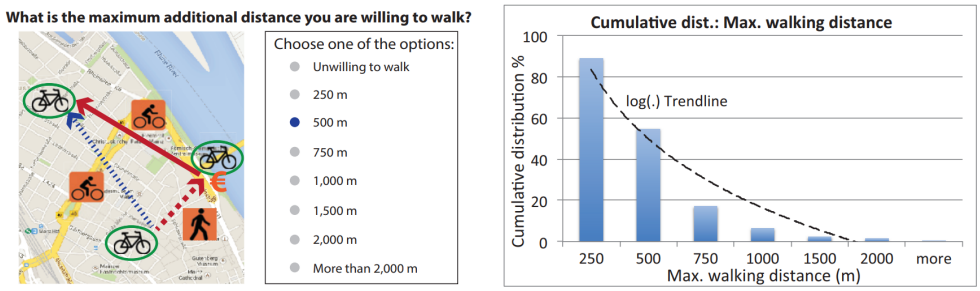
Keďže sa snažím vyvažovať zóny, ich veľkosť by nemala prekročiť maximálnu vzdialenosť, ktorú je väčšina užívateľov ochotná prejsť peši za bicyklom. Na základe ankety 2.1 som zvolil túto vzdialenosť ako 430 m. Najvzdialenejšie body v štvorcovej zóne sú protiľahlé body na diagonálach. Na základe výsledkov ankety a pytagorovej vety som teda zvolil veľkosť strany pre štvorcovú zónu 300 m.

Za povšimnutie stojí, že čím sú zóny väčšie, tým menšie náklady bude mať operátor FBSS na vyvažovanie ale znižuje sa pravdepodobnosť, že užívateľ nájde voľný bicykel v zóne, ktorý nie je od neho príliš ďaleko. Analogicky, čím menšie zóny, tým väčšie náklady na vyvažovanie a väčšia pravdepodobnosť, že užívateľ nájde vo svojej blízkosti voľný bicykel.

Komplikovanejší a zároveň efektivnejší prístup na zhlukovanie mesta do zón je možný pomocou metódy popísanej v práci [15].

Na to aby bolo možné zachytiť celkový dopyt po bicykloch pre nejakú zónu z a časový úsek h, by mal prediktor predikovať počet bicyklov, ktoré prídu do z počas h a taktiež počet užívateľov, ktorí si budú chcieť vypožičať nejaký bicykel zo z počas h. Na základe počiatočného počtu bicyklov v z a týchto hodnôt, je následne možné odhadnúť, koľko bicyklov bude v nejakej zóne chýbať, poprípade o koľko ich tam bude viac a prispôsobiť tomu stratégiu odmeňovania uživateľov s cieľom čo najviac vyvážiť systém.

V prípade, že by nejaký model predikoval iba počet bicyklov v zóne, stratil by informáciu o tom, aký veľký dopyt po bicykloch bude v z počas h. K tomu dôjde v situácií, kedy model predpovedá, že v z počas h bude 0 bicyklov. Z tejto informácie sa nijak nedá zistiť, či je v poriadku, že tam nie je voľný bicykel; napr. v prípade keď si v z počas h nebude chcieť nikto vypožičať bicykel, tak to vporiadku je ale ak tam bude 10 užívateľov, ktorí si budú chcieť vypožičiať bicykel, tak to naopak v poriadku nie je. Z tohoto dôvodu je rozumné predikovať dopyt a počet bicyklov, ktoré užívatelia zaparkujú do z počas h.



Obr. 2.1: Anketa, ktorá bola ponúknutá užívateľom réalneho BSS v meste nachádzajúcom sa v Európe a jej výsledok [2]

2.1.1 Priebeh odmeňovania

V tejto podsekcii popíšem priebeh odmeňovania užívateľa v navrhnutom FBSS.

Uvažujme užívateľa, ktorý príde k nejakému voľnému bicyklu a pokúsi sa ho prenajať. Systém pre každú zónu odhadne čas, kedy by tam približne užívateľ dorazil, za predpokladu, že sa nebude nikde zastavovať a stav zóny pre časový úsek, ktorý príslucha tomuto času. Na základe predpokladaného stavu zóny a dodatočných informácií o prostredí, rôznych trendoch atp. zráta pre všetky zóny odmeny, ktoré môžu byť taktiež nulové a následne ich ponúkne užívateľovi. Ten sa na základe svojej pôvodnej destinácie a výšok odmien pre jednotlivé zóny rozhodne pre miesto, kde zaparkuje bicykel. Každá odmena sa spája s nejakou hornou časovou hranicou. Ak tam užívateľ stihne dôjsť v čase neprekračujúcom túto hranicu, obdrží ponúknutú odmenu. V opačnom prípade sa výška odmeny môže zmeniť.

Výhoda ponúkania odmeny na začiatku jazdy je v tom, že užívateľ si bude môcť lepšie naplánovať cestu a v prípade, že sa rozhodne zaparkovať bicykel do inej než cieľovej destinácie, tak s veľkou pravdepodobnosťou sa taktiež vyhne extra vzdialenosti, ktorú by musel prejsť, ak by mu systém ponúkol odmenu až pri parkovaní bicykla v jeho pôvodnej cieľovej destinácii.

Nevýhoda naopak je, že ak sa užívateľ niekde po ceste zdrží, výška odmeny pre jednotlivé zóny sa môže zmeniť, čo môže mať za následok zvyšenú nespokojnosť užívateľa.

Avšak, podľa [15] väčšina užívateľov využíva BSS na krátke/stredné veľké vzdialenosti, čiže tento štýl odmeňovania by nemal priveľmi trpieť spomenutou nevýhodou.

2.2 Definícia problému

V tejto sekcii formálne definujem problém, ktorý budem v tejto práci riešiť. Pre jednoduchosť, mesto vnímam ako obdĺžnikovu 2D plochu pričom neuvažujem nadmorskú výšku, cesty, budovy, možné vplyvy počasia atp.

BSS. Bod mesta alebo polohu o definujem ako dvojicu $o = (o_{lat}, o_{long}) \in \mathbb{R}^2$ kde o_{lat} resp. o_{long} označuje zemepisnú šírku resp. zemepisnú dĺžku. Množinu zón mesta označím ako $Z = \{z_1, z_2, \ldots, z_n\}$. Zónu prislachujúcu konkrétnej polohe o označím ako $Z(o)$. Deň rozdelím do 144 časových úsekov $h \in H$, kde každý úsek je dlhý 10 minút. Funkciu mapujúcu čas t na časový úsek označím $H(t)$. Množinu bicyklov, s ktorými BSS pracuje označím ako $V = \{v_1, v_2, \ldots, v_m\}$. Každý bicykel má nejakú polohu $L(v)$ a stav $x \in \{free, used\}$, kde stav *free* reprezentuje nevypožičaný bicykel a stav *used* vypožičaný bicykel. Počet voľných bicyklov na konci časového úseku h v zóne z definujem ako $\tau^h(z)$. Aktuálny počet voľných bicyklov v zóne z označím ako $\tau(z)$.

Objektívna funkcia. Kvalitu systému BSS budem hodnotiť podľa počtu situácií, kedy potencionálny užívateľ systému nenájde vo svojej blízkosti voľný bicykel, na základe čoho sa rozhodne nevyužiť služby BSS. Metriku zachytavajúcu tento vzťah prevezmem z [4], ktorej tvar je nasledovný:

$$\text{Service level} = \frac{\text{Potential customers} - \text{No-service events}}{\text{Potential customers}} \tag{2.1}$$

Model užívateľa BSS. Uvažujme užívateľa u s počiatočnou polohou s^u a cieľovou destináciu e^u, ktorý chce využiť služby BSS. Jeho priemernú chodiacu rýchlosť označíme ako α^u_{walk} a priemernú rýchlosť na bicykli ako α^u_{ride}. Maximálnu snahu, ktorú je užívateľ ochotný vynaložiť na cestovanie za voľným bicyklom označím ako $c^u \in \mathbb{R}$. Množinu práve voľných bicyklov označím ako $v' \in V' \subseteq V$. Definujem plán užívateľa ako štvoricu (s^u, v', d^u, e^u) kde v' je bicykel, ktorý si užívateľ plánuje vypožičiať a d^u je miesto, kde užívateľ tento bicykel odloží. Plánu dokážem priradiť jeho cenu pomocou funkcie:

$$\text{price}(plan) = \frac{\|L(v') - s^u\|}{\alpha^u_{walk}} + \frac{\|d^u - L(v')\|}{\alpha^u_{ride}} + \frac{\|e^u - d^u\|}{\alpha^u_{walk}} \tag{2.2}$$

Predpokladám racionalitu užívateľa; to znamená bicykel, ktorý je pre užívateľa najvhodnejší na vypožičanie je rovný $\arg\min_{v' \in V'} \text{price}(plan)$, kde $plan.s^u$ je štartovacia poloha užívateľa, $plan.v'$ je voľný bicykel a miesto odovzdania bicykla $plan.d^u$ je rovné cielovej destinácii $plan.e^u$. V prípade, že V' je neprázdna množina, cena cesty[4] za najvhodnejším bicyklom $\|s^u - L(v')\|/\alpha^u_{walk}$ je menšia než c^u a $\|e^u - s^u\|/\alpha^u_{walk}$, čo reprezentuje cenu cesty ak by sa užívateľ do cieľovej destinácie vydal pešo, tak sa užívateľ rozhodne využiť služby BSS a na svoju jazdu si zarezervuje bicykel v'. V opačnom prípade ich nevyužije a dôjde k poklesu hodnoty objektívnej funkcie.

Odmeňovací model. Označím t ako čas, kedy užívateľ dorazí ku v' a začne svoju jazdu. V tomto momente systém užívateľovi ponúkne n odmien (pre každú zónu jednu) za odvoz bicykla do jednotlivých zón, čím sa snaží

[4]Za povšimnutie stojí, že cena cesty je vlastne rovná času, ktorý užívateľ strávi jej prejdením. To isté platí aj pre cenu plánu.

motivovať užívateľa aby bicykel odniesol do niektorej z vyťažených zón. Predpokladám lineárny vzťah medzi výškou odmeny *bonus* a cenou extra cesty c^*, ktorú je užívateľ ochotný prejsť, čo vyjadrím pomocou rovnice:

$$c^* = \alpha * bonus \tag{2.3}$$

Keďže zóna v réalnom svete má budovy, zablokované ulice atp. a konkrétny užívateľ má rôzne preferencie, miesto v zóne kde užívateľ odnesie bicykel vyjadrím pomocou funkcie:

$$\text{rand(z, e}^{\text{u}}) = \begin{cases} e^u, & \text{ak zvolená zóna } z' = Z(e^u) \\ \text{náhodný bod zo z}, & \text{inak.} \end{cases} \tag{2.4}$$

V prípade, kedy užívateľ vracia bicykel do inej než cieľovej zóny, funkcia generuje náhodné body tak aby vo výsledku extra vzdialenosť, ktorú musí prejsť pešo do e^u nebola príliš veľka. V ideálnom prípade by užívateľ bicykel vrátil priamo na kraj zóny, čo častokrát nie je možné/žiadúce.

Pre každú zónu z označím t' ako čas, kedy užívateľ dorazí na miesto $d^u = \text{rand}(z, e^u)$. Predpokladám, že užívateľ sa po ceste nikde nazastavuje, čiže $t' = t + \|d^u - L(v')\|/\alpha^u_{ride}$. Výška bonusu pre konkrétnu zónu, užívateľa a bicykel je teda daná funkciou:

$$\beta(u, v', z) = \begin{cases} \text{bonus}(), & \text{ak } \tau^h(z) \leq minimum; h = H(t') \\ 0, & \text{inak.} \end{cases} \tag{2.5}$$

Premenná $minimum \in \mathbb{N}$ označuje hornú hranicu nedostatočného počtu bicyklov v zóne. Rôzne stratégie výpočtu funkcie bonus() sú detajlnejšie popísané v sekcii 2.3.

Definujem množinu možných miest na vrátenie bicykla ako $D^u = \{d_1, d_2, \ldots, d_n\}$ kde d_i sú výsledky funkcie $\text{rand}(z, e^u)$ pre užívateľa u a jednotlivé zóny. Na základe vzdialenosti týchto bodov od e^u a jednotlivých odmien pre zóny, si uživateľ vyberie jedno z týchto miest, ktoré minimalizuje celkovú cenu cesty, čo vyjadrím vzťahom:

$$\begin{aligned} \psi(u) = plan = \\ (s^u, v', \arg\min_{d \in D^u}\{\text{price}(s^u, v', d, e^u) - \beta(u, v', Z(d)) * b^u\}, e^u) \end{aligned} \tag{2.6}$$

Užívateľ teda začína interakciu so systémom v čase t na mieste s^u, kde buď nájde alebo nenájde vhodný bicykel. V prípade úspechu, príde peši k bicyklu v', kde mu systém ponúkne odmeny za odnos do jednotlivých zón, ktoré môžu byť taktiež nulové. Užívateľ si jednu z nich vyberie, následne do danej zóny odnesie bicykel, obdrží odmenu a ako posledný krok, dokončí svoju cestu peši z miesta kde vrátil bicykel do jeho finálnej destinácie e^u.

2.3 Stratégie výpočtu odmeny pre užívateľa

V následujúcej časti textu podrobnejšie popíšem rôzne stratégie na výpočet odmeny pre užívateľa za odnos bicykla do vyťažených zón, ktoré je možné dosadiť do rovnice 2.5 namiesto funkcie bonus(). Nenulové odmeny sú teda užívateľom ponúkané len v prípade, kedy je splnená vyvažovacia podmienka daná rovnicou 2.5.

Cieľom týchto stratégií je maximalizovať predpokladané navýšenie objektívnej funkcie s čo najmenšími nákladmi. Stratégie je možné konfigurovať pomocou rôznych parametrov. Efekt, ktoré jednotlivé stratégie majú na vyváženie systému a rôzne štatistiky sú popísané v kapitole 5.

2.3.1 Fixná odmena

Jedná sa o jednoduchú stratégiu, ktorá užívateľovi zakaždým ponúkne jednu vopred stanovenú odmenu.

Aby ponúknutá odmena mala nejaký zmysel, musí byť za ňu užívateľ ochotný odniesť bicykel aspoň do zón, ktoré sú susedné k jeho cieľovej zóne. Predpokladáme, že v priemere je extra vzdialenosť, ktorú by takto musel prejsť rovná priemernej vzdialenosti stredov susedných zón, čo označím d^*. Priermernú rýchlosť užívateľa označím ako s^*. Cena tejto extra cesty, je teda rovná $c^* = d^*/s^*$. To znamená, že výsledná výška odmeny by mala byť podľa 2.3 rovná $bonus = c^*/\alpha$.

Výhody tejto stratégie sú ľahká implementácia, možné okamžité použitie bez nutnosti trénovať nejaké modely atp. a možný lepší dopad na user experience a týmpádom aj zväčšenú pravdepodobnosť prijatia odmeny za odvoz bicykla do vyťaženej zóny, keďže vďaka buď fixnej alebo nulovej odmene je ľahšie si vybrať ideálnu zónu, než v prípade väčšieho počtu rôznych odmien pre rôzne zóny [16].

Medzi hlavné nevýhody patrí fixné oceňovanie zón. Napr. na to aby užívateľ odniesol bicykel do zóny, ktorá je ďaleko od centra alebo nejakým spôsobom neatraktívna, potrebuje väčšiu motiváciu. Naopak v prípade atraktívnych/často navštevovaných zón nie je až tak potrebné užívateľa motivovať vysokou odmenou. Taktiež, niekedy je pre systém výhodnejšie aby užívateľ odniesol bicykel do vyťaženejšej zo zón, čo môžme dosiahnúť navýšením odmeny pre viac vyťaženú zónu oproti ostatným menej vyťaženým zónam.

2.3.2 Dynamické určovanie výšky odmeny

Stratégia využíva na určenie výšky odmeny informácie o pravdepodobnosti zaparkovania bicykla v danej zóne a taktiež bere v úvahu nedokonalosť prediktívneho modelu.

Stratégia sa dá konfigurovať pomocou 3 parametrov: výška minimálnej odmeny *min incentive*, výška štandardnej odmeny *standard incentive* a hranica nedokonalosti prediktoru *prediction error limit*.

Nech z reprezentuje počiatočnú zónu užívateľa. Bonus pre každú ďalšiu zónu $z_i \in Z$ strategia zráta následovne.

V prípade, kedy predikcia stavu zóny $\tau^h(z_i)$ z rovnice 2.5 je v intervale

$$minimum - prediction\ error\ limit \leq \tau^h(z) \leq minimum \tag{2.7}$$

stratégia ponúkne užívateľovi bonus pre zónu z_i o výške *min incentive*. Týmto sa stratégia snaží zbytočne nenavyšovať odmenu pre zóny, u ktorých nie je úplne isté, že ich stav v skutočnosti prekročí hranicu *minimum* kvôli nedokonalosti prediktora.

Pre zónu z označím $P_z = \{p_{z_1}, p_{z_2}, \ldots, p_{z_n}\}$ ako množinu pravdepodobností, kde p_{z_i} označuje pravdepodobnosť, že cieľová zóna jázd začínajúcich v z, bude z_i. V prípade, kedy platí nerovnosť $\tau^h(z_i) < minimum - prediction\ error\ limit$ sa výška bonusu pre cieľovú zónu z_i zráta pomocou následujúcej rovnice:

$$bonus = \max\{(1 - p_{z_i}) * standard\ incentive, min\ incentive\} \tag{2.8}$$

To znamená, že ak je priveľka pravdepodobnosť, že užívateľ štartujúci jazdu v zóne z odnesie bicykel do zóny z_i, nezávislé na odmene, stratégia užívateľovi ponúkne nižšiu odmenu. Týmto sa snažím predísť zbytočnému odmeňovaniu užívateľov, ktorých pôvodná cieľová zóna je s veľkou pravdepodobnosťou z_i.

Medzi možné nevýhody tejto stratégie patrí negatívny dopad na užívateľovo rozhodovanie, kvôli množstve rôznych odmien [16].

2.4 Požiadavky

Na začiatku tvorby akéhokoľvek informačného systému je nutné špecifikovať požiadavky, ktoré má systém spĺňať.

Aby požiadavky dávali väčší zmysel, popíšem najprv základný scénar používania simulačného systému.

Implementujem odmeňovaciu stratégiu, ktorá sa dá konfigurovať pomocou rôznych parametrov. Túto stratégiu otestujem s rôznymi parametrami na rôznych simuláciach BSS, ktoré sú taktiež konfigurovateľné. Na základe výsledkov simulácií zhodnotím efektivitu implementovanej odmeňovacej stratégie a efekt zvolených parametrov. Beh simulácie si pre lepšie pochopenie zvizualizujem pomocou grafickej aplikácie.

Medzi funkčné požiadavky teda patrí:

1. Systém umožňuje simulovať chod FBSS s odmeňovacími stratégiami na základe konfigurácie.

2. Systém dokáže poskytovať štatistiky o behu simulácií, na základe ktorých je možné vyhodnotiť efektivitu týchto stratégií.

3. Beh simulácie je možné graficky zvizualizovať.

4. Simuláciu je čiastočne možné nakonfigurovať pomocou historických dát reálneho BSS.

Pri nefunkčných požiadavkách som sa zameral najmä na rýchlosť a modulárnosť systému:

1. Odmeňovacie stratégie sú implementované ako samostatný modul.

2. Beh simulácie, ktorá sa počtom udalostí a veľkosti mesta približuje reálnemu BSS, zbehne v testovacom prostredí do cca 20 minút.

3. Systém je ľahko rozšíriteľný; po pridaní novej odmeňovacej stratégie, nového prediktivného alebo zhlukovacieho modelu by zmeny v kóde systému mali byť minimálne.

4. Zdrojový kód je rozdelený do znova použiteľných modulov podľa logických celkov.

Kapitola 3

Návrh

V tejto kapitole popíšem môj prístup k realizácii simulačného systému a odmeňovacích stratégií, popísaných v kapitole 2. Základná štruktúra systému odpovedá problémom riešeným v tejto kapitole.

Kapitola má nasledujúcu štruktúru. V prvej sekcii sa venujem návrhu simulačného systému, následne popisujem jeho architektúru a jednotlivé moduly. Kapitolu zavŕšujem popisom tried a ich zodpovedností.

3.1 Simulačný systém

Aby som mohol vyhodnotiť implementované stratégie odmeňovania je potrebné vytvoriť systém, ktorý bude simulovať chod BSS. Takáto simulácia by sa mala čo najviac približovať reálnemu BSS aby výsledky z nej plynúce boli relevantné. V nasledujúcich sekciách priblížim môj prístup k návrhu tohoto systému.

3.1.1 Model simulácie

Simulácia bude simulovať chod BSS počas jedného dňa, ktorý som popísal v sekciách 2.1 a 2.2.

Keďže v rámci práce sa nevenujem problému predikovania stavu zón, musím vytvoriť náhradu za prediktor, ktorá bude pri simulácií poskytovať informácie o budúcom stave zón. Tu priblížim v sekcii 3.1.5.

Na samotné simulovanie je potrebné taktiež vygenerovať scénar, ktorý systém odsimuluje. Scénar tvoria rôzne udalosti, ktoré môžu predstavovať vypožičanie bicykla užívateľom, vrátenie bicykla užívateľom alebo pridanie nového bicykla na nejaké miesto. Každá udalosť nastáva v nejakom čase. Pri vypožičiavaní bicykla sa užívateľ riadi modelom popísaným v sekcii 2.2. Každý užívateľ má teda nejaké štartovacie miesto, finálnu destináciu a vlastnosti ako priemernú rýchlosť na bicykli, rýchlosť chôdze a číslo udavajúce vzťah medzi bonusom a extra vzdialenosťou, ktorú je ochotný za danú odmenu

prejsť. Miesto, kde zaparkuje bicykel zvolí na základe ponuknutých odmien na začiatku udalosti. Užívateľ sa taktiež môže rozhodnúť nevyužiť služby BSS, ak vo svojom okolí nenájde voľný bicykel. Jednotlivé udalosti môžu za behu simulácie rozširovať scénar ďalšími udalosťami. Napríklad úspešná udalosť vypožičanie bicykla vygeneruje udalosť vrátenie bicykla.

Simulačný systém tvoria primárne nasledujúce komponenty:

- zhlukovacia komponenta, ktorá rozdelí mesto do zón,
- predikovacia komponenta, ktorá predikuje budúce stavy zón,
- odmeňovacia komponenta, ktorá počíta odmeny pre zóny na základe zvolenej stratégie,
- komponenta generejúca scénar,
- rozhodovacia komponenta, ktora reprezentuje užívateľovo chovanie,
- komponenta, ktorá si udržuje aktuálny stav simulovaného sveta,
- komponenta, ktorá postupne simuluje vygenerované udalosti a na základe ich výsledkov zbiera štatistiky na finálne vyhodnotenie behu simulácie.

Jednotlivé komponenty ako odmeňovacia komponenta, generátor scénara atp. sú konfigurovateľné pomocou parametrov. Simuláciu teda vytváram na základe konfigurácie, ktorú bližšie popíšem v nasledujúcej sekcii.

3.1.2 Konfigurácia simulácie

Medzi najdôležitejšie parametre patria koordináty mesta. Keďže mesto berem ako obdĺžnikovu 2D plochu, reprezentujem ho pomocou pravého a ľavého horného rohu. Svet rozdeľujem do zón, ktorých šírka a výška je taktiež daná parametrom. Parameter štartovací čas označuje počiatočný čas simulácie. Parameter vyvažovacia hranica určuje maximálny počet bicyklov v zóne, za ktorý sa bude ponúkať odmena. Inými slovami, ak je v zóne väčší predikovaný počet bicyklov než je vyvažovacia hranica, tak odmeňovacia komponenta pre zónu neponúkne odmenu. Jednotlivé odmeňovacie stratégie sú taktiež konfigurovateľné pomocou parametrov, ktoré som opísal v sekcii 2.3.

Medzi parametre generátoru scénara patrí počet udalostí vypožičania bicykla, čas od ktorého má tieto udalosti generovať, počet bicyklov BSS, minimálny počet bicyklov, ktorý má byť v každej zóne na začiatku simulácie a rôzne náhodné veličiny, ktoré sú jednoznačne určené na základe predaných parametrov. Medzi ne patrí náhodná veličina počtu sekúnd, ktorá sa pripočíta ku počiatočnému času simulácie, na základe čoho určím čas, kedy udalosť nastane. Ďalej používa náhodné veličiny na generovanie miest udalostí vypožičania bicykla a vlastností užívateľa ako priemerná rýchlosť chôdze a jazdy na bicykli, maximálna cena cesty chôdzou za voľným bicyklom a koeficient udávajúci

lineárny vzťah medzi extra vzdialenosťou, ktorú je užívateľ ochotný prejsť a výškou odmeny. Náhodné veličiny môžu byť z rovnomerného, normálneho, exponencionálneho alebo z vlastného rozdelenia, ktoré je určené poľom hodnôt, na základe ktorého pre každú unikátnu hodnôtu zrátam pravdepodobnosť výskytu pomocou vzťahu $p = \frac{\text{Počet výskytov v poli}}{\text{Veľkosť poľa}}$.

Aby bolo možné spustiť rovnakú simuláciu dvakrát po sebe, ďalším parametrom je číslo, ktoré nainicializuje generátor pseudonáhodných čísel, ktorý používam v kóde na vytváranie náhodných veličín.

Konfigurácia taktiež obsahuje príznak, ktorý označuje či sa má použiť nejaká odmeňovacia stratégia a ak áno tak ktorá.

Keďže jednotlivé parametre spolu súvisia, napríklad veľkosť sveta, veľkosť zóny, celkový počet bicyklov a minimálny počet bicyklov v zóne, môže sa stať, že výsledná konfigurácia je nevalidná – napríklad v prípade, kedy BSS nemá dostatočný počet bicyklov, aby pokryl každú zónu s minimálnym počtom bicyklov. Simuláciu je možné spustiť iba s validnou konfiguráciou.

3.1.3 Zhlukovanie sveta do zón

Ako som už spomenul, mesto reprezentujem pomocou obdĺžnikovej plochy, taktiež ho rozdeľujem do neprekrývajúcich sa zón o fixnej veľkosti, pričom postupujem od ľavého dolného rohu smerom doprava a hore. Následne potom, čo pokryjem celý svet, rozdelím zóny medzi aktívne a neaktívne. Neaktívne zóny sú tie, pre ktoré neexistuje udalosť vo vygenerovanom scénari, v ktorej sa užívateľova štartovacia poloha alebo finálna destinácia nachádza v danej zóne. Simulovaný BSS ponúka odmeny iba pre aktívne zóny a k neaktivným sa stavia tak ako keby neexistovali. V reálnom BSS neaktívne zóny mesta reprezentujú napríklad vodné plochy.

3.1.4 Generátor scénara

Generátor scénara dokáže generovať scénar na základe konfigurácie, kde sú dané všetky hodnôty alebo si časť konfigurácie – náhodné veličiny týkajúce sa vypožičania bicykla – dokáže doplniť na základe historických jázd reálneho stanicového BSS.

Prvá udalosť scénara je pridanie všetkých bicyklov do mesta. Tie sú rozmiestnené následovne. Do každej zóny sa vygeneruje na náhodne miesto minimálny počiatočný počet bicyklov pre zónu. Náhodné miesta zóny sú z rovnomerného rozdelenia. Následne, zvyšné bicykle sa rozmiestnia náhodne po meste, pričom miesto, kde budú umiestnené je dané náhodnou veličinou, ktorá udáva štartovacie miesta užívateľov. Týmto docielim, že najvýťaženejšie zóny budú mať na začiatku simulácie najviac bicyklov, čo je tradičný postup operátorov BSS – v noci, kedy je najmenší dopyt po bicykloch, operátori riešia statický problém vyvažovania a pripravujú BSS na ďalší deň [17].

Následne generátor generuje udalosti vypožičania bicykla. V prípade, že konfigurácia vznikla na základe nejakého historických jázd stanicového BSS, generátor vytvorí nahodnú veličinu, ktorá reprezentuje počet sekúnd, ktorý sa má pričítať ku počiatočnému času simulácie, na základe čoho určím, kedy má udalosť nastať. Ďalej, pre každý časový úsek vytvorí náhodnú veličinu, ktorá udáva id štartovacej stanice a taktiež, pre každý časový úsek a štartovaciu stanicu, vytvorí náhodnú veličinu udavajúcu id cieľovej stanice. Náhodné rozdelenia týchto veličín určí práve na základe historických jázd. Tieto veličiny následne použije na vygenerovanie udalostí vypožičania bicykla, kde postupne vygeneruje: čas udalosti, štartovaciu stanicu a cieľovú stanicu.

Keďže simulujem FBSS a ten stanice nemá, tak takýto scénar bude musieť generátor ešte trochu poupraviť. Každú stanicu – buď počiatočnú alebo cieľovú – v scénari nahradí náhodnym bodom z nejakého blízkeho okruhu tejto stanice. Týmto sa zachová to, že užívateľova finálna destinácia je blízko jeho pôvodnej cieľovej stanici a zároveň sa priblížim réalnemu FBSS, kde užívatelia parkujú bicykle naprieč celým mestom.

V prípade, že generátor používa klasickú kompletnú konfiguráciu, na vygenerovanie štartovacieho miesta a finálnej destinácie použije rovnakú náhodnú veličinu danú konfiguráciou.

Všetky miesta udalostí sú normalizované tak aby sa vo výsledku nachádzali vo vnútri plochy mesta. V prípade, že náhodná veličina vygeneruje bod mimo mesta, tento bod sa pousunie k najbližšiemu miestu na hranici.

Podobne to je aj s časom udalosti. Keďže simulujem jeden deň, čas kedy nejaká udalosť scénara nastane, nemôže nastať skôr, než počiatočný čas simulácie alebo neskôr než 24 h od počiatočného času simulácie.

Aj keď budem generovať scénar na základe datasetu, je dôležité si uvedomiť, že nemusí úplne korešpondovať s realitou, keďže v datasete nie sú informácie o užívateľoch, ktorí nenašli vo svojom blízkom okolí voľný bicykel, na základe čoho nevyužili služby SBSS. V tejto práci avšak predpokladám, že SBSS, ktorého dataset používam, dosahoval 100% kvalitu systému danú rovnicou 2.2.

3.1.5 Náhrada za prediktívny model

Na to, aby som pre nejakú zónu mohol spočítať primeranú výšku odmeny, je potrebné poznať jej „budúci“ stav. Je dôležité si uvedomiť, že predikovaný stav zóny má citeľný dopad na objektívnu funkciu systému, keďže stratégie odmeňovania sú silne závislé na tejto predikcii. V prípade, že systém používa zlý prediktor, nie je rozumné očakávať, že stratégie odmeňovania budú fungovať kvalitne. Ak by som v rámci simulácie vytvoril dokonalý prediktor, bolo by možné spoznať hornú hranicu efektivity stratégií odmeňovania, čo je vlastne cieľom simulácie.

Keďže simulácie pozná celý scénar a taktiež rozhodovací model užívateľa, mohlo by sa zdať, že pre simuláciu je možné vytvoriť dokonalý prediktor.

Avšak, je potrebné si uvedomiť, že výsledná predikcia stavu zóny ovplyvňuje to, či systém pre nejakú zónu ponúkne užívateľovi odmenu alebo nie. Na základe ne/ponúknutej odmeny sa užívateľ rozhodne, či do danej zóny bicykel ne/zaparkuje. Tým pádom zmení jej stav. V skratke to znamená, že výsledná predikcia stavu zóny môže ovplyvniť jej budúci stav kvôli čomu nie je možné vytvoriť dokonalý prediktor, dokonca ani v simulácií.

Simulácia sa kvôli potrebám prediktora, ktorý v nej používam, skladá z dvoch behov rovnakej simulácie, ktoré sa líšia len v tom, že prvý z nich nepoužíva odmeňovacie stratégie. Odmeňovacie stratégie sú samozrejme vyhodnocované len na druhom behu. Štatistiky z prvého behu použijem na inicializáciu prediktívneho modelu druhého behu. Konkrétne používam štatistiky o dopyte po bicykloch a počet vrátených/pridaných bicyklov v jednotlivých zónach pre každý časový úsek. Keďže obe simulácie spúšťam s rovnakým scénarom, môžem tieto informácie použiť v druhom behu. Avšak, druhý beh používa odmeňovacie stratégie, ktoré môžu zmeniť užívateľovo správanie, čo trošku komplikuje prediktívny model. V prípade, že v druhom behu dôjde k nejakej zmene v užívateľovej udalosti, napr. zmení parkovacie miesto alebo si vypožičia iný bicykel, než v prvom behu, udalosť informuje prediktor o týchto zmenách, ten si následne poupraví model tak, aby tieto zmeny reflektoval. Nevýhoda je, že prediktor je možné o týchto zmenách informovať až v momente, kedy sa udalosť vykonáva, keďže až do tohoto momentu nie je jasné, či vlastne užívateľ bude mať poblízku nejaký voľný bicykel, či zmení miesto parkovania atp., keďže aj tesne predchádzajúca udalosť môže ovplyvniť následujúcu udalosť.

3.1.6 Štatistiky simulácie

Pre každý beh simulácie zbieram data o výsledkoch všetkých udalostí scénara, na základe ktorých vyhodnocujem efektivitu odmeňovacích stratégií, získavam rôzne štatistiky o FBSS alebo pomocou ktorých dokážem graficky znázorniť celkový priebeh simulácie.

Odmeňovacie stratégie primárne porovnávam na základe počtu udalostí, kedy sa užívateľ rozhodol nevypožičať bicykel, ktoré sa snažím minimalizovať viď 2.2 a súčtu rozdaných odmien, ktorý sa taktiež snažím minimalizovať.

Medzi ďalšie štatistiky taktiež patria informácie z prediktivného modelu, na základe ktorých je možné vyhodnotiť jeho presnosť, štatistiky ako priemerná prejdená užívateľova vzdialenosť peši a na bicykli, počet udalostí kedy užívateľ za odmenu zaparkoval bicykel do inej než cieľovej zóny, priemerný počet bicyklov v zóne a počet aktívnych zón, do ktorého bolo mesto rozdelené.

3.2 Architektúra systému

Simulačný systém som rozdelil do 2 spustiteľných aplikácií, ktoré zdieľajú spoločný modul `simulation`, ktorý predstavuje jadro celej simulácie.

Jedná sa o `simulation-controller`, ktorá má na starosti spúšťanie simulácie na základe konfigurácie a ukladanie výsledkov, a `simulation-drawing`, ktorá dokáže graficky znázorniť priebeh simulácie.

Jadro celého systému `simulation` ponúka všetky potrebné funkcionality na vytvorenie a spustenie simulácie.

3.2.1 Simulácia

Medzi základné funkcionality modulu `simulation` patrí vytvorenie simulácie, popísanej v sekcii 3.1 na základe dodanej konfigurácie, vytvorenie scénara pre simulácií a to buď z historických dát alebo pomocou vopred určenej konfigurácie a zbieranie štatistík o samotnom behu simulácie.

3.2.2 Spúšťanie simulácie

Aplikácia `simulation-controller` má na starosti vytvorenie simuláčnej konfigurácie z extérneho súboru. Túto konfiguráciu môže doplniť o data z historických jázd reálneho BSS. Konkrétne v práci používam SBSS citibike NYC[5]. Výslednú konfiguráciu následne predá modulu `simulation`. Ten na základe toho vytvorí a spustí simuláciu a na konci vráti výsledok. Modul `simulation-controller` z výsledku simulácie vytvorí rôzne štatistiky, ktoré následne uloží do súboru na posúdenie operátorovi.

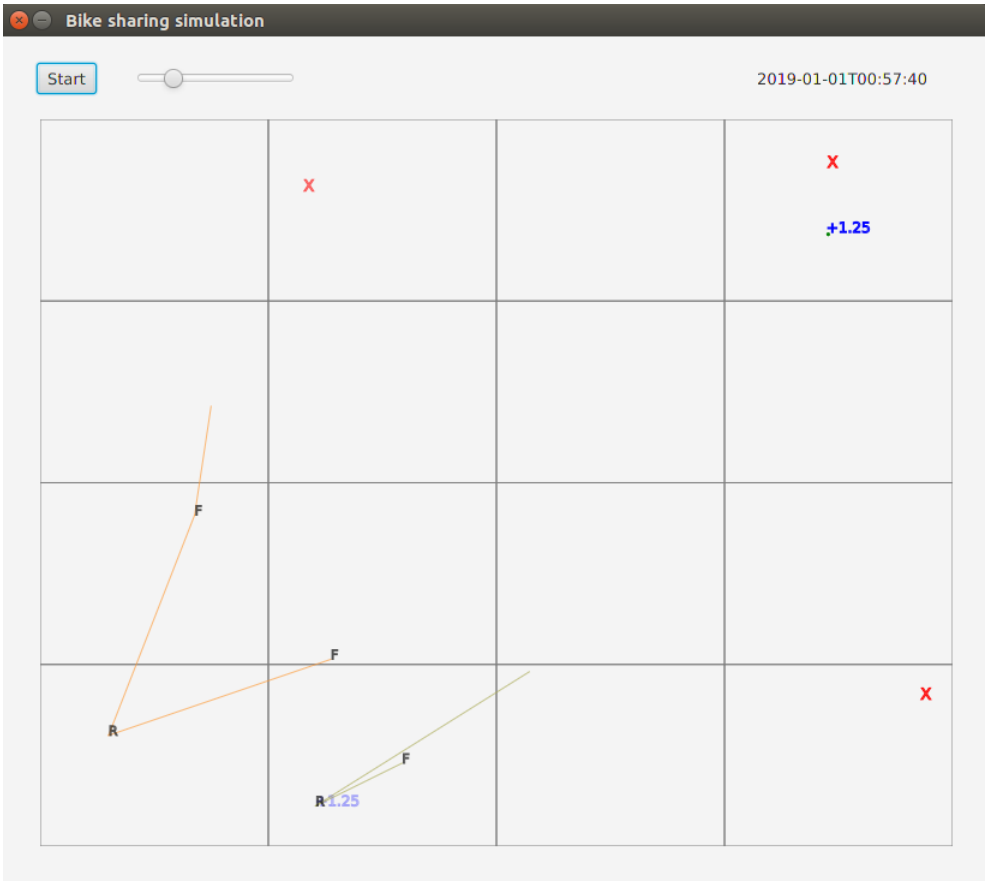
3.2.3 Vykresľovanie simulácie

Aplikácia `simulation-drawing` spúšťa simuláciu na základe konfigurácie z externého súboru, ktorej výsledky ukladá a následne graficky vykresľuje pomocou desktopovej aplikácie.

Na obrázku 3.1 je vidno základné grafické rozhranie. Pomocou tlačitka `Start` je možné spustiť a pozastaviť vykresľovanie simulácie. Rýchlosť vykresľovania udalostí simulácie sa dá zmeniť pomocou šúpatka. Aktuálne vykresľovaný čas simulácie je možné vidieť v pravom hornom rohu.

Najväčšiu časť grafickej plochy zaberá plátno, na ktoré sa vykresľuje dianie simulovaného FBSS. Obdĺžnikove plochy, ohraničené čiernou čiarou, reprezentujú zóny mesta. Zelené bodky predstavujú voľné bicykle. Červené X označuje užívateľa, ktorý vo svojom okolí nenašiel voľný bicykel, na základe čoho sa rozhodol nevyužiť služby FBSS. Zelená zlomená čiara označuje klasickú cestu užívateľa, ktorý nijak nezmenil svôju pôvodne plánovanú jazdu. Úsek cesty od písmena F po R predstavuje trasu, ktorú užívateľ prešiel peši po bicykel.

[5] https://www.citibikenyc.com/system-data



Obr. 3.1: Grafická vizualizácia simulácie

Úsek cesty od R po koniec čiary predstavuje úsek, ktorý užívateľ prešiel na bicykli. Na konci čiary, bez písmena, bicykel zaparkoval. Oranžová zlomená čiara označuje cestu užívateľa, ktorý prijal odmenu do inej než pôvodne cieľovej zóny. Úsek od písmena F po R označuje trasu, ktorú prešiel peši po bicykel. Úsek cesty od R po písmeno F, ktoré sa nachádza na ďalšom zlome čiary, predstavuje úsek, ktorý prešiel na bicykli. V tomto zlome bicykel zaparkoval a úsek cesty od F po koniec čiary, bez písmena, označuje trasu, ktorú prešiel peši do svojej cieľovej destinácie. Teda koniec čiary bez písmena označuje cieľovú destináciu. Celá cesta sa zjaví v momente, kedy sa užívateľ rozhodol zarezervovať bicykel a postupne sa stráca do neznáma. Modré číslo označuje výšku odmeny, ktorú užívateľ na tomto mieste a čase obdržal. Zjavujúce sa zelené bodky predstavujú bicykle, ktoré boli práve zaparkované.

3.2.4 Štruktúra systému

Systém som kvôli priehľadnosti a znovupoužiteľnosti rozdelil do viacerých modulov; každý rieši nejaký problém spomenutý v predošlých sekciách:

- `clustering` rozdeľuje mesto do zón.
- `environment` poskytuje funkcionalitu na udržovanie stavu sveta simulácie.
- `decide-best-option` predstavuje rozhodovací model zákazníka popísany v 2.2.
- `rides-prediction`, je v ňom implementovaná náhrada za prediktívny model 3.1.5, ktorá sa používa v rámci simulácie.
- `user-rebalancing` obsahuje implementované stratégie odmeňovania 2.3.
- `shared` obsahuje základné spoločné objekty, ktoré sa používajú vo všetkých ostatných moduloch.
- `simulation` poskytuje funkcionalitu na vytvorenie a spustenie simulácie.
- `simulation-controller` vytvára konfiguráciu zo súboru, spúšťa simuláciu a výsledok ukladá taktiež do súboru.
- `simulation-drawing` vykresľuje beh simulácie.

Vzájomné závislostí týchto modulov je možné vidieť na obrázku 3.2.

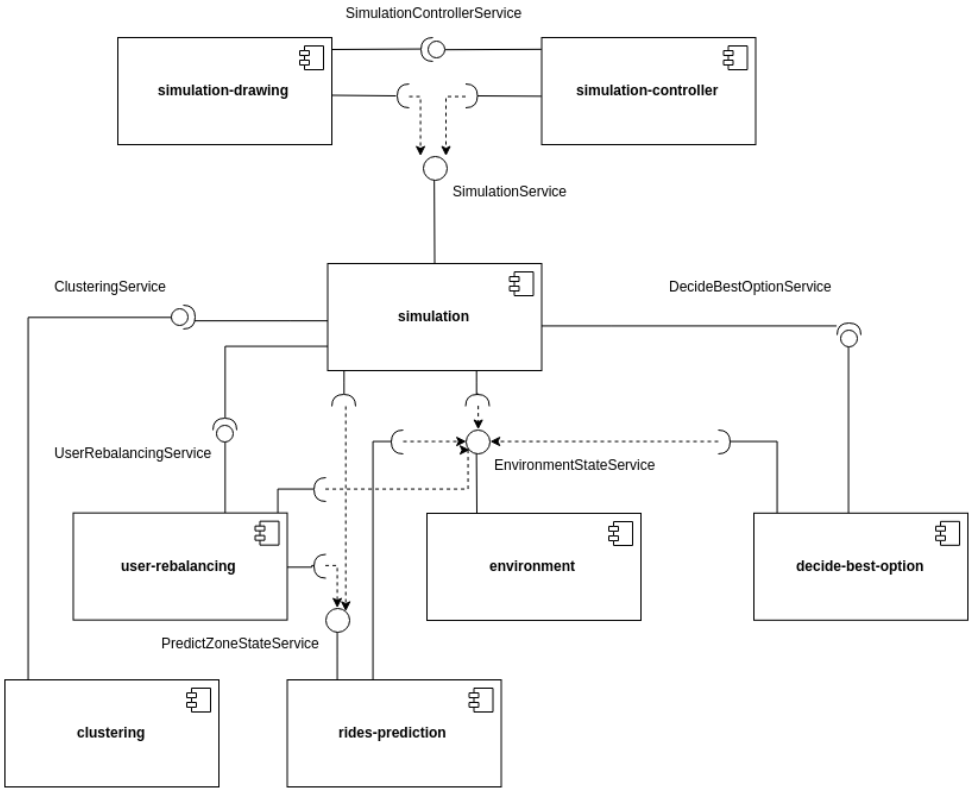
3.3 Návrh tried

V tejto sekcii priblížim zodpovednosť jednotlivých tried, ktoré systém obsahuje. Na diagrame tried, ktorý je rozdelený do dvoch obrázkov, 3.3 a 3.4, sú znázornené najdôležitejšie triedy systému a väzby medzi nimi. Kvôli čitateľnosti nie sú vymenované metody a atribúty týchto tried.

Jednotlivé moduly (až na modul `simulation`, ktorý to dáva všetko dokopy) sú závislé len na servisných rozhraniach iných modulov. Rozpad do logických celkov, ktoré sú spojené pomocou malých rozhraní, zvyšuje prehliadnosť kódu, znižuje jeho komplexitu, umožňuje jednoduchú rozšíriteľnosť a taktiež náhradu kódu.

Bližší pohľad na rozhrania a niektoré ich metody, je možné vidieť na obrázku 3.5. Zodpovednosť týchto rozhraní a ich prislúchajúcich modulov podrobnejšie popíšem v nasledujúcich sekciách.

Pre úplnosť a lepšie chápanie súvislosti medzi triedami, vskratke popíšem „kolečko“ základného modulu `simulation`, ktoré je taktiež znázornené na sekvenčnom diagrame 3.6. Jediné servisné rozhranie, ktoré poskytuje je



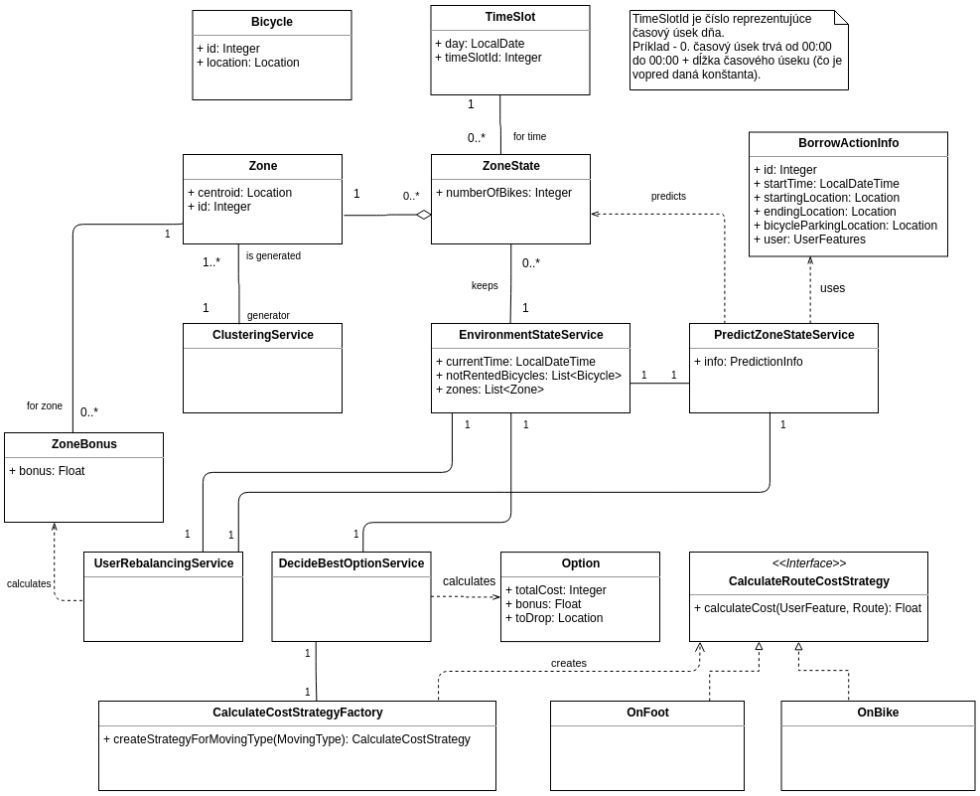
Obr. 3.2: Diagram komponent simulačného systému

`SimulationService` a to má iba jednu metodu `startSimulation`. Tá v parametri prijíma objekt reprezentujúci konfiguráciu, na základe ktorej sa simulácia vytvorí. Počas vytvárania simulácie sa generuje scénar a inicializujú jednotlivé servisy ostatných modulov, na ktorých je simulácia závislá. Každá udalosť v scénari obsahuje čas a akciu, ktorá sa má vykonať. Trieda `Simulation` postupne odoberá všetky udalosti scénara, ktoré následne vykonáva, pričom si ukladá ich výsledný efekt na svet. Po vykonaní všetkých udalostí, pozbiera štatistiky o efektoch udalostí a prediktívnom modeli, do výslednej štatistickej správy, ktorú vráti ako výsledok volania metody.

3.3.1 Zhlukovanie

Modul má na starosti rozdeľovanie plochy mesta do zón. Na zákalde konfigurácie zisti dĺžku a šírku zóny, pomocou ktorých rozdelí mesto do pravidelných zón. V prípade, že šírka/dĺžka mesta nie je deliteľná šírkou/dĺžkou zóny v konfigurácií, posledný stĺpec zón na pravej strane alebo najvyšší riadok, su oseknuté tak aby spadali do plochy mesta.

Základné rozhranie, ktoré tento modul poskytuje je `ClusteringService`.

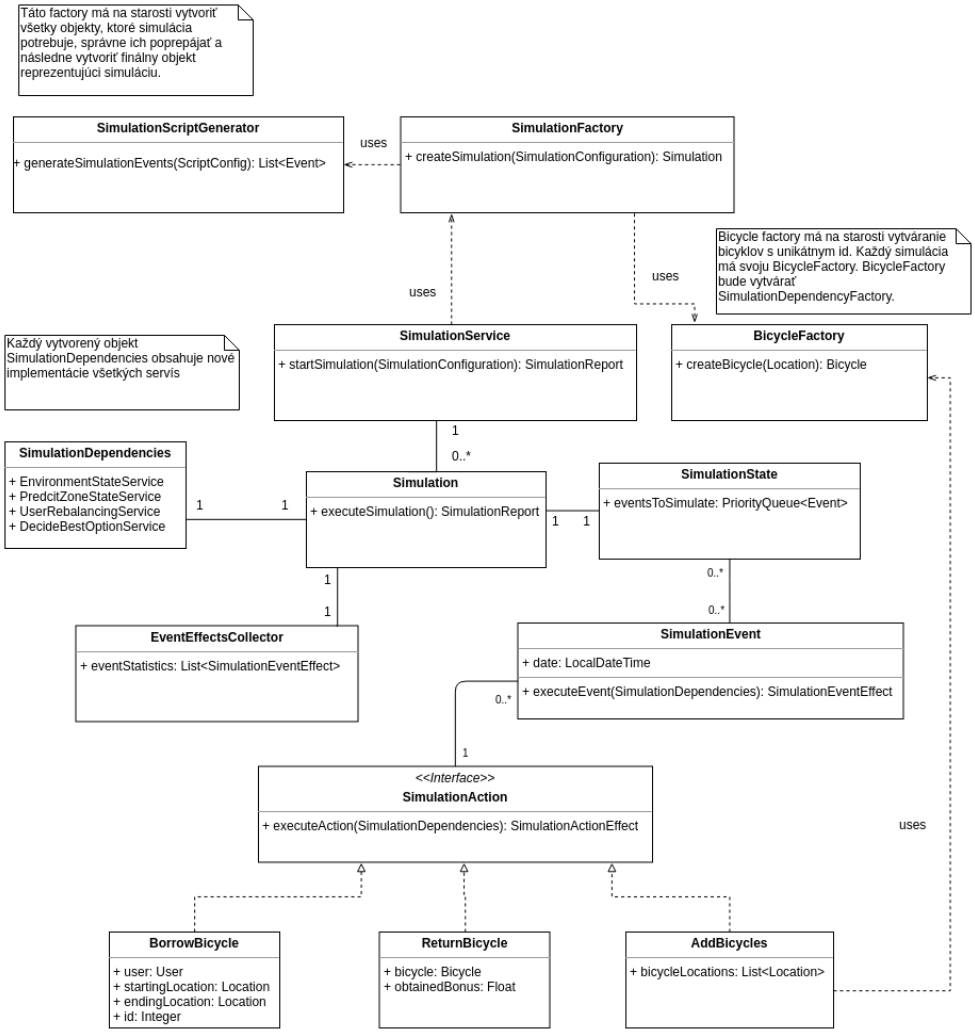


Obr. 3.3: Diagram tried simulačného systému (1)

Obsahuje jedinú metodu `clusterToZones`, ktorá prijíma plochu mesta a vrácia zóny. Táto servisa netuší nič o aktívnych a neaktívnych zónach, ktoré boli popísané v sekcii 3.1.3. Celá funkcionalita spočíva v jednoduchom rozdelení mesta do štvorcových neprekrývajúcich sa zón a to tak aby pokryli celú plochu. Na vytvárania zón používa triedu `ZoneFactory`, ktorá každej zóne priradí unikatné id.

3.3.2 Stav sveta

V rámci simulácie je potrebné udržiavať aktuálny stav sveta a jednotlivým udalostiam poskytovať základné informácie o najbližšom voľnom bicykli, aktívnych zónach, aktuálnom čase, ceste z bodu A do bodu B, predošlých a aktuálnych stavoch zón. Udalosti musia mať taktiež možnosť meniť stav sveta. Modul teda poskytuje funkcionalitu aj na základné manipulovanie so svetom pomocou metod na pridanie/odobranie voľného bicykla a na posúvanie času. V prípade, že nejaká udalosť vykoná nelogickú akciu, napr. pokúsi sa vypožičať bicykel, ktorý napr. neexistuje alebo sa pokúsi posunúť čas do minulosti, dôjde k výnimke `InvalidEnvironmentActionException`.

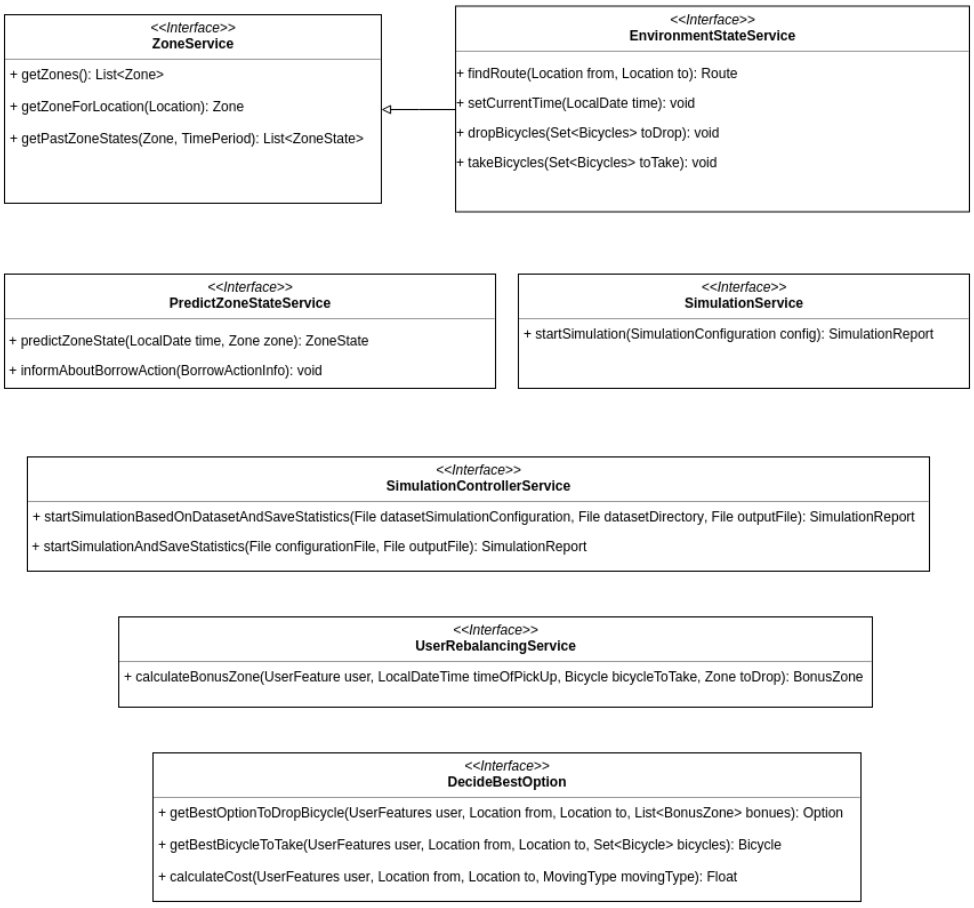


Obr. 3.4: Diagram tried simulačného systému (2)

Opisané funkcionality má na starosti modul `environment`, ktorý komunikuje s ostatnými modulmi pomocou rozhrania `EnvironmentStateService`.

3.3.3 Náhrada za prediktívny model

Na určovanie vhodnej výšky odmeny je potrebné predikovať budúci stav zóny. Keďže tomuto problému sa nevenujem, náhradu za túto funkcionalitu v simulácii poskytuje modul `rides-prediction`, ktorý obsahuje `PredictZoneStateService`. Ako som ukázal v predošlých sekciách, nie je možné vytvoriť perfektný prediktor, preto počas behu simulácie jednotlivé udalosti informujú túto servisu o prevedených udalostiach.



Obr. 3.5: Základné rozhrania modulov

Medzi základné metody rozhrania patrí `predictZoneState`, `informAboutBorrowAction` a `calculateProbabilityOfRide`. Metoda `predictZoneState` predikuje stav zóny pre nejaký časový úsek. Nevalidný dotaz o predikcii stavu zóny do minulosti vyvolá výnimku `InvalidPredictZoneStateRequestException`.

Pomocou druhej metody `informAboutBorrowAction` informujú práve prebiehajúce udalosti o ich aktuálnom/budúcom efekte na stavy zón, na základe čoho si servisa upravuje prediktívny model. Informácie, ktoré si prediktor aktualizuje sú predpokladaný počet bicyklov, ktoré by mali byť v zóne zaparkované v nejakom časovom úseku a predpokladaný dopyt po bicykloch v zóne pre nejaký časový úsek. Toto je možné z dôvodu, že na začiatku každej udalosti vypožičania bicykla je jednoznačne určené, či si užívateľ nejaký bicykel vypožičia, ak áno, tak ktorý a taktiež kde ho nakoniec zaparkuje.

Ako som spomenul v sekcii 3.1.5 v prípade, že spúšťam viacero simulácií za sebou, s rovnakým scénarom, je možné nadchádzajúcej simulácii poskytnúť

informácie o efekte vykonaných udalostí z predošlých behov simulácie.

Pre priblíženie popíšem ako funguje upravovanie modelu prediktora na následujúcom príklade. Každá udalosť scénara je identifikovateľná pomocou vygenerovaného id. Na začiatku zbehne prvá simulácia, ktorá vo svojej šatistickej správe vráti efekt jednotlivých udalostí spolu s ich id. Toto info použije prediktor v ďalšom behu aby dokázal lepšie predikovať stavy zón, keďže druhá simulácia sa spúšťa s rovnakým scénarom ako prvá. V momente, keď sa vykoná nejaká udalosť v druhej simulácii, informuje prediktor o jej novom efekte pomocou metody `informAboutBorrowAction`. Ten sa pozrie na jej predošlý efekt z prvej simulácie, ktorý vie identifikovať na základe id, porovná či sa zmenil oproti aktuálnemu efektu a ak áno, upraví si model odvolaním predošlého efektu udalosti a aplikuje jej nový efekt.

Posledná metoda `calculateProbabilityOfRide` zráta pravdepodobnosť jazdy pre usporiadanú dvojicu zón t.j. pravdepodobnosť jazdy z prvej zóny do druhej zóny. Na to využíva informácie o aktuálne vykonávaných jazdách a taktiež informácie z predošlých behov simulácie, v prípade, že je s nimi prediktor nainicializovaný.

Prediktor si ukladá predikcie a na konci simulácie, kedy už sú známe všetky predošlé stavy zón, sa vyhodnucuje jeho presnosť.

3.3.3.1 Trieda FakePredictZoneStateServiceImpl

V prípade, kedy sa simulácia spúšťa bez odmeňovacích stratégií, nie je nutné používať prediktor, avšak je potrebné zachovať informácie o udalostiach, ktoré sú prediktoru poskytované cez `informAboutBorrowAction`, keďže sa môžu využiť v nadchadzajúcich simuláciách. Z tohoto dôvodu existuje trieda `FakePredictZoneStateServiceImpl`, ktorá implementuje rozhranie `PredictZoneStateService` a ktorá sa používa v simulácií s vypnutým odmeňovaním, pričom neobsahuje žiadnu logiku. Jedná sa teda o „mock“[6].

3.3.4 Odmeňovacie stratégie

Modul `user-rebalancing` obsahuje implementáciu stratégií, ktorých funkcionalitu definuje rozhranie `UserRebalancingService`. Základná metoda rozhrania je `calculateBonusZone`, ktorá prijíma vlasnosti užívateľa (priemerná rýchlosť na bicykli atp.), bicykel, ktorý si užívateľ vypožičia, čas vypožičania a zónu, do ktorej by bicykel zaparkoval. Každá odmeňovacia stratégia navyše dedí od triedy `AbstractIncentiveStrategy`. Všetky stratégie sú konfigurovateľné pomocou parametrov.

[6]Význam slova mock, vo softwarovom svete, by sa dal vysvetliť ako objekt, ktorý kontrolovane napodobňuje správanie iného objektu.

3.3.4.1 Trieda FakeUserRebalancing

Táto „mock“ trieda sa používa pri simulácii s vypnutým odmeňovaním zón, podobne ako trieda 3.3.3.1. To znamená, že zakaždým na volanie metody `calculateBonusZone` odpovie nulovým bonusom.

3.3.5 Rozhodovanie užívateľa

Modul `decide-best-option` poskytuje funkcionality, ktoré simulujú rozhodovanie užívateľa podľa 2.2. Základné rozhranie poskytuje 3 metody: `calculateCost` zráta cenu cestu medzi 2 bodmi, metoda `getBestOptionToDropBicycle` nájde spomedzi predaných zón a nim prislúchajúcich bonusov najlepšiu možnosť, kde zaparkovať bicykel a metoda `getBestBicycleToTake` nájde najlepší bicykel na vypožičanie v okolí užívateľa.

3.3.6 Simulácia

Modul `simulation` tvorí jadro celého simulačného systému. Základné rozhranie `SimulationService` poskytuje metodu na vytvorenie a spustenie simulácie. Servisa vytvára simuláciu pomocou triedy `SimulationFactory`, ktorá okrem iného na základe konfigurácie vytvorí a nainicializuje servisné objekty modulov používaných v simulácii a vygeneruje scénar. Servisa následne simuláciu spustí a vráti výsledok.

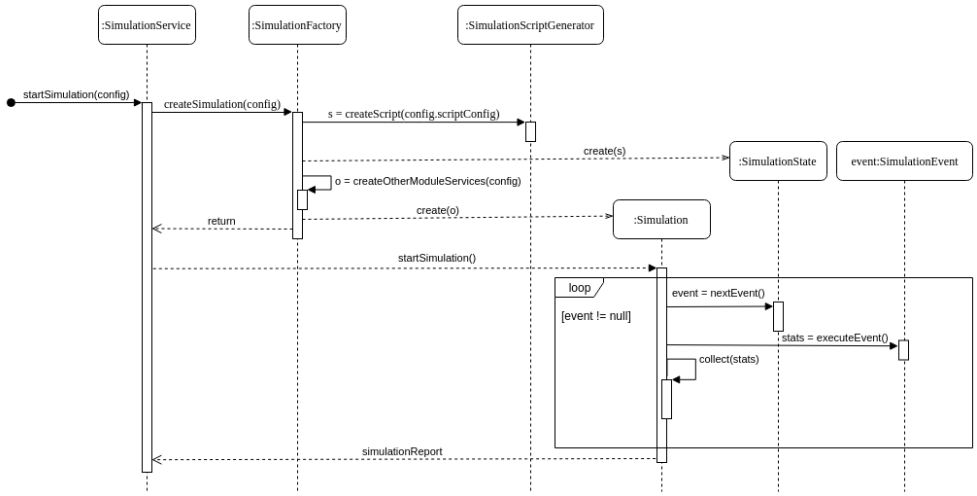
Trieda `Simulation` – výsledok `SimulationFactory` – si udržuje frontu udalostí, ktoré má ešte odsimulovať pomocou triedy `SimulationState`. Štatistiky o vykonaných udalostiach simulácie si udržuje trieda `EventEffectsCollector`. Na sekvenčnom diagrame 3.6 je možné detailnejšie vidieť celkový beh simulácie.

Udalosť scénara reprezentuje trieda `SimulationEvent`, ktorá má 2 atribúty: čas, kedy sa má vykonať a akciu, ktorú ma vykonať. Medzi možné akcie simulácie patrí pridanie nových bicyklov - `AddBicycles`, vypožičanie bicyka - `BorrowBicycle` a vrátenie bicykla `ReturnBicycle`. Každá akcia implementuje rozhranie `SimulationAction`. Hlavná metoda tohoto rozhrania, `executeSimulationAction`, vykoná akciu a vráti efekt tejto akcie, ktorý reprezentuje trieda `SimulationActionEffect`. Tá obsahuje atribúty ako cesty, ktoré sa vykonali v rámci akcie, obdržané bonusy, miesta kde sa rozhodol užívateľ nevyužiť služby BSS, vypožičané, pridané a vrátené bicykle.

3.3.7 Spúšťanie simulácie a ukladanie výsledkov

Modul `simulation-controller` má na starosti načítavanie konfigurácie z externého súboru, ktorá je uložená v json formáte. Taktiež podporuje vytváranie konfigurácie na základe historických jázd BSS[7]. Cesta k súboru s konfiguráciou

[7]Pre bližšie info o štruktúre konfigurácie viď zložku data/configuration kde sú uložené json schémy pre obe konfigurácie: klasická konfigurácia a konfigurácia z historických jázd.



Obr. 3.6: Sekvenčný diagram celkového behu simulácie.

a ku súboru, kde sa má uložiť výsledok, sa predáva programu ako parameter pri spúšťaní. Trieda `SimulationControllerServiceImpl` načíta konfiguráciu zo súboru, na základe nej spustí simuláciu bez vyvažovania, s jej výsledkami nainicializuje a spustí novú simuláciu so zapnutým vyvažovaním. Výsledky oboch simulácií uloží do súboru, pričom výsledok simulácie bez vyvažovania sa ukladá do súboru s rovnakou cestou ako predaný výsledný súbor avšak s extra prefixom *no-rebalancing*.

3.3.8 Vykresľovanie simulácie

Základná trieda modulu `simulation-drawing` je `SimulationGraphicalController`. Jedná sa o JavaFX Controller [18], ktorého definícia je v `simulation.fxml` súbore. Aktuálny stav sveta, ktorý má vykresliť si udržuje pomocou triedy `EnvironmentSnapshot`, ktorá si udržuje všetky zóny, bicykle atp. Trieda `RenderVisitor` vykresľuje stav sveta na JavaFX Canvas [18].

3.3.9 Spoločne triedy a utility jednotlivých modulov

Keďže jednotlivé moduly zdieľajú základné triedy ako zóna, časový úsek atp. a taktiež niektoré utility, vytvoril som modul `shared`, ktorý tieto spoločné triedy obsahuje a na ktorom sú závislé všetky ostatné moduly.

Kapitola 4

Implementácia

V tejto kapitole rozoberiem môj prístup k vývoju a zaujímavé problémy, s ktorými som sa pri implementovaní jednotlivých modulov stretol. Kapitolu zavŕšujem sekciou o možných rozšíreniach simulačného systému.

4.1 Použité technológie

Simulačný systém a odmeňovacie stratégie som implementoval v programovacom jazyku Java. Vzhľadom k súčasnému rozšíreniu som použil verziu 8. Na zostavenie aplikácie som použil nástroj Gradle [19]. Pre účely logovania používam knižnicu log4j [20]. Na vykresľovanie simulácie som použil framework JavaFX [18]. Na vytváranie náhodných rozdelení som použil knižnicu Commons math [21]. Na vytvorenie objektu z textu vo formáte json som použil knižnicu Jackson [22].

4.2 Vývojové a testovacie prostredie

Na vývoj a testovanie som použil osobný PC s 16GB pamäťou a s 4 jadrovým procesorom, pričom maximálna frekvencia jedného jadra je 3 GHz. Na vývoj a testovanie som použil 64-bitovú JVM Oracle Hotstop, verzia 1.8.0_191. Maximalnú veľkosť haldy JVM som obmädzil na 10 GB. Ostatné nastavenia som ponechal nezmenené. Operačný systém, ktorý som použil je Ubuntu 16.04.

4.3 Vývoj

V rámci vývoju som používal verzovací nástroj git. Pri každom zostavovaní aplikácie sa automaticky spúšťaju testy, ktoré kontrolujú funkcionalitu aplikácie. V kóde som sa snažil o čo najviac nemenných objektov, keďže tento prístup zvyšuje bezpečnosť a priehľadnosť kódu. Na riadenie behu programu v nečakaných situáciách používam kontrolované výnimky.

4.4 Štruktúra projektu

Každý modul obsahuje vopred danú štruktúru balíčkov, kde rozlišujem triedy, ktoré vykonávajú nejakú logiku v rámci modulu – balíček `logic`, servisné triedy – balíček `service` a objekty, ktoré neobsahujú skoro žiadnu logiku a sú teda prevažne len nositeľmi atribútov – balíček `model`.

Všetky moduly sú obalené v hlavnom Gradle projekte. V jeho skripte sú definované premenné, ktoré sa používajú na určenie verzií knižníc naprieč celým projektom. Taktiež pre všetky ostatné moduly (Gradle podprojekty) nastavuje základnú zostavovaciu konfiguráciu.

4.5 Generátor scénara

Na vygenerovanie scénara atp. je nutné vytvoriť náhodné rozdelenia. Na to v kóde jednotne používam triedu `DistributionUtils`, ktorá zakaždým nainicializuje novo-vytvorené náhodné rozdelenie s iným číslom, ktoré čerpám z nejakej vopred určenej postupnosti. Týmto zabránim tomu aby vznikli dve rovnaké náhodné rozdelenia v jednom behu simulácie. Taktiež, v prípade, kedy chcem spustiť v jednom behu programu 2 simulácie, ktoré majú byť identické až na napr. odmeňovacie stratégie, tak to jednoducho dokážem resetovaním postupnosti čísla v `DistributionUtils`, s ktorým tieto náhodné rozdelenia inicializujem. Týmto zaručím, že oba behy simulácie budú používať rovnaké náhodné rozdelenia.

Scénar vytváram pomocou triedy `ScriptGenerator`. Jedná sa o thread-safe singleton, ktorý si neudržiava žiadny stav. V prípade, že generujem scénar na základe datasetu, konvertujem historické jazdy z CSV súboru do internej štruktúry – trieda `DatasetRide`. To zaručuje jednoduchú rozšíriteľnosť. V prípade, ak by v budúcnosti chcel pridať generovanie scénara z datasetu, v ktorom historické jazdy majú iný formát, než formát citibike BSS, stačí vytvoriť nový konverter, ktorý stransformuje jazdy nového datasetu do internej reprezentácie `DatasetRide` a zvyšok kódu sa skoro nijak nemení.

V prípade generovania scénaru z datasetu, musím vytvoriť vlastné náhodné rozdelenie. Na to používam triedu `EnumeratedIntegerDistribution` z knižnice `Apache commons math3`. Táto trieda prijíma v konštruktori pole čísel (obor hodnôt náhodnej veličiny) a pole určujúce ich pravdepodobnosť, ktorú odhadujem pomocou následujúceho vzťahu:

$$p = \frac{\text{počet výskytov hodnoty v datasete}}{\text{celkový počet hodnôt v datasete}} \tag{4.1}$$

4.6 Vykresľovanie simulácie

Grafické rozhranie, ktorého štruktúra je uložená v fxml súbore, som vytvoril pomocou nástroja `SceneBuilder`. Za behu aplikácie z tohotu súboru vytváram

objekt FXML `Controller`, ktorý zobrazím v aplikačnom okne.

Potom, čo na plátno vykreslím nejakú udalosť, napr. cestu užívateľa, tak vykreslená udalosť začne byť stále viac a viac priehľadná až zmizne úplne. Vykresľované udalosti sú uložené v `DelayQueue` v triede `EnvironmentSnapshot`. Priehľadnosť objektu nastavujem podľa toho ako dlho sa v tejto fronte nachádza. V každom kroku vykresľovania odoberiem z fronty všetky objekty, ktoré vracia metoda `poll`, to znamená objekty, ktoré sú vo fronte pridlho a tým pádom ich prestanem vykresľovať.

Všetky vykresľované udalosti dedia od generickej triedy `DelayObject`, ktorá ponúka základnú funkcionalitu pre vykresľovanie a taktiež implementuje rozhranie `Delayed`, ktoré umožňuje tieto udalosti pridávať do `DelayQueue`.

Keďže rozmery sveta nemusia byť rovné veľkosti JavaFX Canvas, na ktorý tieto udalosti vykresľujem, vytvoril som mapper, ktorý dokáže každému bodu sveta, ktorý je daný svetovými koordinátmi, priradiť bod na JavaFX Canvas.

4.7 Tvorba grafov z výsledkov simulácie

Kvôli potrebám vizualizácie štatistík zo simulačnej správy som vytvoril python skript, ktorý prečíta data z nejakého súboru v json formáte a vytvorí základnú sadu grafov. Na načítanie dát zo súboru používam modul `json` a na vytváranie grafov modul `matplotlib`. Cestu k súboru so štatistikami predávam skriptu cez parameter pri spúšťaní. Python som zvolil kvôli jednoduchej implementácii.

4.8 Stav sveta

Celkový stav sveta udržuje trieda `WorldRepository`. Trieda obsahuje informácie o čase, aktívnych zónach, aktuálnych a predošlých stavoch zón. Kvôli rýchlemu vyhľadávaniu si udržujem niektoré informácie v kolekciách `HashMap`.

4.9 Testovanie

Neoddeliteľnou súčasťou tvorby softwaru je testovanie, ktoré sa delí na automatické a manuálne. Automatickými testami som sa snažil primárne pokryť jadro celého systému a triedy, ktoré spracúvavajú simulačné výsledky. Na testovanie využívam framework JUnit verziu 4. Na manuálne testovanie som využil grafické vyobrazenie simulácie pomocou modulu `simulation-drawing`.

4.10 Spúšťanie aplikácie

Projekt obsahuje 2 spustiteľné aplikácie. Ako som spomenul v predošlých sekciách, jedná sa o `simulation-controller` a `simulation-drawing`. Kvôli

jednoduchému spúšťaniu som vytvoril vlastnú gradle úlohu, ktorú je možné spúšťať pomocou gradle skriptu, v jednotlivých moduloch, následovne `gradle customFatJar`. Tá zostaví celú aplikáciu do spustiteľného jar súboru. Program prijíma povinne buď 2 alebo 3 parametre. V prípade, že spúšťam simulácie z historických jázd tak prvý parameter musí byť cesta k súboru, v ktorom je konfigurácia uložená v json formáte, druhý parameter je cesta k súboru do ktorého sa zapíšú výsledky simulácie a tretí parameter je cesta ku zložke, ktorá obsahuje csv súbory s historickými jazdami, z ktorých sa vygeneruje scénar. V prípade, že spúšťam simulácia s klasickou konfiguráciou predávam len prvé 2 parametre.

4.11 Možnosti rozšírenia

Modul `simulation-controller` by sa dal rozšíriť o webové rozhranie, pomocou ktorého by sa dala spustiť simulácia na základe predanej konfigurácie. Výsledok simulácie by sa mohol následne ukladať do NoSQL databázy vzhľadom ku pestrej štruktúre štatistickej simulačnej správy. Po nasadení webovej aplikácie na nejaký server, by mohol napr. analytik kedykoľvek spúšťať rôzne simulácie a sledovať ich výsledky uložené do databázy.

Kapitola 5

Experimentálne vyhodnotenie

V tejto kapitole priblížim 3 konfigurácie simulácie, pomocou ktorých následne vyhodnotím efektivitu odmeňovacích stratégií. Pri hodnotení efektivity sa budem primárne pozerať na 2 štatistiky: hodnota objektívnej funkcie 2.2 a výška rozdanej odmeny v eurách.

Jednotlivé konfigurácie sa od seba značne odlišujú. Prvá z nich generuje miesta a čas udalosti simulácie z uniformného rozdelenia, druhá generuje najviac udalostí okolo 14 hodiny, pričom miesta týchto udalostí sa nachádzajú primárne v centre mesta a tretia – posledná – konfigurácia, vznikla na základe historických dát reálneho SBSS, čiže najviac sa približuje realite. Tieto konfigurácie podrobnejšie popíšem v nasledujúcich sekciách.

Pomocou aplikácie `simulation-controller` som spustil pre každú konfiguráciu 3 simulácie: bez odmeňovania, s fixnou odmeňovacou stratégiou a s dynamickou odmeňovacou stratégiou. Tie sa od seba líšia len v odmeňovacej stratégii; inak majú všetky paramatre rovnaké.

5.1 Simulačné konfigurácie

V tejto sekcii detailnejšie popíšem jednotlivé konfigurácie, pomocou ktorých som testoval efektivitu stratégií, pričom sa budem zameriavať len na dôležité parametre. Kompletné konfigurácie s popisom parametrov je možné nájsť na priloženom médiu v zložke data/configuration.

Jednotlivé konfigurácie väčšinu parametrov zdieľajú, líšia sa prevažne len vo veľkosti mesta, počte bicyklov, počte udalostí, rozmiestnení udalosti vo svete a v čase. Parametre, ktoré zdieľajú sú:

- šírka a výška zóny, ktorá je rovná 300 m,
- hodnota *minimum* z rovnice 2.5 je rovná 0, keďže v zóne vždy je vhodné mať aspoň jeden bicykel, kvôli nečakaným udalostiam,
- hodnota minimálneho počiatočného počtu bicyklov v zóne je 2,

- počiatočný čas simulácie, ktorý je rovný začiatku dňa 01.01.2019,
- náhodná veličina z normálneho rozdelenia udávajúca priemernú rýchlosť užívateľa peši so strednou hodnotou 1,5 m/s a smerodajnou odchýlkou 0,05 m/s [23],
- náhodná veličina z normálneho rozdelenia, udávajúca priemernú rýchlosť užívateľa na bicykli so strednou hodnotou 4,3 m/s a smerodajnou odchýlkou 0,1 m/s,
- pravdepodobnosť, že užívateľ nepríjme odmenu nezávisle na jej výške, ktorá je rovná 15 %,
- náhodná veličina z normálneho rozdelenia, ktorá reprezentuje koeficient udávajúci lineárny vzťah medzi výškou bonusu a cenou extra cesty, ktorú je užívateľ ochotný za tento bonus prejsť, so strednou hodnotou 220 a smerodajnou odchýlkou 10, jedná sa teda o parameter α z rovnice 2.3,
- náhodná veličina z normálneho rozdelenia, ktorá reprezentuje maximálnu cenu cesty pri vypožičiavaní, ktorú je ochotný užívateľ prejsť za bicyklom, so strednou hodnotou 287 a smerodajnou odchýlkou 20,
- výška minimálneho bonusu je 0,625 €, výška štandardnej[8] odmeny je 1,25 a € hranica nedokonalosti prediktoru je 1.

Pravdepodobnosť, že užívateľ nepríjme odmenu nezávislé na jej výške, parametre normálneho rozdelenia pre maximálnu cenu cesty a koeficient udávajúci lineárny vzťah medzi výškou bonusu a cenou extra cesty, som zvolil na základe ankety urobenej v rámci práce [2], podobne ako šírku a výšku zóny, čo som už popísal v predošlých kapitolách.

Počet bicyklov a udalostí som v jednotlivých konfiguráciach volil tak, aby sa mal FBSS možnosť vyvážiť ale zároveň aby bol čiastočne vyťažený.

5.1.1 Uniformná konfigurácia

Uniformná konfigurácia rozdeľuje udalosti vypožičania rovnomerne po celom meste a taktiež rovnomerne v čase. Šírka a výška mesta je 3000 m, to znamená, že mesto bude rozdelené do 100 zón. Počet udalostí vypožičania bicykla je 17 000. Počiatočné a koncové miesta jázd sú vyberané rovnomerne náhodne z celej plochy mesta. Tieto udalosti sú rovnomerne rozmiestnené v čase v rámci cca 6,5h intervalu. Počet bicyklov je 650. Ten som zvolil na základe priemernej dĺžky jazdy užívateľa, čo je približne do 10 minút[9]. Teda na základe týchto

[8] Pre fixnú a taktiež dynamickú stratégiu.

[9] Konkrétny výpočet bol následujúci: počet obsadených bicyklov v nejakom čase je cca 430 = 17000/6.5/(60/10). Aby systém nebol vždy kompletne vyťažený, k tomuto číslu som pridal ešte polovicu.

informácií môžem očakaváť, že priemerný počet vypožičaných bicyklov v nejakom čase bude rovný 2/3 celkového počtu bicykov.

Táto konfigurácia sleduje ako si odmeňovacia stratégia dokáže poradiť s neprestajne vyťaženým systémom naprieč celou plochou mesta.

5.1.2 Normálna konfigurácia

Šírka a výška mesta v tejto konfigurácií je 5100 m. Normálna konfigurácia generuje čas udalostí vypožičania bicykla z normálneho rozdelenia so strednou hodnotou 52 200 a smerodajnou odchýlkou 7 200. Počiatočné a koncové miesta jázd sú generované z 2 normálných rozdelení, pre zem. šírku/výšku. Kombinácia stredných hodnôt týchto rozdelení je bod v strede mesta. Smerodajná odchýlka pri oboch rozdeleniach je rovná cca 1250 m. Z toho teda plynia, že 68 % jázd má počiatočné alebo koncové miesto vo štvorci, ktorého stred je stred mesta a dĺžka strany je 2,5 km. Počet udalostí je 50 000 a počet bicyklov je 2 000, ktorý som určil podobným spôsobom ako pri uniformnej konfigurácii.

Táto konfigurácia sleduje ako si odmeňovacia stratégia dokáže poradiť s kratšími, ale veľkými vlnami dopytu po bicykloch z centra mesta. To pripomína situáciu pooebede z reálneho FBSS, kedy si väčšina ľudí po práci vypožičiava bicykel práve z centra mesta.

5.1.3 Konfigurácia z historických jázd

V tejto konfigurácií simulujem reálny SBSS citibke NYC. Mapa sa skladá primárne z centra Manhattan ale zahŕňa aj časť Boston. Historické jazdy, z ktorých vytváram počiatočné a cieľové miesta udalostí sú z januára 2019, avšak vynechávam víkendy a sviatky, keďže štýl jázd v týchto dňoch sa značne líši od pracovného dňa. Počet udalostí v dni je 40 000 a počet bicyklov je 8 000, čo sú viacmenej réalne údaje tohoto SBSS. Počet aktívnych zón v systéme je približne 1075, zvyšné zóny mesta sú vodnaté plochy atp. Udalosti sú rozmiestnené v čase taktiež podľa historických jázd. To znamená, že systém je najviac vyťažený ráno, kedy ľudia chodia do práce a poobede, kedy naopak prichádzajú z práce domov.

Táto konfigurácia sa snaží získať odhad efektu odmeňovacej stratégie v reálnom BSS. Oproti ostatným konfiguráciám je mesto, v ktorom prebieha simulácia podstatne väčšia, avšak s 40 000 udalosťami FBSS nie je až tak vyťažený ako s ostatnými konfiguráciami.

5.2 Výsledky

V tejto sekcii popíšem dosiahnuté výsledky odmeňovacích stratégií, ktoré boli vyhodnotené na simulácii. Okrem pôvodných konfigurácií som stratégie vyhodnotil taktiež na konfiguráciách, ktoré sú zhodné s pôvodnými až na menší

počet bicyklov a väčší počet udalostí. Týmto som chcel otestovať efektivitu stratégií v nadmieru vyťaženom BSS.

Jednotlivé výsedky behov simulácií je možné vidieť v tabuľkách 5.1 až 5.6. Stĺpec Service level udáva hodnotu objektívnej funkcie, stĺpec No service events, udáva počet udalostí kedy sa užívateľ rozhodol nevypožičiať bicykel a stĺpec Náklady predstavuje náklady na vyvažovanie v eurách. Kvôli priehladnosti som v tabuľkách označil počet, kedy užívateľ prijal odmenu za odnos bicykla do inej, než jeho pôvodne cieľovej zóny ako AR. Keďže v simulácií využívam pseudonáhodné generátory, hodnoty v tabuľkách sú spriemerované z 3 behov simulácií (každá s iným random seed).

Výsledky behov simulácie v json formáte je možné nájsť na priloženom médiu v zložke data/results.

5.2.1 Výsledok uniformnej konfigurácie

Detailnejší priebeh simulácie s uniformnou konfiguráciou je možné vidieť v dodatku na obrázku B.1 kde je vidno, že systém bol vyťažený počas celého behu simulácie, pričom ku najviac nevypožičaniam dochádzalo okolo 4 hodiny, kedy bol systém najviac nevyvážený. Stratégia, ktorá mala najväčšiu hodnotu objektívnej funkcie je fixná stratégia.

Ak porovnám behy s a bez stratégií, dokážem určiť koľko systém stálo znížiť No service events o 1. V prípade fixnej stratégie to je 18,4 € a v prípade dynamickej 12,5 €. Najúspornejšia stratégia bola teda dynamická, keďže jedna ušetrená jazda[10] ju stála o 5,9 € menej než v prípade fixnej stratégie.

V tabuľke 5.2.1 je možné vidieť výsledky simulácie s uniformnou konfiguráciou, kde som zväčšil počet akcií a zmenšil počet bicyklov. Týmto som chcel porovnať ako efektívne budú jednotlivé stratégie vo vyťaženom systéme. Z tabuľky je vidno, že fixná stratégia má znova lepšiu hodnotu objektívnej funkcie, avšak dynamická je ďaleko viac úspornejšia, pričom počet No service events v prípade fixnej a dynamickej sa značné nelíši.

Stredná hodnota chyby prediktoru bola, v prípade fixnej stratégie, rovná 0,13 so smerodajnou odchýlkou 2,4.

Stredná hodnota chyby prediktoru bola, v prípade dynamickej stratégie, rovná 0,07 so smerodajnou odchýlkou 1,86.

Tieto výsledky nie sú úplne uspokojivé keďža cena jednej jazdy v reálnom BSS citibike NYC je okolo 2,5 €. Inými slovami, cena za vyvažovanie značne prevyšuje zisky získané z extra jázd, ku ktorým by bez vyvažovania nedošlo.

5.2.2 Výsledok normálnej konfigurácie

Detailnejší priebeh simulácie s normálnou konfiguráciou je možné vidieť v dodatku na obrázku B.2. Systém bol najviac vyťažený okolo 14 hodiny čo sa

[10] Ušetrenou jazdou sa myslí jazda, ku ktorej by bez vyvažovania nedošlo.

Odmeňovacia stratégia	Service level	No service events	Náklady	AR
Žiadna	0,964	618	–	–
Fixná	0,992	142	8 745	5 477
Dynamická	0,984	273	4 300	3 626

Tabuľka 5.1: Štatistiky simulácie s uniformnou konfiguráciou 5.1.1

Odmeňovacia stratégia	Service level	No service events	Náklady	AR
Žiadna	0,854	3 363	–	–
Fixná	0,897	2 379	23 530	7 349
Dynamická	0,889	2 550	14 677	6 936

Tabuľka 5.2: Štatistiky simulácie s uniformnou konfiguráciou 5.1.1, kde bol pozmänený počet vypožičaní na 23 000 a počet bicyklov na 500

Odmeňovacia stratégia	Service level	No service events	Náklady	AR
Žiadna	0,961	1930	–	–
Fixná	0,990	483	28 942	15 824
Dynamická	0,980	1 025	15 409	10 648

Tabuľka 5.3: Štatistiky simulácie s normálnou konfiguráciou 5.1.2

odrazilo na rýchlosti rastu celkového počtu nevypožičaní a taktiež na celkový počet prijatých odmien.

Z tabuľky 5.2.2 je vidno, že najväčšiu hodnotu objektívnej funkcie dosahovala fixná stratégia. Najúspornejšia bola naopak dynamická a to s 17 € za jednu ušetrenú jazdu.

V druhej simulácii, kedy bol systém omnoho viac vyťažený, viď tabuľka 5.2.2, bol rozdiel v objektívnej funkcie medzi fixnou a dynamickou stratégiou zanedbateľný. Avšak náklady systému boli pri dynamickej stratégii 1,39 krát menšie než v prípade fixnej stratégie.

Stredná hodnota chyby prediktoru bola, v prípade fixnej stratégie, rovná 0,02 so smerodajná odchýlkou 1,87.

Stredná hodnota chyby prediktoru bola, v prípade dynamickej stratégie, rovná -0,014 so smerodajná odchýlkou 1,53.

5.2.3 Výsledok konfigurácie z historických jázd

Detailnejší priebeh simulácie s konfiguráciou z historických jázd reálneho BSS, je možné vidieť v dodatku na obrázku B.3. Na grafe „Počet stojacich bicyklov“ sú 2 doliny: jedna okolo 9 hodiny, kedy ľudia prichádzajú do práce a druhá okolo 17 hodiny, kedy ľudia naopak z práce odchádzajú. Najviac nevyvážený systém bol zrána okolo 9 hodiny, kedy počet nevypožičaní rástol najrýchlejšie.

Odmeňovacia stratégia	Service level	No service events	Náklady	AR
Žiadna	0,833	10 028	–	–
Fixná	0,857	8 594	46 297	12 973
Dynamická	0,854	8 733	33 280	13 042

Tabuľka 5.4: Štatistiky simulácie s normálnou konfiguráciou 5.1.2, kde bol pozmänený počet vypožičaní na 60 000 a počet bicyklov na 1 600

Odmeňovacia stratégia	Service level	No service events	Náklady	AR
Žiadna	0,976	993	–	–
Fixná	0,986	559	6 013	4 063
Dynamická	0,980	773	2 364	2 517

Tabuľka 5.5: Štatistiky simulácie s konfiguráciou z historických jázd 5.1.3

Odmeňovacia stratégia	Service level	No service events	Náklady	AR
Žiadna	0,888	7 282	–	–
Fixná	0,936	4 113	31 684	18 652
Dynamická	0,911	5 774	14 181	11 671

Tabuľka 5.6: Štatistiky simulácie s konfiguráciou z historických jázd 5.1.3, kde bol pozmänený počet vypožičaní na 65 000 a počet bicyklov na 4 600

Najväčšiu hodnotu objektívnej funkcie dosahovala fixná stratégia. Najúspornejšia stratégia bola dynamická, ktorej cena za jednu ušetrenú jazdu bola 9,4 €.

V tabuľke 5.2.3 je možné vidieť výsledky simulácie s konfiguráciou z historických jázd, v ktorej som navýšil počet udalostí a zmenšil počet bicyklov. Touto simuláciou som sledoval ako sa bude systém správať v prípade, kedy je viac vyťažený než obyvkle. Fixná stratégia dosahuje znova lepšiu hodnotu objektívnej funkcie, avšak má viac než dvojnasobné náklady oproti dynamickej stratégii.

Stredná hodnota chyby prediktoru bola, v prípade fixnej stratégie, rovná -0.02 so smerodajnou odchýlkou 1,03.

Stredná hodnota chyby prediktoru bola, v prípade dynamickej stratégie, rovná -0.04 so smerodajnou odchýlkou 0,82.

5.2.4 Rozbor výsledkov

Z predošlých výsledkov je zrejmé, že fixná stratégia dosahuje pri každej konfigurácii väčšie hodnoty objektívnej funkcie, avšak dynamická stratégia sa niekoľkokrát priblížila ku efektivite fixnej stratégií a to za podstatne menšiu výšku rozdaného bonusu. Dynamickej stratégii sa darilo viac vo vyťaženej

simulácii. Fixnej stratégii sa naopak darilo v klasickej simulácii, ktorá sa približuje k reálnej vyťaženosti BSS.

Na to aby sa BSS oplatilo používať odmeňovacie stratégie s cieľom vyvážiť systém, musí byť celkový zisk z extra jázd, ku ktorým by bez vyvažovania nedošlo, vyšší než celkové náklady na vyvažovanie. Na zisk BSS sa dá pozerať z rôznych hľadísk, napríklad z hľadiska celkovej spokojnosti zákazníkov, z finančného hľadiska atp. V prípade, že sa BSS pozerá na zisk iba z finančného hľadiska, tak na to aby sa systému oplatilo používať vyvažovacie stratégie, nesmie priemerná cena za jednu extra jazdu prekročiť finačnú hodnotu priemernej jazdy.

Pri pohľade na výsledky simulácie je rozumné si klásť otázku: ako je možné, že napr. v prípade fixnej stratégií 5.2.1, 5 477 ľudí zmenilo svoju cieľovú zónu ale počet No service events klesol len o 476? Nemal by byť počet prijatých odmien do iných zón skoro rovný „zachráneným“ jazdám?

Prvý z dvoch dôvodov, ktorý už bol spomenutý v predošlých kapitolách, je nedokonalosť prediktívneho modelu. To spôsobuje napr. situáciu, kedy systém užívateľovi za zaparkovanie bicykla do konkrétnej zóny ponúkne odmenu, keďže prediktor hlási, že tam je nedostatočný počet bicyklov ale v skutočnosti je zóna kompletne vyvážena. Dá sa konštatovať, že v takomto prípade systém ponúka odmenu úplne zbytočne. Alebo opačný prípad, kedy prediktívny model nesprávne označí zónu ako vyvaženú. Avšak, s dostatočne rozumným prediktorom nedokonalosť nespôsobuje značný problém.

Druhý podstatný dôvod, prečo systém ponúke viac než 5 000 odmien pre cca 600 No service events, popíšem na nasledujúcom príklade. Nech užívateľ plánuje jazdu zo zóny z_s do zóny z_e, pričom je ochotný bicykel zaparkovať v susednej zóne z_ne, za odmenu ponúknutú systémom. Nech predikovaný stav zóny z_e v predpokladanom čase príchodu je vporiadku, avšak v ďalšom časovom úseku, budú v zóne z_e chýbať dva bicykle. Keďže užívateľ nedostane pre zónu z_e žiadnu odmenu, odnesie bicykel do z_ne. Avšak, to spôsobí to, že v ďalšom časovom úseku budú v pôvodne cieľovej zóne z_e chýbať tri bicykle namiesto dvoch. Čiže systém užívateľovi zaplatí odmenu, avšak z dlhodobého hľadiska sa objektívna funkcia nijak nezmení, keďže v ďalšom časovom úseku dôjde ku jednému extra nevypožičaniu bicykla, v zóne z_e [11]. Inými slovami, ak sa užívateľ rozhodne na základe odmeny zaparkovať bicykel do inej, než jeho pôvodnej cieľovej zóny z_e, môže týmto spôsobiť nevyváženosť z_e v nasledujúcich časových úsekoch.

Ukazuje sa ako kľúčove neponúkať odmeny pre užívateľov, ktorých cieľová zóna bude v následujúcich časových úsekoch nevyvážená. Ak som do odmeňovacieho mechanizmu zapojil túto podmienku, počet No service events výrazne stúpol, avšak priemerná cena za jednu extra jazdu pri vyvažovaní sa znížila na približne 5 €, čo sa čiastočne približuje reálnej cene za jednu jazdu.

[11] Samozrejme to nastane len v prípade, kedy systém znova nepresvedčí iného užívateľa aby zaparkoval bicykel do z_e v ďalšom časovom úseku.

Keďze navrhnutý systém ponúka odmeny pre zóny na začiatku jazdy a to bez toho, aby sa užívateľa spýtal, ktorá zóna je jeho cieľová, tomuto problému sa nie je možné len tak ľahko vyhnúť. Navyše, aj keby sa BSS užívateľa pýtal, je nutné brať v úvahu, že užívateľ sa môže správať ako strategický agent, ktorý chce maximalizovať svoj zisk kvôli čomu môže klamlivo uvádzať svoju cieľovú destináciu.

Navrhnutý odmeňovací mechanizmus, popísaný v sekcii2.1.1, sa javí teda ako nepostačujúci aj napriek svojim výhodam, kedy si môže užívateľ lepšie naplánovať cestu, na základe čoho sa môže rozhodnúť prijať odmenu. Javí sa, že informácia o cieľovej destinácií je pre odmeňovacie stratégie kľúčova. Lepší prístup by teda bol ponúkať užívateľovi odmenu až na konci jeho jazdy t.j. potom, čo dorazí do svojej cieľovej destinácie. Aj v tomto prípade je možné, že užívateľ bude klamlivo žiadať o akciu vrátenia bicykla napr. 250 m pred jeho cieľovou destináciou s cieľom maximalizovať svoj zisk. Avšak, to je podstatne menší problém, keďže k týmto prípadom bude pravdepodobne dochádzať len v okolí cieľovej destinácie a systém sa taktiež časom môže naučiť o trendoch v správaní konkrétneho užívateľa (na základe jeho historických jázd) a na základe toho efektívne odhadovať jeho skutočnú cieľovú destináciu [2].

S týmto prístupom by teda dávalo zmysel oceňovať len zóny blízke k cieľovej destinácii, keďže je rozumné predpokladať, že existuje horná hranica dĺžky extra cesty, ktorú bude ochotný priemerný užívateľ prejsť za štandardnú odmenu. Tým pádom by sa podstatne ušetril aj CPU čas, keďže odmena by sa rátala len pre zopár blízkych zón.

Záver

V tejto bakalárskej práci som sa zaoberal možnosťami vyvažovania systému zdieľaných bicyklov (BSS) pomocou odmeňovania užívateľov. Na začiatku práce som sa zoznámil s existujúcimi riešeniami a opísal ich výhody. Následne som formálne definoval problém, ktorý budem v rámci tejto práce riešiť, pričom som taktiež navrhol vlastný BSS a vlastné odmeňovacie stratégie. Kvôli potrebám vyhodnotenia efektivity týchto stratégií som vytvoril simulačný systém, ktorý je plne konfigurovateľný a ktorý taktiež dokáže graficky vyobraziť priebeh simulácie.

Efektivitu stratégií som vyhodnotil na základe niekoľkých rôznych simulácií, pričom niektoré z nich vznikli z dát reálneho BSS. Výsledky, ktoré som dosiahol naznačujú, že navrhnutý odmeňovací mechanizmus nie je príliš efektívny. V sekcii 5.2.4 som opísal dôvody, ktoré spomínanú neefektivitu spôsobujú a taktiež možné vylepšenia tohoto mechanizmu. Hlavné a čiastočné ciele práce, analyzovať, implementovať a vyhodnotiť možnosti vyvažovania BSS sa teda podarilo naplniť.

Túto prácu by bolo možné rozšíriť niekoľkými spôsobmi, ktoré považujem za zaujímavé. Ako prvý z nich je vytvoriť robustnejší simulačný systém, ktorý by poskytoval webové rozhranie, pričom výsledky simulácie by boli ukladané do databázy. Takto by mohli operátori BSS flexibilne vyhodnocovať efektivitu odmeňovacích stratégií.

Ďalšie možné rozšírenie je implementovať odmeňovací mechanizmus, ktorý by ponúkal odmenu za vypožičanie bicykla z inej než pôvodne zóny. To sa zíde v prípade, kedy je zóna, z ktorej si užívateľ plánuje vypožičiať bicykel priveľmi vyťažená. Čiže odmeňovacie stratégie by ponúkali odmenu pri parkovaní a vypožičiavaní bicykla.

Taktiež by bolo možné vytvoriť lepší zhlukovací algoritmus, ktorý mesto rozdeľuje do zón a prediktívny model, ktorý by dokázal kvalitne predpovedať budúci stav zón.

Za myšlienku taktiež stojí spojiť vyvažovanie pomocou flotily s odmeňovaním užívateľov. Ťažko prístupné zóny pre bežného užívateľa, napr. na strmom

kopci by vyvažovali dodávky a ľahko prístupné, atraktívne zóny by boli vyvažované pomocou odmien pre užívateľov.

Pri tvorbe tejto práce som využil poznatky z celého bakalárskeho štúdia, predovšetkým z predmetov zameriavajúcich sa na oblasť softwarového inžinierstva, objektového modelovania, pravdepodobnosti a štatistiky.

Medzi prínosy tejto práce teda patrí analýza existujúcich riešení, vyhodnotenie efektivity navrhnutých odmeňovacích stratégií a vytvorený simulačný systém, ktorý je plne konfigurovateľný.

Literatúra

[1] Pal, A.; Zhang, Y.: Free-floating bike sharing. *Transportation Research Part C: Emerging Technologies*, ročník 80, 2017: s. 92–116, ISSN 0968090X, doi:10.1016/j.trc.2017.03.016. Dostupné z: `https://linkinghub.elsevier.com/retrieve/pii/S0968090X17300992`

[2] Singla, A.; Santoni, M.; Bartók, G.; aj.: Incentivizing users for balancing bike sharing systems. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[3] O'brien, O.; Cheshire, J.; Batty, M.: Mining bicycle sharing data for generating insights into sustainable transport systems. *Journal of Transport Geography*, ročník 34, 2014: s. 262–273.

[4] Pfrommer, J.; Warrington, J.; Schildbach, G.; aj.: Dynamic Vehicle Redistribution and Online Price Incentives in Shared Mobility Systems. *IEEE Transactions on Intelligent Transportation Systems*, ročník 15, č. 4, 2014: s. 1567–1578, ISSN 1524-9050, doi:10.1109/TITS.2014.2303986. Dostupné z: `http://ieeexplore.ieee.org/document/6754195/`

[5] Erdoğan, G.; Battarra, M.; Calvo, R. W.: An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. *European Journal of Operational Research*, ročník 245, č. 3, 2015: s. 667–679, ISSN 03772217, doi:10.1016/j.ejor.2015.03.043. Dostupné z: `https://linkinghub.elsevier.com/retrieve/pii/S0377221715002659`

[6] Laporte, G.; Nobert, Y.: A Cutting Planes Algorithm for the m -Salesmen Problem. *Journal of the Operational Research Society*, ročník 31, č. 11, 2017-12-20: s. 1017–1023, ISSN 0160-5682, doi:10.1057/jors.1980.188. Dostupné z: `https://www.tandfonline.com/doi/full/10.1057/jors.1980.188`

[7] Chemla, D.; Meunier, F.; Calvo, R. W.: Bike sharing systems. *Discrete Optimization*, ročník 10, č. 2, 2013: s. 120–146, ISSN

15725286, doi:10.1016/j.disopt.2012.11.005. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S1572528612000771

[8] Savelsbergh, M. W. P.; Sol, M.: The General Pickup and Delivery Problem. *Transportation Science*, ročník 29, č. 1, 1995: s. 17–29, ISSN 0041-1655, doi:10.1287/trsc.29.1.17. Dostupné z: http://pubsonline.informs.org/doi/abs/10.1287/trsc.29.1.17

[9] Cruz, F.; Subramanian, A.; Bruck, B. P.; aj.: A heuristic algorithm for a single vehicle static bike sharing rebalancing problem. *Computers & Operations Research*, ročník 79, 2017: s. 19–33, ISSN 03050548, doi:10.1016/j.cor.2016.09.025. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0305054816302489

[10] Wilson, S. W.; aj.: Explore/exploit strategies in autonomy. In *Proc. of the Fourth International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, ročník 4, 1996, s. 325–332.

[11] Vermorel, J.; Mohri, M.: Multi-armed Bandit Algorithms and Empirical Evaluation. In *Machine Learning: ECML 2005*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ISBN 978-3-540-29243-2, s. 437–448, doi:10.1007/11564096_42. Dostupné z: http://link.springer.com/10.1007/11564096_42

[12] Fricker, C.; Gast, N.: Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity. *EURO Journal on Transportation and Logistics*, ročník 5, č. 3, 2016: s. 261–291, ISSN 2192-4376, doi:10.1007/s13676-014-0053-5. Dostupné z: http://link.springer.com/10.1007/s13676-014-0053-5

[13] Pan, L.; Cai, Q.; Fang, Z.; aj.: A Deep Reinforcement Learning Framework for Rebalancing Dockless Bike Sharing Systems. 2018, 1802.04592.

[14] White, C. C.: Markov decision processes. In *Encyclopedia of Operations Research and Management Science*, Dordrecht: Kluwer Academic Publishers, 2001, ISBN 0-7923-7827-X, s. 484–486, doi:10.1007/1-4020-0611-X_580. Dostupné z: http://www.springerlink.com/index/10.1007/1-4020-0611-X_580

[15] Caggiani, L.; Camporeale, R.; Ottomanelli, M.; aj.: A modeling framework for the dynamic management of free-floating bike-sharing systems. *Transportation Research Part C: Emerging Technologies*, ročník 87, 2018: s. 159–182, ISSN 0968090X, doi:10.1016/j.trc.2018.01.001. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0968090X18300020

[16] Chernev, A.; Böckenholt, U.; Goodman, J.: Choice overload: A conceptual review and meta-analysis. *Journal of Consumer Psychology*, ročník 25, č. 2, 2015: s. 333–358.

[17] Dell'Amico, M.; Hadjicostantinou, E.; Iori, M.; aj.: The bike sharing rebalancing problem. *Omega*, ročník 45, 2014: s. 7–19, ISSN 03050483, doi:10.1016/j.omega.2013.12.001. Dostupné z: `https://linkinghub.elsevier.com/retrieve/pii/S0305048313001187`

[18] JavaFX Overview. [Online], [cit. 2019-05-4]. Dostupné z: `https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm`

[19] Gradle User Manual. [Online], [cit. 2019-05-4]. Dostupné z: `https://docs.gradle.org/current/userguide/userguide.html`

[20] Log4j User Manual. [Online], [cit. 2019-05-4]. Dostupné z: `https://logging.apache.org/log4j/2.x/manual/index.html`

[21] Commons math library documentation. [Online], [cit. 2019-05-4]. Dostupné z: `https://commons.apache.org/proper/commons-math/`

[22] Jackson library overview. [Online], [cit. 2019-05-4]. Dostupné z: `https://github.com/FasterXML/jackson`

[23] Wagnild, J.; Wall-Scheffler, C. M.; Konigsberg, L.: Energetic Consequences of Human Sociality. *PLoS ONE*, ročník 8, č. 10, 2013-10-23, ISSN 1932-6203, doi:10.1371/journal.pone.0076576. Dostupné z: `https://dx.plos.org/10.1371/journal.pone.0076576`

Dodatok **A**

Zoznam použitých skratiek

BSS Systém zdieľaných bicyklov

SBSS Stanicový systém zdieľaných bicyklov

FBSS Voľne plavúci systém zdieľaných bicyklov

MILP Zmiešané celočíselné lineárne programovanie

SCRP Statické úplne vyvažovanie pomocou flotily vozidiel

m-TSP Viacnásobný problém obchodného cestujúceho

JSON JavaScriptový objektový zápis

CSV Formát súboru, kde sú hodnoty oddelené čiarkami

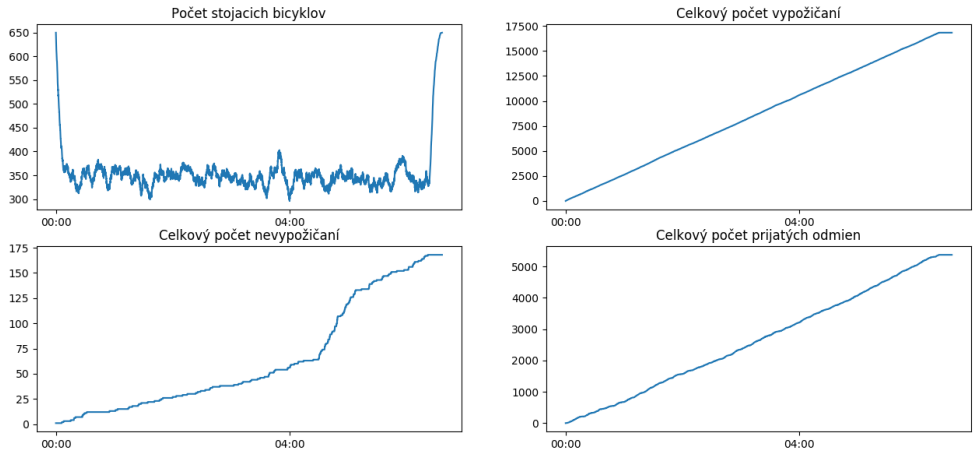
JVM Java virtuálny stroj

NYC New York City

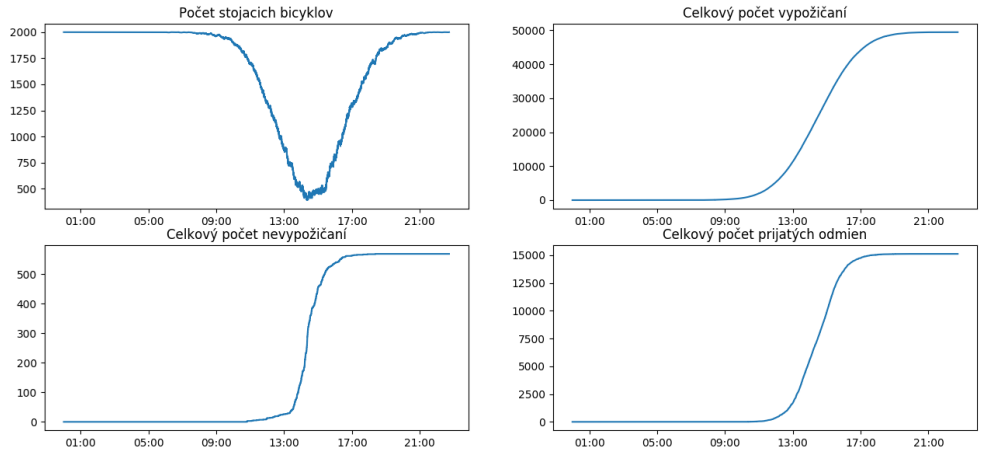
AR Počet udalostí kedy užívateľ kvôli odmene zaparkoval bicykel do inej, než jeho pôvodne cieľovej zóny

Dodatok B

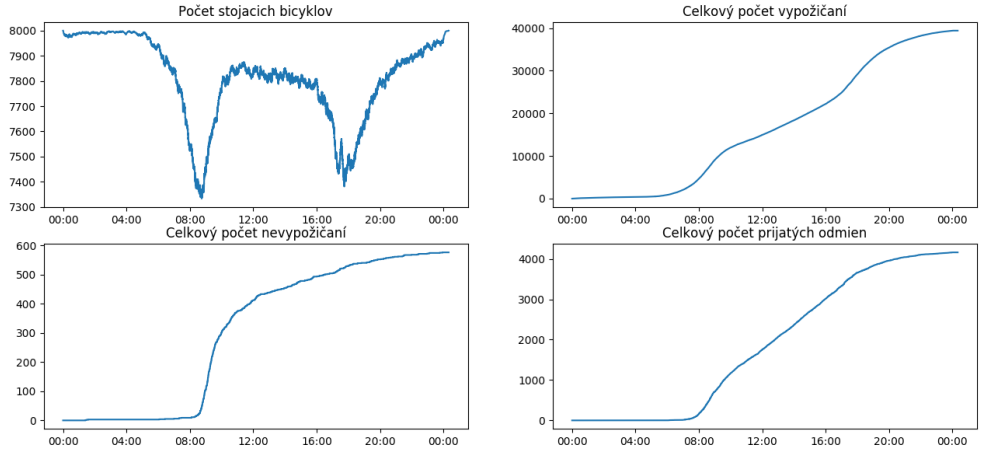
Diagramy



Obr. B.1: Grafy znázorňujúce priebeh simulácie s uniformnou konfiguráciou 5.1.1



Obr. B.2: Grafy znázorňujúce priebeh simulácie s normálnou konfiguráciou 5.1.2



Obr. B.3: Grafy znázorňujúce priebeh simulácie s konfiguráciou z historických jázd 5.1.3

Dodatok C

Obsah priloženého USB

- `readme.txt` ... stručný popis obsahu USB a návod ku spusteniu aplikácie
- `bin` ..... adresár so spustiteľnou formou implementácie a skriptom, ktorý vytvára grafy zo simulačnej správy
- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `thesis.pdf` ..... text práce vo formáte PDF
- `data`
  - `configuration` ..... príklady konfigurácií v json formáte
  - `schema` ..... json schéma konfigurácií a simulačnej správy
  - `dataset` ..... historické jazdy citibike NYC
  - `results` ..... výsledky simulácií v json formáte, ktoré som použil na experimentálne vyhodnotenie