ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA EKONOMICKÁ

Bakalářská práce

Návrh na zlepšení logistických činností v podniku zaměřeném na poskytování internetových služeb

Proposal for improving logistical operations inside business focused on providing internet service

Ondřej Šíp

Plzeň 2023

Čestné prohlášení

Prohlašuji, že jsem bakalářskou práci na téma

„Návrh na zlepšení logistických činností v podniku zaměřeném na poskytování internetových služeb“

vypracoval samostatně pod odborným dohledem vedoucího bakalářské práce za použití pramenů uvedených v přiložené bibliografii.

Plzeň dne 24. 4. 2023 v. r. *Ondřej Šíp*

Zásady pro vypracování práce

1. Charakterizujte logistické problémy související s poskytováním služeb
2. Charakterizujte vybraný podnikatelský subjekt
3. Analyzujte sledované procesy a navrhněte jejich zlepšení
4. Očekávané přínosy doložte výpočty na reálných datech
5. Navrhněte implementaci navrženého řešení ve vybraném podniku

Poděkování

Děkuji prof. Dr. Ing. Miroslavu Plevnému za pomoc při vedení mé bakalářské práce. Mé poděkováni patří také mojí rodině, přítelkyni a přátelům, kteří mě podporovali během celého studia.

Obsah

Seznam použitých zkratek

TSP Travelling Salesman Problem (problém obchodního cestujícího)

API Application Programming Interface

JVM Java Virtual Machine

Úvod

Tato práce si klade za cíl pomoci vybranému ekonomickému subjektu s plánováním optimální trasy technika v síti zajišťující internetové připojení. Po přiblížení jednotlivých stěžejních teorií a technologií v teoretické části práce autor ekonomický subjekt představí, rozebere některé ekonomické problémy, které daný ekonomický subjekt řeší, navrhne řešení konkrétního problému pomocí optimalizačního algoritmu a následnou implementaci v ekonomickém subjektu. Tyto problémy bude autor řešit pomocí programu, který si sám napíše a na konci práce autor popíše jím navrhované řešení a rozebere jeho teoretické přínosy pro subjekt. Budou zmíněna i jiná možná řešení, která by mohla být zvážena.

Cíle práce

Hlavním cílem této práce je identifikace zdroje nákladů, které ekonomickému subjektu vznikají v souvislosti s jeho působením v oblasti poskytování internetového připojení a které by se daly minimalizovat použitím optimalizačních algoritmů. Bude následovat návrh řešení tohoto problému pomocí sestaveného programu, výpočet teoretického ekonomického přínosu spojeného s odstraněním těchto nákladů a následný návrh spolku na implementaci autorova řešení v praxi.

1 Teoretický úvod do problematiky

Každý ekonomický subjekt, kterému vznikají náklady spojené s pohybem zaměstnanců po pozemních komunikacích by se měl snažit tyto náklady alespoň z části eliminovat. To se provádí použitím algoritmu zaměřeného na zlepšení efektivity pohybu zaměstnanců po pozemních komunikacích. V této kapitole budou představeny teoretické základy a pojmy, které poté budou použity pro sestavení takového algoritmu.

1.1 Teorie grafů

Teorie grafů je obor diskrétní matematiky zaměřující se na řešení úloh, kde jsou dána určitá místa a spojení mezi nimi. Místa se v této teorii nazývají vrcholy a spojení hrany. U hran někdy záleží na směru a někdy nikoli. Tuto situaci si můžeme představit jako situaci, kde vrcholy grafu jsou křižovatky a hrany jsou jednotlivé ulice. Pokud je tato hrana spojena ulicí s chodníkem, pak chodci nezáleží na tom, jakým směrem jsou tyto vrcholy spojené, ale že jsou. Na druhou stranu pro řidiče na směru záleží. Tato hrana může být tvořena například jednosměrnou ulicí. (Bělohlávek, Vychodil. 2006)

U grafů, kde jsou vrcholy tvořeny městy a hrany jsou silnicemi mezi jednotlivými městy, se v praxi často řeší úlohy spojené s hledáním určité cesty skrz tento graf. Může to být nejkratší cesta v grafu, problém obchodního cestujícího, nebo problém čínského listonoše. Tyto problémy budou přiblíženy níže.

Symetrický graf je takový, kde nezáleží na směru pohybu mezi vrcholy. To znamená, že vzdálenost z vrcholu A do vrcholu B je stejná jako z vrcholu B do vrcholu A.

Asymetrický graf je takový graf, kde se vzdálenost z A do B neshoduje se vzdáleností z B do A.

Hamiltonovská kružnice je graf, který spojuje všechny vrcholy grafu uzavřenou cestou a tím tvoří kružnici. Cesta, která je otevřená a vede všemi vrcholy grafu je Hamiltonovská cesta. Využívány jsou také minimální a maximální Hamiltonovské kružnice. Takové kružnice jsou, pokud obsahují minimální nebo maximální součet délek hran v grafu. (Krátký, 2020)

Optimální řešení je takové, které je nejlepším možným řešením.

Suboptimální řešení je přípustné řešení, které ovšem není optimální. Například pro hledání Hamiltonovské kružnice je minimální, případně maximální Hamiltonovská

kružnice optimálním řešením a všechny ostatní Hamiltonovské kružnice jsou suboptimálním řešením.

1.1.1 Nejkratší cesta

Nejkratší cesta grafem se často řeší pomocí Dijkstrova[1] algoritmu. Tento algoritmus se používá u grafů s nezáporně ohodnocenými hranami. Algoritmus funguje tak, že každému vrcholu nastaví na počátku vysoké číslo a postupně toto číslo vylepšuje.

Algoritmus funguje v těchto krocích:

1. Ke všem vrcholům se přiřadí počáteční vzdálenost. Tato vzdálenost je pro počáteční vrchol 0 a pro všechny ostatní vysoké předem nastavené číslo, které je možné pouze vylepšit.
2. Všechny vrcholy jsou zařazeny do množiny nenavštívených vrcholů a počáteční vrchol je nastavený jako aktuální vrchol.
3. Algoritmus se „podívá“ na své sousedy a vypočte pro ně vzdálenost. Tuto vzdálenost vypočte jako součet vzdálenosti do současného vrcholu a délku hrany k sousednímu vrcholu. Pokud soused již má vzdálenost přiřazenou, ale algoritmus vypočetl celkovou vzdálenost lepší, nahradí hodnotu stávající za nově vypočtenou.
4. Jakmile jsou prohlédnuti všichni sousedé současného vrcholu, tento vrchol přesuneme do množiny navštívených a již se k němu nevracíme.
5. Pokud byl minulý vrchol označen jako cílový nebo je množina nenavštívených vrcholů prázdná, algoritmus končí.
6. Z množiny nenavštívených vrcholů je vybrán další vrchol, který má celkovou nejmenší vzdálenost od počátku a program se opakuje od bodu 3.

Algoritmus tedy prohledává graf do šířky a postupně zlepšuje celkovou vzdálenost k jednotlivým vrcholům. Výstup je nejkratší cesta z počátku ke všem ostatním vrcholům. (Dijkstrův algoritmus, n.d.)

1.1.2 Problém obchodního cestujícího

Problém obchodního cestujícího (TSP) je často řešeným problémem kombinatorické optimalizace. Cílem je nalézt nejkratší délku trasy mezi několika vrcholy tak, aby každý

[1] Edsger Wybe Dijkstra (1930–2002)

vrchol byl navštíven právě jednou a trasa končila ve výchozím vrcholu. Tento problém má mnoho praktických aplikací, například v logistice, cestovním ruchu, řešení problémů v distribuci, nebo plánování tras pro robotické systémy.

Tento problém může být formulován jako optimalizační problém, kde je úkolem minimalizovat celkovou délku cesty. V praxi se tento problém řeší ve dvou variantách. Ve variantě s asymetrickým grafem, kde je délka hrany mezi dvěma vrcholy závislá na směru, nebo v symetrické variantě, kde na směru nezáleží.

Problém obchodního cestujícího je znám jako NP-úplný problém, což znamená, že pro velké instance problému je téměř nemožné nalézt optimální řešení v rozumném čase. Proto se v praxi často používají heuristické algoritmy a aproximativní metody pro nalezení suboptimálních řešení.

Mezi nejznámější algoritmy pro řešení TSP patří:

1. Algoritmus hrubé síly (Brute Force algoritmus): Tento algoritmus vypočte všechny možné cesty a najde tu nejkratší. Tento přístup je ovšem velmi časově náročný a použitelný pouze pro malé instance problému.
2. Algoritmus nejbližšího souseda: Tento algoritmus začíná v předem určeném výchozím vrcholu a postupně vybírá nejbližší vrchol, který ještě nenavštívil, a pokračuje tímto způsobem, dokud neprojde všemi vrcholy. Tento algoritmus však nemusí vždy vést k nejlepšímu řešení.
3. Christofidesův algoritmus: Tento algoritmus nalézá nejprve minimální kostru grafu a poté ji transformuje na Hamiltonovskou kružnici. Je schopen nalézt řešení, které je nejvýše o 50 % horší než optimální řešení.
4. Genetické algoritmy: Tyto algoritmy využívají principy evolučních procesů a jsou schopny nalézt suboptimální řešení v relativně krátkém čase. Základem je tvorba jedinců, kteří se dají ohodnotit funkcí fitness. Čím lepší má jedinec tuto hodnotu, tím větší pravděpodobnost má na reprodukci.
5. Lin-Kernighan heuristika: Tento algoritmus využívá úpravy existujících řešení a iterativně je vylepšuje. Tento algoritmus je řazen mezi nejvýkonnější heuristické algoritmy pro řešení TSP. (Talska, 2019)

Existuje mnoho dalších algoritmů a metod pro řešení TSP, včetně hybridních metod a metod založených na umělé inteligenci. Výběr konkrétního algoritmu závisí na konkrétních požadavcích a parametrech problému.

Je třeba zdůraznit, že TSP je velmi složitý problém a většinou není možné nalézt exaktní řešení v rozumném čase pro velké instance problému.

1.1.2.1 Algoritmus hrubé síly

Tento algoritmus zahrnuje kontrolu všech možných permutací vrcholů k nalezení optimálního řešení. Časová složitost tohoto algoritmu znamená, že se stává nepraktickým pro velké instance problému. Navzdory jeho neefektivnosti pro velké objemy dat, je přístup hrubé síly užitečný pro malé instance TSP a pro testování přesnosti ostatních algoritmů.

Algoritmus hrubé síly je jednoduchý, ale výpočetně náročný algoritmus pro řešení TSP. Principem algoritmu je prohledávání všech možných permutací vrcholů a vyhodnocení délky každé trasy. Tento postup se opakuje pro všechny možné permutace, dokud není nalezena nejkratší trasa. Je garantováno, že tento algoritmus vždy nalezne optimální řešení, ale vzhledem k vysokému počtu možných permutací je algoritmus výpočetně náročný pro velký počet vrcholů.

I když algoritmus hrubé síly má vysokou výpočetní náročnost, stále se používá pro řešení malých instancí TSP, kde je možné nalézt optimální řešení. Význam algoritmu hrubé síly není v jeho efektivitě, ale v tom, že slouží jako kontrolní bod pro vývoj pokročilých metod řešení TSP, protože můžeme říci, že nalezení řešení tímto algoritmem je opravdu optimální.

1.1.2.2 Branch and Bound

Branch and Bound algoritmus (algoritmus větví a hranic) pracuje na základě principu omezování horních a dolních mezí a využívá strategie větví a hranic k omezení počtu možných permutací, které je třeba vyhodnotit. Tento postup umožňuje algoritmu snížit počet permutací, které je třeba vyhodnotit, což vede ke zrychlení výpočtu. (Talak, 2019)

1.1.2.3 Algoritmus nejbližšího souseda

Tento algoritmus začíná od zvoleného vrcholu a vybírá nejbližší neprohlížený vrchol jako další vrchol k návštěvě. Tento proces se opakuje, dokud nejsou navštívena všechny vrcholy. Algoritmus nejbližšího souseda je jednoduchý a časově efektivní, ale je náchylný k produkci suboptimálních řešení.

Metoda nejbližšího souseda (anglicky Nearest Neighbor Approach) je jednou z nejstarších a nejjednodušších metod pro řešení TSP. Metoda začíná v počátečním vrcholu

a postupně přechází do nejbližšího neobsazeného vrcholu až do té doby, než jsou navštíveny všechny vrcholy. Tato metoda je snadno implementovatelná a rychlá, ale může vést k podprůměrným výsledkům. Počáteční vrchol může být zvolen i náhodně.

Metoda nejbližšího souseda funguje následujícím způsobem:

1. Začneme ve vybraném vrcholu a zvolíme nejbližší neobsazený vrchol jako další vrchol, který navštívíme.
2. Přejdeme na námi vybraný vrchol a vzdálenost mezi minulým a momentálním vrcholem přičteme k celkové délce.
3. Tento postup opakujeme, dokud nejsou navštíveny všechny vrcholy.

Výsledkem této metody je hamiltonovská kružnice, která projde všechny vrcholy. Pak se kružnice vrátí zpět do výchozího vrcholu.

Metoda nejbližšího souseda má několik výhod i nevýhod. Mezi hlavní výhody patří její jednoduchost a rychlost. Metoda je také poměrně robustní vůči chybám v datech a může být použita pro řešení mnoha různých variant TSP. Nicméně metoda nejbližšího souseda trpí několika vážnými nevýhodami, které omezují její použití v mnoha aplikacích. (Talak, 2019)

Pokud je zvolen nesprávný výchozí vrchol může metoda nejbližšího souseda vést k výsledkům, které se velmi liší od optimálního řešení. Další nevýhodou této metody je, že nebere v úvahu celkovou délku trasy, ale pouze nejbližší vrcholy. Tím může vzniknout situace, kdy je trasa delší, než by mohla být, protože se vybírají blízké vrcholy namísto vzdálenějších vrcholů, které by byly lepší volbou a celkově by zlepšily trasu, jak to dělají ostatní algoritmy pro řešení TSP.

Přestože metoda nejbližšího souseda není vždy nejlepším řešením pro TSP, má stále své využití. Může být užitečná v případech, kdy je potřeba rychle nalézt přibližné řešení. V mnoha případech může být také použita jako součást kombinovaných algoritmů pro zlepšení celkového výsledku.

1.1.2.4 Genetický algoritmus

Tento algoritmus je inspirován procesem přirozeného výběru a evoluce. Zahrnuje generování populace prvotních řešení a iterativní aplikaci genetických operátorů, jako jsou výběr, křížení a mutace, k vytváření nových generací řešení. Genetický algoritmus může produkovat dobré výsledky pro TSP, zejména pro instance s velkým počtem

vrcholů. Proces je založen na principu přirozeného výběru, kde ti nejlepší potomci jsou vybráni k reprodukci a jejich vlastnosti jsou předány další generaci.

Genetický algoritmus pro řešení problému obchodního cestujícího obsahuje následující kroky:

1. Inicializace – Prvním krokem je vytvoření počáteční populace potenciálních řešení. To lze udělat náhodně nebo pomocí heuristického algoritmu, jako je algoritmus nejbližšího souseda.
2. Hodnocení – Každý jedinec je hodnocen na základě cílové funkce.
3. Výběr – Dalším krokem je výběr nejlepších jedinců z populace.
4. Křížení – Vybraní jedinci jsou pak zkříženi, aby vznikli potomci. To se dělá výběrem dvou rodičů a kombinováním jejich genetického materiálu pro vytvoření nového jedince.
5. Mutace – V kroku mutace je náhodně změněn některý z genetického materiálu potomků, aby byly zavedeny nové vlastnosti. Toto slouží k uchování genetické rozmanitosti populace.
6. Opakování – Kroky 2–5 se opakují, dokud není splněna ukončovací podmínka.

Genetický algoritmus má svoje výhody i nevýhody. Mezi výhody patří jeho schopnost neuvíznout v lokálním maximu a jsou využívány pro problémy, které mají velkou množinu přípustných řešení. Zároveň algoritmus má tendenci se v průběhu času blížit skutečně optimálnímu řešení. Ovšem implementace algoritmu není vždy jednoduchá a genetické algoritmy mají problém najít skutečně optimální řešení. (Petr Luner, n.d.)

Variací genetických algoritmů je například memetický algoritmus (Memetic Algorithm). Tento algoritmus kombinuje genetický algoritmus s technikami pro hledání lokálního minima za účelem zlepšení řešení. Techniky lokálního vyhledání jsou používány pro zlepšení řešení generovaných genetickým algoritmem. (Moscato, Cotta, Mendez. 2004)

1.1.2.5 Mravenčí kolonie

Algoritmus optimalizace mravenčí kolonií (Ant Colony Optimalization) je založen na mravencích hledajících trasu mezi mraveništěm a zdrojem potravy. Mravenci za sebou zanechávají feromonovou stopu, kterou mohou poté následovat ostatní mravenci z kolonie. Pokud ovšem není chemická stopa obnovována používáním postupně vymizí. Pro problém obchodního cestujícího se dá optimalizace mravenčí kolonií využívat následujícím způsobem. Máme ze začátku určitý počet mravenců, náhodně tyto mravence

umístíme ve vrcholech grafu a dovolíme jim pohybovat se libovolně mezi nimi. Mravenec ovšem mravenec nesmí opětovně navštívit vrchol, ve kterém se již jednou nacházel (pouze s výjimkou toho, že se jedná o vracení do původního vrcholu a uzavření cesty). Mravenec, který prošel nejkratší cestou, po sobě zanechává feromonovou stopu nepřímo úměrnou délce jeho cesty. Čím silnější stopu feromonů má daná cesta, tím spíše ji budou ostatní mravenci preferovat. Tento proces se bude opakovat, dokud nebude nalezena nejkratší cesta. (Matai, Surya, Murari. 2010)

1.1.2.6 Simulované žíhání

Algoritmus je založen na procesu žíhání v metalurgii. Zahrnuje iterativní modifikaci aktuálního řešení náhodnou výměnou dvou vrcholů a přijetím nového řešení, pokud zlepší celkovou funkci nebo s určitou pravděpodobností dokonce i tehdy, pokud celkovou funkci zhorší. Pravděpodobnost přijetí horších řešení se snižuje s časem, což umožňuje algoritmu uniknout z lokálních optim a prozkoumat prostor pro hledání více extenzivně. Simulované žíhání může produkovat dobré výsledky pro TSP, zejména pro instance s rozmanitým prohledávacím prostorem. (Yu, Susanto, Jodiawan, Ho, Lin, Huang. 2022)

Proces je inspirován žíháním při obrábění kovů. Tento proces pomalého ochlazování u kovů pomáhá odstranit defekty a zvýšit sílu kovu. Obdobně při simulovaném žíhání algoritmus začíná pracovat s vysokou teplotou, která se postupně snižuje v čase. Tento proces umožňuje algoritmu přeskočit lokální maxima a dosáhnout maxima globálního.

Algoritmus simulovaného žíhání pro řešení problému obchodního cestujícího zahrnuje tyto kroky:

1. Inicializace – První krok je zvolení počátečního řešení a teploty. Tento krok může být proveden buď náhodně nebo za použití nějakého jiného algoritmu jako například algoritmu nejbližšího souseda.
2. Zvolení souseda – V tomto kroku si algoritmus generuje souseda. Tento soused je vygenerován provedením malé změny u již známého výsledku. Tento krok může být proveden využitím různých algoritmů jako například 2-opt, 3-opt, nebo k-opt přístupu.
3. Kritérium přijetí – Nové řešení je ohodnoceno za použití přijímacího kritéria. Za předpokladu, že nové řešení je lepší než řešení předchozí, je přijato jako nové momentální řešení. Pokud je však řešení horší než momentální, může být stále

přijato s určitou pravděpodobností, která je dána teplotou prostředí a rozdílem, který tato dvě řešení dělí. Tato pravděpodobnost klesá s teplotou.

4. Plán chlazení – Teplota je potom snížena podle plánu chlazení. Plán chlazení určuje, jak rychle klesá teplota v prostředí. Z počátku teplota klesá teplota zpravidla pomaleji a postupně se pokles zrychluje.
5. Opakují se kroky 2–4, dokud není nalezeno řešení, které splňuje ukončovací podmínku. (Yu, Susanto, Jodiawan, Ho, Lin, Huang. 2022)

Jednou z hlavních výhod použití simulovaného žíhání je schopnost algoritmu uniknout lokálním minimům a blížit se skutečně globálnímu minimu. To je proto, že algoritmus má šanci v počáteční fázi algoritmu při vysoké teplotě prostředí akceptovat i horší výsledek. Toto se však mění s klesající teplotou a potupně se algoritmus stává selektivnějším a přijímá pouze výsledky, které jsou lepší. Simulované žíhání může být použito pro hledání více dobrých výsledků rozběhnutím několika simulací s rozdílnými vstupními podmínkami.

Výkonnost simulovaného žíhání však závisí na zvolení metody pro generování sousedů a zvolení plánu žíhání. Dobře zvolená metoda pro generování sousedů by měla být schopna generovat rozmanité výsledky, které jsou blízké originálnímu řešení. Dobře zvolený plán žíhání by měl být schopný skloubit prozkoumávání okolí a prozkoumávání výsledků do hloubky.

1.1.3 Problém čínského listonoše

Tento problém je založen na tom, že listonoš musí každou ulici navštívit právě jednou, aby roznesl poštu. Oproti problému nejkratší cesty grafem, u kterého se řeší pouze nejkratší cesta mezi dvěma body, a problému obchodního cestujícího, kde se řeší, aby byl každý vrchol navštíven právě jednou, se u problému čínského listonoše zabýváme tím, že každá hrana grafu musí být navštívena právě jednou. To si lze představit jako jednoduchou úlohu kreslení domečku jedním tahem. Při kreslení domečku jedním tahem nám také záleží na tom, abychom po každé hraně pomyslného domečku jeli jen jednou. (Kaliaganov, 2021)

1.2 Technologie spojené s poskytováním internetových služeb

Všechny ekonomické subjekty, které fungují v oblasti poskytování internetových služeb svým členům, používají stejnou technologii. Tato technologie se dá dělit na dvě skupiny, a to bezdrátové technologie a drátové technologie.

Mezi bezdrátové technologie patří jakékoli využití technologie, která k přenosu dat mezi dvěma vysílači nepoužívá fyzické spojení, ale vlnění o různých vlnových délkách. Tato technologie se používá převážně v oblastech, které byly vystavěny před tím, než byl vynalezen internet, a z toho důvodu k těmto budovám není založeno vedení internetu do země.

Naopak drátové technologie se využívají převážně v oblastech, ve kterých se s budoucím internetovým připojením počítalo, a proto jsou do země založeny internetové kabely a nebo je oblast na toto zavedení připravena.

1.2.1 Bezdrátové komunikace

Bezdrátová komunikace je komunikace dvou subjektů, které nejsou spojeny fyzicky kabelem, ale bezdrátovou technologií. Pro tato spojení se využívá technologie pracující na mnoha různých frekvencích.

Nízká komunikační pásma jsou kvůli jejich historické zátěži, technickému vývoji a fyzikálním zákonům poměrně zaplněna. Regulační orgány proto závádí pravidla pro provoz bezdrátových spojů. Regulátoři musí zabezpečit nerušený provoz nejenom pro telekomunikační účely, ale i pro účely vojenských a civilních radarů, provozu družic a řízení dopravy. Ve vyšších pásmech je naopak volněji a podléhají méně regulacím. (Bezdrátová komunikace v infrastruktuře internetových sítí, n.d.)

Výhody využití bezdrátových technologií jsou následující:

- Nízké počáteční náklady,
- Možnost připojení jakéhokoli typu objektu,
- Možnost překonání téměř jakéhokoli terénu.

Naopak mezi nevýhody bezdrátových spojení patří:

- Vysoká citlivost na vnější podmínky,
- Citlivost na rušení způsobené anténami na stejné, nebo podobné frekvenci.

1.2.2 Drátové komunikace

Drátová komunikace je přenos informací pomocí drátu. Z historických důvodů se stále ve většině případů používají metalické kabely, ale s moderní technologií a dobou se přechází stále víc a víc na technologii optických kabelů. Kabelová komunikace je obecně považována za nejstabilnější ze všech typů komunikačních služeb. Ve srovnání s bezdrátovými komunikačními řešeními jsou neovlivnitelné nepříznivým počasím a povětrnostními podmínkami. Tyto vlastnosti umožnily, aby kabelová komunikace zůstala populární, i když pokrok bezdrátových řešení pokračoval. (GeeksforGeeks, n.d.)

1.3 Software použitý pro řešení

1.3.1 Java

Programovací jazyk Java je objektově orientovaný jazyk, který byl navržen tak, aby byl nezávislý na platformě. To znamená, že Java může fungovat na jakémkoli systému. Jediné omezení je, že tento systém musí mít nainstalovaný JVM (Java Virtual Machine). Jazyk byl vytvořen Jamesem Goslingem a jeho týmem v Sun Microsystems v polovině devadesátých let a od té doby se stal jedním z nejpoužívanějších programovacích jazyků na světě. (Javatpoint, n.d.)

1.3.2 Application Programming Interface

API neboli Application Programming Interface je v informatice rozhraní, které umožnuje komunikaci dvou různých programů a vyměňování dat mezi těmito programy. Slouží k rozšíření funkcionality určitého programu bez nutnosti tyto nové funkce vymýšlet a implementovat. Tím API šetří programátorům jak čas, tak i peníze. Odpověď většinou přichází ve formátu JSON. (Co je to API, n.d.)

1.3.3 OpenRouteServis

OpenRouteService je webová stránka, která umožňuje plánování tras. Tato stránka zároveň poskytuje možnost komunikace přes rozhraní API, která se dá použít ke zjištění vzdálenosti dvou bodů. Tato stránka své služby nabízí do určité míry zadarmo. Omezení je nastaveno na 2000 dotazů na vzdálenost za den. (Services, n.d.)

Tato stránka je vyvíjená Heilderbergským institutem pro geoinformační technologie HeiGIT. Tento institut vyvíjí spolu se stránkou OpenRouteServis i další služby, které jsou

postaveny na těchto mapách. Mezi tyto služby patří mapa očkovacích center proti nemoci covid-19, mapa dostupnosti lékařské péče a další. (What we do, n.d.)

1.3.4 IntelliJ IDEA

IntelliJ IDEA je integrované vývojové prostředí (Integrated Development Environment, zkráceně „IDE“) pro programování v Javě. Toto programovací prostředí bylo vytvořeno společností JetBrains se sídlem v České republice a je široce uznáváno jako jedno z nejlepších a uživatelsky nejpřátelštějších IDE pro jazyk Java.

IntelliJ IDEA je výkonné a uživatelsky přátelské programovací prostředí, které umožňuje vývojářům snadno psát, ladit a testovat kód v Javě. Zahrnuje širokou škálu funkcí, včetně automatického doplňování kódu, nástrojů pro ladění a integrovaného testování jednotek. Obsahuje také řadu pluginů a rozšíření, které lze použít k dalšímu rozšíření jeho funkcionality.

Jednou z hlavních výhod použití IntelliJ IDEA je její snadné ovládání. Toto programovací prostředí je navrženo tak, aby bylo intuitivní a snadno se s ním pracovalo, což znamená, že se v něm dají najít všechny nástroje a funkce, které jsou potřebné pro úspěšný vývoj aplikací a programů. Navíc IntelliJ IDEA nabízí mnoho funkcí, které pomáhají vývojářům zlepšit produktivitu, jako jsou například rychlé nástroje pro refaktorizaci kódu a různé způsoby, jak analyzovat kód a najít chyby a nedostatky.

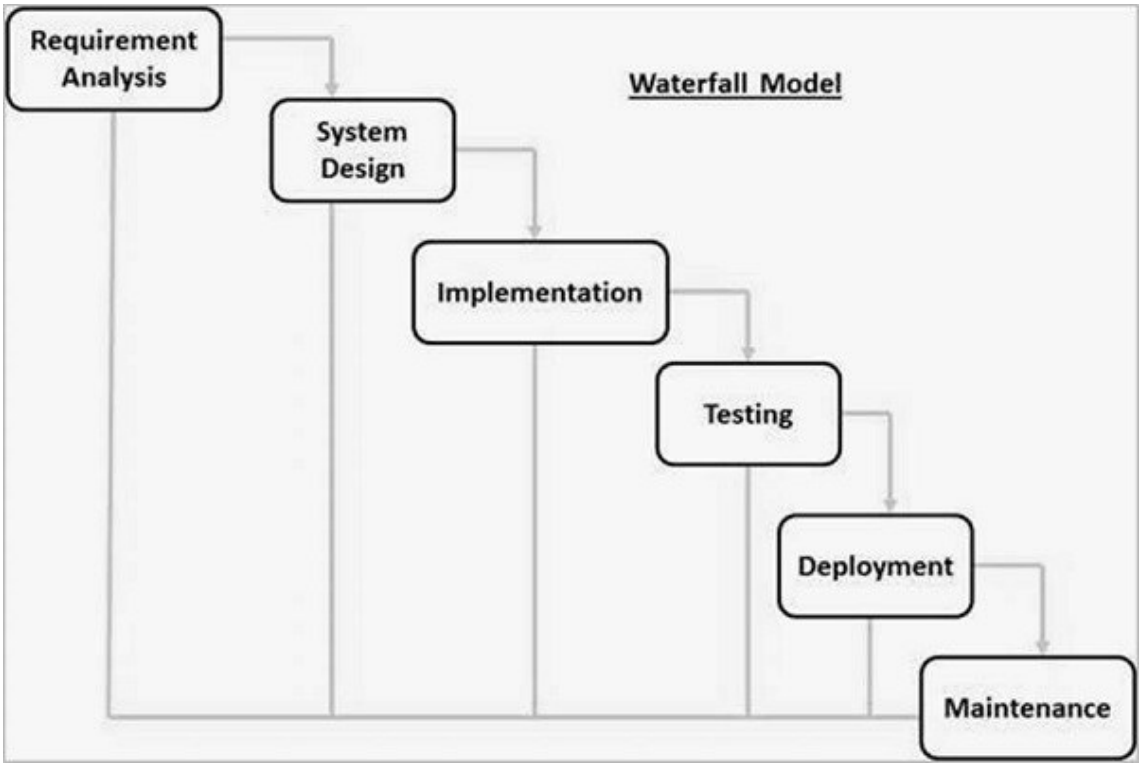
Další výhodou IntelliJ IDEA je její integrace s dalšími nástroji a technologiemi, které jsou běžně používány v Javě. Například podporuje systémy jako jsou Git a Subversion, nástroje pro správu závislostí jako Maven a Gradle, a také řadu frameworků pro vývoj webových aplikací jako Spring a Struts. (JetBrains, n.d.)

1.3.5 Implementace softwarového řešení

Implementace programového řešení pro výpočet problému obchodního cestujícího se vztahuje k procesu nasazení nového softwarového řešení v organizaci. Během implementace je nutné zajistit, aby byly splněny požadavky projektu a že nové řešení je bez chyb a v souladu s potřebami organizace. Implementace softwaru může být složitý proces a často vyžaduje úzkou spolupráci mezi vývojáři a zainteresovanými stranami v organizaci.

Obecně implementace jakéhokoli programového řešení technikou vodopádu ve společnosti obsahuje tyto jednotlivé fáze:

1. Analýza a plánování – V této fázi projektový tým provádí důkladnou analýzu stávajících firemních procesů a systémů s cílem určit požadavky na nové programové řešení. To může zahrnovat sběr vstupů od zainteresovaných stran a uživatelů, tvorbu map procesů a analýzu dat.
2. Návrh – V této fázi projektový tým navrhuje softwarové řešení, včetně uživatelského rozhraní, funkčnosti a architektury programu.
3. Vývoj – Tato fáze zahrnuje samotné psaní kódu a vývoj programového řešení. To může zahrnovat vytváření vlastních funkcí nebo integraci softwaru třetích stran.
4. Testování – Po dokončení vývoje softwarového řešení musí být důkladně otestováno, aby splňovalo požadavky a bylo bez chyb. To může zahrnovat jednotkové testování, integrační testování a testování reakcí, které toto řešení vyvolá u uživatelů a jestli uživatelé toto řešení přijmou.
5. Nasazení – Po úspěšném otestování a schválení může být softwarové řešení nasazeno do produkce. To může zahrnovat instalaci programu na servery, konfiguraci programu a migraci dat.
6. Údržba – Po nasazení programového řešení je nutná jeho pravidelná údržba a podpora, aby i nadále plnilo potřeby organizace. To může zahrnovat opravy chyb, aktualizace a vylepšení. (Waterfall Model, n.d.)



Obrázek 1) Schéma vyvíjení programu pomocí metody vodopádu. Zdroj: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

2 Charakteristika vybraného ekonomického subjektu

Spolek BOK*Net, z.s. je poskytovatelem telekomunikačních služeb pro oblast Plzeň a okolí. Spolek se specializuje na poskytování internetového připojení pro své členy. Od roku 2006 spolek poskytuje své služby členům spolku a zaměstnává sedm pracovníků. Ačkoli se jedná o malou spolek, zajišťuje internetové připojení pro zhruba 2600 členů spolku. Hlavním produktem spolku je poskytování připojení k internetu, ale spolek je schopen poskytnout i další služby, které se týkají internetového připojení.

*„Zakládající členové spolku BOK*Net působili původně jako správci jiného menšího spolku v Plzni. Líbila se jim myšlenka komunitního fungování spolku, který své členy připojuje bezpečnou sítí k Internetu. Jenže jak síť rostla, nadšení v původním spolku vyprchalo a hlavně se začaly objevovat tlaky na komercionalizaci. Jednotlivé stížnosti uživatelů nebyly řešeny, dokud nepředstavovaly ekonomický problém a o snaze vyjít vstříc ani nemluvě To se Ondrovi, Tomášovi a Jardovi nelíbilo, a tak se rozhodli v roce 2006 založit vlastní sdružení. Nejprve připojovali své známé, kamarády a sousedy, ale postupem let se síť spolku rozrostla až na současných téměř 2 300 spokojených členů.“* (O nás, n.d.)

Technici spolku mají pohyblivou pracovní dobu kvůli povaze výkonu práce a z důvodu toho, že se ne vždy dá technikova pracovní doba předpovědět a může nastat situace, kdy je technik potřeba v terénu neočekávaně. Každý zaměstnanec je placen od hodiny dle výkazu pracovních hodin a spolek hradí technikům náklady spojené s provozem vozidla, které vznikají správcům spolku, protože všichni používají pro výkon práce své soukromé vozidlo. Každý technik má vlastní auto a pracuje ze svého domova. Spolek nemá kancelář, pouze sklad materiálu, kam mají technici přístup bez omezení.

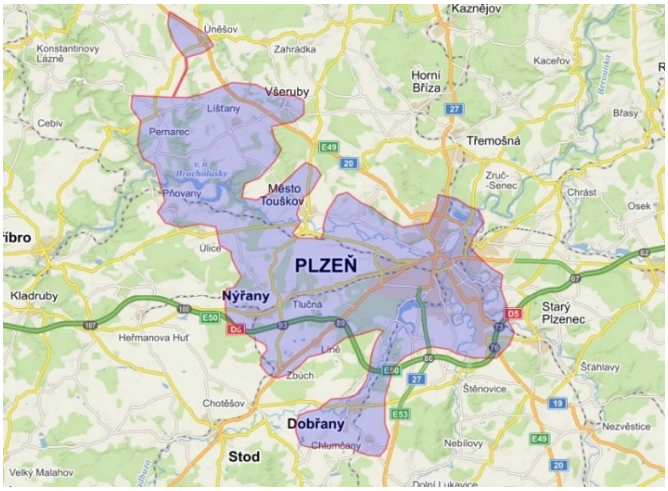
Spolek poskytuje svým členům internetové připojení domů nebo bytů do své vlastní sítě a všechny servisní činnosti s touto službou spojené. Spolek je však schopný pro svoje uživatele zajistit i jiné služby. Další služby, které je spolek schopen poskytnout, jsou:

- Vysokorychlostní připojení firem,
- realizace optických internetových sítí pro nově se rozvíjející stavební etapy,
- realizace kabeláže po bytu nebo domu zákazníka,
- realizace rozvodných elektronických skříní do bytu nebo domu zákazníka,
- připojování bezdrátových tiskáren a dalších zařízení.

Spolek svým členům poskytuje internet standartně za cenu 300 korun českých měsíčně a průměrná rychlost internetu v síti se pohybuje okolo 75Mbit/s. Spolek nabízí svým členům možnost uplatnění slev pro lidi v důchodovém věku a osoby s průkazem ZTP. Svým uživatelům spolek účtuje poplatek za servis, jen pokud si řešený problém zavinil uživatel sám anebo chce provést práci, která není nezbytně nutná pro zajištění fungování internetového připojení. Takové služby si klienti nemusejí zaplatit, a platí proto většinou jen cenu elektronických prvků, které technik zanechal na místě během výkonu své práce. Spolek zároveň svým členům při instalaci a servisu nabízí několik cenových kategorií těchto prvků.

Většina členů spolku je k síti připojena pomocí bezdrátových technologií tvořených anténami, které se umisťují členům buď na střechu, nebo za okno. Spolek využívá převážně antény, které fungují na dvou hlavních frekvencích 5 GHz a 60 GHz. Historicky se uživatelé připojovali za využití 2,4 GHz frekvence, ale v dnešní době se již používá téměř výhradně 5 GHz. Ve většině případů jsou uživatelé připojeni za použití 5 GHz antény, ale pro náročnější spoje spolek zpravidla využívá 60 GHz. Spolek využívá také jiné frekvence jako 10 GHz, 2,4 GHz a 24 GHz, ale ty se vyskytují spíše méně a používají se ve specifických situacích, jako je na příklad přenos na velkou vzdálenost.

Spolek se dále zabývá realizací, návrhem a provozem optických sítí. Spolek má v současné chvíli ve správě pět oblastí realizovaných pomocí technologie přenosu dat přes optické kabely. Do každé optické sítě, kterou spolek vlastní, je připojeno okolo padesáti domácností. Tyto sítě se nacházejí ve Vejprnicích a Radčicích u Plzně. Ohledně optické technologie spolek spolupracuje se společností z Města Touškov nesoucí název TNtech, s. r. o. Tato společnost disponuje bohatšími zkušenosti v oblasti návrhu a realizace optických sítí. Navíc tato společnost nevytváří přímou konkurenci sledovanému spolku z důvodu odlišných lokalit, kde poskytují své služby, a proto tato společnost spolupracuje se sledovaným spolkem při realizaci a údržbě optických sítí.



Obrázek 2) Mapa pokrytí spolku. Zdroj: https://boknet.cz/mapa

2.1 Připojení uživatelů

Spolek si sám navrhuje svoji síť a zároveň si na svoje náklady zřizuje vysílače, které jsou důležitými body v síti, a realizuje propojení jednotlivých vysílačů. Jak již bylo v této práci řečeno, pro finální přenos dat od vysílače ke koncovému uživateli se ve většině případů používají antény, které fungují na 5GHz frekvenci. Na straně vysílače se používají antény všesměrové se záběrem v některých případech až 120°, naopak na straně uživatele se používají antény směrové, které mají záběr pouze několik málo stupňů.

Obrázek 3) Ubiquiti Rocket AC. Zdroj: https://www.wireless-stock.com/ubiquiti-rocket-ac-r5ac-ptp-airprism.html

Obrázek 4) Ubiquiti LBE-5AC-gen2. Zdroj: https://www.kasa.cz/pristupovy-bod-ap-ubiquiti-litebeam-5ac-gen2-lbe-5ac-gen2-bily/

2.2 Vysokorychlostní připojení firem

Spolek nabízí připojení i firmám vysokou rychlostí (pohybující se v řádech stovek megabitů za sekundu). Při realizaci takového připojení spolek musí zhotovit nový spoj, který je tvořen spojením jednoho vysílače s jednou anténou. Tyto přímé spoje jsou pro spolek nákladnější na realizaci. Zároveň spolek vyčlení dané firmě větší podíl konektivity sítě, aby firma mohla využívat toto vysokorychlostní spojení. Přesná cena takového připojení se navíc mění ještě v závislosti na dalších faktorech, jako je na příklad vzdálenost od vysílače. Cena takového připojení je výrazně vyšší než cena normálního připojení.

Obrázek 6) IgniteNet 60GHz. Zdroj: https://www.abctech.cz/?cls=stoitem&stiid=35964&gclid=Cj0KCQiAq5meBhCyARIsAJrtdr7xsKOOrjbrjdQxEcBSVWcwQPNaiWZ1cGgLlCHLAy7dFx6JiVktEKMaAoZyEALw_wcB

Obrázek 5) Ubiquiti Airfiber 60GHz. Zdroj: https://www.bohemiapc.cz/ubnt-airfiber-af60-60ghz-radio-4-6gbps-ota-vc-zalozniho-5ghz-866mbps-ota-1gbps-ethernet-cena-za-kus/

Pro tento typ připojení se nejčastěji používají antény, které fungují na 60 GHz frekvenci. V síti se používají 60 GHz antény primárně od dvou výrobců: Ubiquiti a IgniteNet. Se zvyšující se frekvencí spojení stoupá kapacita spoje, ale zároveň stoupá i citlivost spoje na vnější podmínky, jako jsou překážky v přímém výhledu, hustý déšť nebo sníh. Z těchto i dalších důvodů je realizace takového spoje náročnější a pro spolek nákladnější. Zároveň hraje i roli fakt, že se stoupající frekvencí klesá úhel pod kterým je anténa schopna udržet spolehlivé spojení z fyzikálních důvodů, které spojují frekvenci vlnění a velikost vln.

2.3 Optické sítě

Spolek má v současné chvíli pět oblastí, ve kterých má na své náklady realizované optické vedení. Toto vedení poté spolek na své náklady spravuje a využívá jej k poskytnutí připojení do sítě spolku pomocí optické technologie. Tři se nacházejí ve Vejprnicích, jedna v Radčicích u Plzně a jedna v Plzni na Skvrňanech. Optická síť s sebou přináší několik výhod a nevýhod.

Mezi výhody patří:

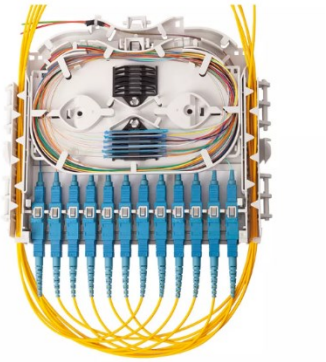
- vysoká rychlost spojení pomocí optických kabelů,
- vysoká kapacita spojení dvou telekomunikačních zařízení,
- vysoká odolnost vůči počasí,
- spojení se navzájem neruší díky omezení signálu na vnitřek optického kabelu.

Optické sítě spolu nesou i některé nevýhody:

- vysoká investice při realizaci spoje,
- vysoká cena vybavení nutné pro realizaci optické sítě,
- náročná oprava poškozeného kabelu,
- náročná manipulace. Optické kabely jsou mnohem křehčí než kabely metalické. Například při moc ostrém ohybu vzniká ve vlákně zlom, který znemožňuje další využití kabelu a kabel se musí ve většině případů celý nahradit.

Výhody optických sítí v některých lokalitách převažují nad nevýhodami, ale ne ve všech případech tomu tak je. Za předpokladu, že se s optickou sítí počítá již ve fázi plánování developerského projektu, tak výhody zpravidla převažují nad nevýhodami z důvodu preferování uložení optických kabelů do země. V připadě, že se optická síť chce dodělat v oblasti, která je již zastavěna, (jako je městské centrum), tak vykopání ulice a založení kabelů je finančně náročné, proto se tyto projekty spojují například s rekonstrukcí ulice.

Existuje i možnost optických převisů. Optický převis je technika, kdy jsou spojeny dvě střechy pomocí technologie optických kabelů převěšením kabelu mezi nimi. Pro tento účel spolek využívá speciální optické kabely, které mají do své svrchní ochrany zapracovaná nylonová vlákna, která zajišťují sílu nutnou, aby se kabel nepřetrhl. Tuto možnost má spolek realizovanou v Plzni v městské části Skvrňany, ale toto řešení je velice situační a ve velké většině situací je lepší zvolit bezdrátovou technologii místo optického převisu.



Obrázek 7) Příklad optické kazety s optickými sváry.
Zdroj: https://www.micostelcom.com/cs/opticke-kazety/opticka-kazeta-km-5

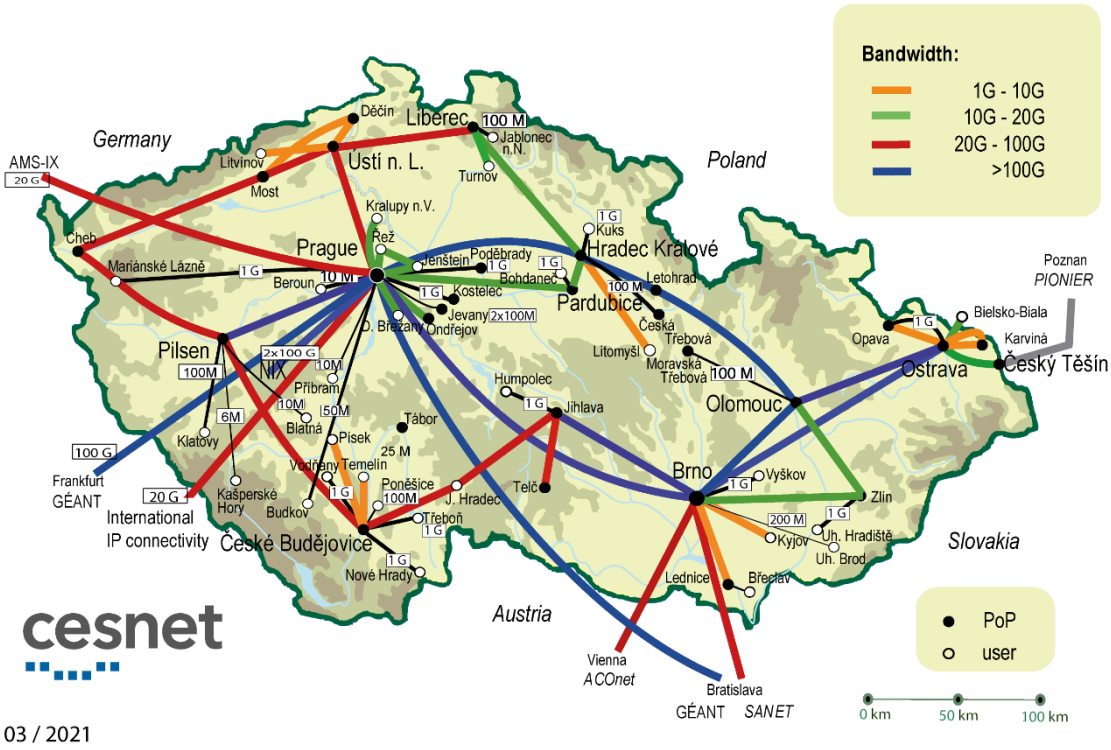
2.4 Návrh sítě

Spolek si platí konektivitu pro svoji síť od spolku CESNET připojením v Plzni na Nové Hospodě. Od založení spolku se vedení mezi lety 2006 a 2023 nezměnilo, a toto vedení síť rozšířilo ze sítě tří kamarádů na současnou síť čítající přes 2600 klientů. K tomuto číslu společnost spravuje dalších 270 vysílačů. Tito klienti a vysílače dohromady tvoří síť, kterou v současné době spolek vlastní. Růst sítě je organický a síť roste vždy na místě s největší poptávkou. Plánování umístění jednotlivých vysílačů probíhá podle těchto čtyř základních zásad návrhu bezdrátových sítí:

- Predikce: Je důležité zajistit dostatečné pokrytí signálem v celém prostoru, kde bude síť používána. Pokud potřebujete pokrýt velký prostor, můžete zvážit využití více přístupových bodů, které budou propojeny mezi sebou. Tato predikce je poté ověřována fyzickým průzkumem lokality.
- Kapacita: Je třeba zvážit, kolik uživatelů bude síť používat a jakou kapacitu budou potřebovat. Je třeba zajistit dostatečnou kapacitu sítě pro plynulý a spolehlivý provoz.
- Bezpečnost: Bezdrátové sítě jsou náchylnější k útokům a zabezpečení sítě by mělo být na vysoké úrovni. Doporučuje se využít šifrování a autentizaci uživatelů.
- Správa sítě: Je třeba zvážit, jak bude síť spravována a monitorována. Je vhodné využít nástroje pro centrální správu a monitorování sítě. (Smith, Clint P.E., and Daniel Collins. 2014)

Vzhledem k poměrně malému počtu zaměstnanců a velkému množství členů i vysílačů je síť, podle zkušeností autora práce, které autor získal při práci pro spolek jako správce sítě

a technik, navržena dobře a architektura sítě je stále vylepšována. V současné chvíli se však spolek dostává do situace, kdy je část sítě již velice zastaralá a je zapotřebí zlepšit použitou technologii na těchto místech. Zároveň má spolek poměrně dost žádostí o nové členy.



Obrázek 8) Mapa optické sítě spolku CESNET. Zdroj: https://www.cesnet.cz/sluzby/pripojeni/topologie/

2.5 Správa sítě

Síť spolku spravují správci sítě, které spolek zaměstnává. Tito správci zároveň fungují jako technici, kteří se starají o servis jednotlivých členů. Všichni technici spolku pracují ze svého bydliště a pro výkon prací v síti využívají svá vlastní vozidla. Na konci každého měsíce všichni technici podávají hodinový výkaz práce, na základě čehož jsou placeni. Společně s hodinovým výkazem všichni technici zároveň podávají spolku i výkaz najetých kilometrů, podle kterých spolek technikům hradí náklady vzniklé v souvislosti s využíváním vozidla. Síť v momentální chvíli spravuje 8 pracovníků. Mezi tyto osoby patří: hlavní administrátor, hlavní technik, jedna operátorka telefonní linky a pět techniků. Spolek často zaměstnává správce na částečný úvazek nebo brigádníky. Díky velice flexibilní pracovní době a hodinovému platu to představuje vhodné zaměstnání pro zaměstnance, kteří potřebují pohyblivou pracovní dobu a nemají čas na zaměstnání na plný úvazek. To splňují například právě studenti. Spolek má v současné době hodně

zakázek a jediné, co spolku brání v dalším růstu, je momentální absence zájemců o zaměstnání v tomto oboru.

3 Analýza nákladů vznikajících při činnosti spolku

Spolek zajišťuje svým členům bezplatný servis za předpokladu, že servisní činnost není způsobená uživatelem, ale je nutná pro správný chod internetu. Tyto služby zahrnují, ale nejsou omezené na:

- prvotní instalaci internetového připojení,
- výměnu internetové antény za novější generaci,
- servis Wi-Fi zařízení,
- výměnu defektního kabelu.

Spolek hradí svým členům pouze práci. Jakýkoli materiál, který je nutné vyměnit, si hradí uživatel sám.

Spolek se v současné době potýká se situací, kdy by zvýšení měsíčních příspěvků mohlo způsobit ztrátu členů, ale nároky na internetové připojení se stále zvyšují a ceny nových technologií rovněž. Proto se spolek snaží optimalizovat své vnitřní procesy.

Spolek se zabývá optimalizací primárně těchto dvou procesů:

- efektivností techniků
- efektivností architektury internetové sítě.

3.1 Efektivnost techniků

V současné době je technikům práce přidělována metodou odhadu nejkratší trasy, kterou jim daný den připraví a domluví operátorka telefonní linky. Tato zaměstnankyně spolku zároveň zpracovává žádosti zadané pomocí telefonátu a ty, které jsou zadané pomocí internetové stránky. Operátorka poté zkombinuje časové možnosti členů a stanoví technikům trasu, kterou sestaví odhadem.

V momentální implementaci efektivita trasy není pro spolek prioritou. Technici si mohou domluvit i návštěvu člena přímo bez zapojení operátorky telefonní linky. Takové servisy tvoří malý podíl a technici si je domlouvají sami, a proto nad nimi spolek nemá kontrolu. Tento přístup není podporován z důvodu efektivity, ale v některých případech je tato možnost tolerována.

Průběh zpracování žádostí je tedy zorganizován následujícím způsobem:

- Člen spolku nahlásí problém buď za pomoci použití mobilního telefonu, nebo podá žádost přes internetové stránky. Servis nahlášený přes internetovou stránku je automaticky vložen do vnitřního informačního systému spolku, kam má každý technik a operátorka telefonní linky přístup.
- Servisy hlášené na firemní infolinku a žádosti registrované pomocí internetové stránky jsou poté zkombinovány a je vzato přibližně šest nejstarších žádostí.
- Operátorka telefonní linky poté zkombinuje časové dispozice technika na daný den, adresy členů a odhadem určí ideální trasu a časy ve kterých by se technik měl dostavit na místo.
- Operátorka telefonní linky poté uskuteční telefonický hovor na každý servis a zkusí domluvit dané členy v čas jim určený.
- Pokud člen v daný čas nemůže, je kvůli němu trasa buď upravena, nebo je vyloučen a je s ním rovnou domluvený vlastní den, kdy bude servis možný. Tento člen je poté zpracován v domluvený den prioritně.

Pro optimalizaci procesu pohybu techniků mezi servisy budou klíčové následující informace:

- adresy členů (vrcholy v grafu)
- adresy techniků (počáteční vrcholy)
- trasy mezi jednotlivými vrcholy (hrany grafu)

Pro výpočet konkrétního ekonomického přínosu bude nutné zjistit vzdálenost, jakou technik firmy reálně vykonal za současného systému. Poté pro identickou situaci navrhnout pomocí vhodné optimalizační metody novou trasu, která by minimalizovala ujetou vzdálenost při obsluze členů spolku. Z těchto výsledků bude autor práce schopen vypočítat teoretické úspory, kterých by spolek mohl dosáhnout.

3.2 Efektivnost architektury internetové sítě

Spolek využívá dvě možnosti na budování sítě. Buď si spolek na svoje náklady buduje vysílače, nebo si pronajímá již hotový vysílač a osazuje na něj své vlastní antény. Spolek je připojen v Plzni na Nové Hospodě na páteřní optickou síť sdružení CESNET, odkud poté distribuuje internetové připojení čistě svojí sítí. Síť se postupně větví do menších částí celku, a proto se postupně snižuje i celková konektivita, aby byli uživatelé spokojeni

a spolek nekupoval spoje s moc velkou kapacitou přenosu, musí se k této snižující kapacitě přihlížet a brát ji v potaz při návrhu spojů. Tento problém je v současnosti ve spolku řešen zkušenostmi, protože tento spolek stále řídí původní zakladatelé. Ti vše řeší na bázi zkušeností získaných za posledních 17 let od vzniku spolku.

Tento problém momentálně není akutní pro chod spolku, protože vše za současného stavu funguje. Pro informovanost vedení tohoto spolku by však mohla být užitečná informace o alternativním návrhu struktury sítě sestavené vhodným algoritmem na základě kritérií respektujících dostatečnou kapacitu spojů vyžadovanou členy a související náklady na kapacitu spoje.

Pro tento rozbor budou hlavní:

- adresa vysílače,
- spojení vysílače se sousedními vysílači, případně se členy,
- kapacita jednotlivých spojení,
- vytížení spojení,
- priorita jednotlivých koncových uživatelů.

Vzhledem k momentálnímu stavu ve spolku a ke zkušenostem, které získal autor práce během vykonávání servisní činnosti ve spolku, dospěl autor této práce k závěru, že pro potřeby této práce se bude dále zabývat pouze problémem optimalizace pohybu techniků v síti. Analýza efektivity architektury sítě a její optimalizace je výrazně složitější problém, jehož řešení by zřejmě překračovalo rozsah této bakalářské práce, a proto se této problematice autor práce již níže věnovat nebude.

4 Praktické zpracování

Vzhledem k omezenému počtu členů, které může technik navštívit během jednoho dne, se autor práce po zvážení okolností rozhodl použít pro řešení problému obchodního cestujícího přístup hrubé síly. Pro řešení algoritmu si autor práce napsal program v jazyce Java, který umí pracovat s daty, která jsou extrahována z vnitřního systému spolku, a dovoluje zadávat trasu pro optimalizaci pouze jako dlouhý řetězec znaků oddělovaný čárkou, který obsahuje identifikační kombinaci čísla členů a zkratky odlišující členy a vysílače. Tato kombinace je pro každý záznam v systému jedinečná.

Autor práce zvolil program IntelliJ IDEA pro zpracování a naprogramování algoritmu pro výpočet nejkratší trasy. Program byl napsán v jazyce Java. Program IntelliJ IDEA a jazyk Java si autor vybral především z důvodu předchozích zkušeností s tímto jazykem.

Existuje mnoho důvodů, proč si autor práce vybral pro zpracování této bakalářské práce Java a IntelliJ IDEA. Jednou z hlavních výhod použití Javy je její nezávislost na platformě. To znamená, že Java program může běžet na jakémkoli systému s JVM, ať už se jedná o počítač s operačním systémem Windows, Mac nebo Linux. To činí Javu ideální pro vývoj softwaru, který je třeba nasadit na více platformách, jako jsou webové aplikace nebo mobilní aplikace. Výhodou programu IntelliJ IDEA je možnost odstraňování chyb v programu průchodem programu řádku po řádce díky funkci debug.

Další výhodou Javy je její schopnost zpracovávat i veliké objemy dat. Java je navržena tak, aby byla vysoce škálovatelná, což znamená, že dokáže snadno zpracovat velké množství dat. To jí činí ideálním jazykem pro vývoj podnikových aplikací, které musí být schopny zpracovávat mnoho uživatelů nebo dat.

V České republice je použití Javy a IntelliJ IDEA velmi rozšířené. V oblasti IT a softwarového inženýrství existuje mnoho společností, které se specializují na vývoj softwaru v Javě a používají IntelliJ IDEA jako své primární vývojové prostředí.

4.1 Základní data

Základní data potřebná pro výpočet problému obchodního cestujícího získal autor práce po udělení přístupu do vnitřního systému spolku. Samotné stáhnutí dat proběhlo prostřednictvím tabulky ve formátu „.csv". Tato tabulka obsahuje pro každý vstup identifikátor, který stanoví, jestli je vstup v tabulce členem nebo vysílačem. Dále

obsahuje pro každý prvek v tabulce jeho identifikační číslo, název v systému a jeho GPS souřadnice. U každého člena má spolek v systému uloženy i informace typu telefonní číslo a emailová adresa, ale tyto informace nejsou pro zpracování této bakalářské práce nikterak důležité, a proto je autor nestahoval. V praxi by však kvůli komunikaci s členy spolku byly tyto informace klíčové. Případná úprava programu tak, aby zobrazoval i tyto informace, by však byla minimální.

4.1.1 Vrcholy grafu

Jeden řádek ve zdrojové tabulce definuje jeden potenciální vrchol grafu a každý řádek má zadané tyto atributy:

- Polohu vrcholu; udává se v souřadnicích GPS.
- Název vrcholu grafu; tento název je v systému používán pro identifikaci členů a vysílačů při komunikaci mezi zaměstnanci spolku.
- Identifikátor; určuje, zda se jedná o uživatele nebo vysílač.
- Identifikační číslo; automaticky přiřazované všem záznamům v systému.

4.1.2 Počáteční vrcholy grafu

Počáteční vrcholy tvoří adresy techniků. Před tím, než by se spustil samotný algoritmus pro řešení problému obchodního cestujícího, tak by operátorka telefonní linky přiřadila jednotlivým technikům servisy, které jsou jim nejblíže, a až poté spustila algoritmus pro každou podmnožinu zvlášť.

4.1.3 Množina hran grafu

Pomocí internetových služeb, které nabízí společnost OpenRouteService, si autor práce zjistí vzdálenost mezi všemi vrcholy v grafu. Tyto hodnoty budou ukládány do matice vzdáleností. Program autora práce zohledňuje případnou asymetrii vzdáleností mezi jednotlivými vrcholy.

Základní data autor obdržel jako tabulku ve formátu CSV, která obsahuje šest sloupců. Protože v tabulce nejsou psáni jen členové, ale i vysílače, které také potřebují servis, obsahuje tabulka sloupec, který označuje, zda se jedná o člena nebo vysílač. Pokud je v prvním sloupci záznam „user“, poté je v druhém sloupci zadané identifikační číslo člena a třetí sloupec je prázdný, tedy „NULL“. Pokud je záznam v prvním sloupci „ap“, tak má druhý sloupec hodnotu „NULL“ a třetí sloupec má hodnotu identifikačního čísla vysílače,

který má přiřazený v systému. Tento systém je zavedený, aby jeden člen a jeden vysílač mohli sdílet jedno číslo v systému, ale v kombinaci s prvním sloupcem je každý záznam jedinečný. Třetí sloupec obsahuje textové pole označující název záznamu v tabulce. Čtvrtý a pátý sloupec obsahují GPS souřadnice záznamu, které jsou poté potřebné pro výpočet trasy.

user	5186	NULL	nanacikova	49.6943142	13.2553107
user	5187	NULL	Stupka	49.7532822	13.2819639
user	5188	NULL	smidovaMarie	49.7437254	13.4116016
user	5194	NULL	Tuna_J	49.6999628	13.2189106
ap	NULL	1	Vejprnice bytovka	49.7320472	13.2815833
ap	NULL	2	Vejprnice beta	49.7281333	13.2858667

Obrázek 9) Příklad dat ve vstupní tabulce. Zdroj: vnitřní systém spolku

4.2 Popis programu

Program je navržen tak, aby se dal využít pro výpočet nejkratší trasy mezi zadanými vrcholy. Vstup programu je zadán jako dlouhý řetězec znaků, kde jsou zadáni uživatelé nebo vysílače, které vyžadují návštěvu od technika. Tyto jednotlivé vrcholy jsou oddělené čárkou a jsou zadávány ve tvaru „cl3“, kde „cl“ označuje, že se jedná o člena a číslo 3 značí identifikační číslo v systému, který vrchol má. Místo „cl“ je možnost napsat „ap“, a to by programu sdělilo informaci, že se jedná o vysílač, nikoli o člena.

Program si tento řetězec oddělí pomocí funkce, která rozdělí tento dlouhý textový řetězec podle čárek, a poté načte data ze souboru ve formátu CSV obsahující seznam vrcholů, z nichž každý nese určitý název a GPS souřadnice. Vrcholy jsou ze základních dat načítány vyhledáváním v prvních třech sloupcích podle vstupního souboru. Program si poté sestaví seznam vrcholů ze souboru CSV, které odpovídají vstupním podmínkám, a používá tyto vrcholy k vytvoření matice sousednosti hran.

Po vytvoření matice sousednosti hran program využívá napojení na API OpenRouteService k výpočtu vzdálenosti mezi každou dvojicí vrcholů. To činí tak, že provádí HTTP požadavky na API a zpracovává JSON odpovědi. Program poté používá algoritmus hrubé síly pro řešení problému obchodního cestujícího (v angličtině Traveling Salesman Problem, TSP), aby našel nejkratší trasu, která navštíví všechny vrcholy.

Nakonec program vypisuje seznam vrcholů v pořadí, v jakém by měly být navštíveny, aby byla realizována nejkratší trasa, a délku této trasy. Program z důvodu porovnání ekonomického přínosu do firmy také vypisuje vzdálenost trasy v pořadí zadaném, tedy v pořadí, v jakém technik trasu reálně absolvoval.

Program je tvořen několika třídami, které dohromady utváří celek programu. Tyto třídy jsou:

- App.java – Tato třída je hlavní a běží v ní hlavní program, který si potom volá ostatní třídy.
- Node.java – Tato třída představuje vrchol v grafu, který může být buď počáteční vrchol, vysílač nebo člen. Každý vrchol nese název, má GPS souřadnice a identifikační číslo.
- Edge.java – Tato třída reprezentuje hranu mezi dvěma vrcholy v grafu. Každá hrana disponuje počátečním a koncovým vrcholem a vzdáleností mezi nimi.
- TSP.java – Tato třída je implementace samotného algoritmu na řešení problému obchodního cestujícího.
- Repository.java – Tato třída slouží jako rozhraní pro přístup k databázi, která ukládá, upravuje nebo maže informace o vrcholech v databázi.

4.2.1 App.java

Tato třída programu obsahuje kód v Javě implementující aplikaci, která zpracovává seznam GPS souřadnic a pomocí OpenRouteService API vypočítává vzdálenosti mezi nimi, aby nakonec vyřešila problém obchodního cestujícího (TSP) a našla nejkratší cestu spojující všechny vrcholy.

V aplikaci je definována třída „App”, která obsahuje metody pro získání vrcholů z GPS souboru. Získání vzdáleností mezi vrcholy probíhá pomocí API OpenRouteService a následuje vytvoření matice vzdáleností mezi vrcholy. Třída App obsahuje také metodu „main”, která obsahuje část kódu starající se o zpracovávání vstupu od uživatele, načtení GPS souboru a pomocí výše zmíněných metod vypočítání nejkratší cesty spojující všechny vrcholy.

Metoda „getNodesFromFile” načte GPS soubor a vrátí seznam vrcholů, kde každý vrchol reprezentuje jeden řádek v souboru. Metoda zkontroluje, zda záznam existuje, a pokud neexistuje, vypíše chybové hlášení a ukončí aplikaci. Pro každý vrchol se ověří, zda je

obsažen v množině požadavků, a pokud ano, přidá se do seznamu vrcholů. Metoda poté vrátí seznam vrcholů.

Metoda „getDistance" vypočítává vzdálenost mezi dvěma vrcholy pomocí API OpenRouteService. Používá se GET požadavek na konkrétní URL adresu, která zahrnuje souřadnice počátečního a koncového vrcholu trasy. Poté se získává odpověď a extrahuje se vzdálenost z odpovědi pomocí metody „extractProperty". Vzdálenost se pak ukládá do instance třídy „Edge".

Metoda „main" přijímá vstup od uživatele, který je tvořen seznamem požadovaných vrcholů grafu oddělených čárkami. Poté se načte vstupní soubor a získají se GPS souřadnice vrcholů. Pro vrcholy, které odpovídají těmto požadavkům je následně vypočítána matice vzdáleností pomocí metody „createDistanceMatrix". Poté je volán samotný algoritmus určený pro řešení problému obchodního cestujícího pomocí metody „TSP.getTsp(matice vzdáleností)". Tato metoda po zavolání převezme matici vzdáleností, která je jí podána jako vstupní argument a vypočte nejkratší trasu grafem za použití přístupu hrubé síly.

Další důležitou metodou v této třídě je „createDistanceMatrix()", která vytváří matici vzdáleností mezi vrcholy. Tato metoda přijímá seznam vrcholů a vrací matici hran. Výsledná matice je dvourozměrné pole objektů třídy Edge, které představují jednotlivé hrany grafu. Pro výpočet vzdálenosti mezi vrcholy se používá metoda „getDistance()", která využívá externí API OpenRouteService, které vrací vzdálenost mezi dvěma vrcholy na základě jejich GPS souřadnic.

Další důležitou částí této třídy je metoda „main()", která slouží jako vstupní vrchol pro spuštění programu. V této metodě se nejprve načítá vstup od uživatele, který se rozdělí na seznam jednotlivých zkratek uživatelů a vysílačů pomocí čárek. Poté se načítají vrcholy ze vstupního souboru a vytváří se matice vzdáleností mezi nimi pomocí metody createDistanceMatrix(). Následně se hledá nejkratší cesta pomocí algoritmu pro řešení problému obchodního cestujícího a vypisuje se výsledek.

4.2.2 Node.java

Tato třída představuje vrchol v grafu, který může být buď počáteční vrchol, vysílač nebo člen. Každý vrchol má název, GPS souřadnice a identifikační číslo.

Konstruktor třídy Node zpracovává vstupní řetězec a extrahuje z něj informace o vrcholech. Vstupní řetězec obsahuje informace oddělené čárkami a uzavřené do uvozovek. Po odstranění uvozovek se řetězec rozdělí pomocí funkce split() na jednotlivé vrcholy.

Funkce isAp() vrací informaci o tom, zda je tento konkrétní vrchol vysílač. Funkce setAp() slouží k nastavení této informace.

Funkce equals() a hashCode() jsou psány tak, aby porovnávaly vrcholy na základě jejich názvu, identifikačního čísla a typu vrcholu.

Funkce toString() vrací řetězec obsahující název, GPS souřadnice, identifikační číslo a informaci o tom, zda se jedná o vysílač, nebo člena. GPS souřadnice jsou v této funkci „převráceny", tj. zeměpisná šířka je zobrazena jako druhá položka a zeměpisná délka jako první položka, aby byl výstup v souladu s konvencí většiny zeměpisných systémů.

Funkce „toInputString()" slouží k vytvoření řetězce, který lze použít jako vstupní řetězec pro další zpracování. Pokud GPS souřadnice nejsou k dispozici (jsou nastaveny na hodnotu „NULL"), pak tato funkce vrátí prázdný řetězec. Jinak se do výstupního řetězce zapíše zkratka pro typ vrcholu (vysílač / člen) a identifikační číslo vrcholu.

4.2.3 Edge.java

Třída Edge v tomto programu reprezentuje hranu mezi dvěma vrcholy v grafu a matice vzdáleností je potom tvořena mnoha instancemi třídy Edge. Každá hrana má počáteční a koncový vrchol a vzdálenost mezi nimi. Třída má několik metod pro získání a nastavení těchto vlastností.

Konstruktorem se zadávají dva vrcholy, které hrana spojuje. Vzdálenost je při vytvoření hrany neznámá a může být nastavena zpětně metodou setDistance.

Metoda getDistance je určena k získání vzdálenosti mezi počátečním a koncovým vrcholem hrany. Pokud vzdálenost není nastavena, metoda vrátí hodnotu NULL.

Metody getFrom a getTo slouží k získání počátečního a koncového vrcholu hrany.

Metody equals a hashCode jsou použity pro porovnání hran a zajištění unikátnosti hran v grafu. Hrany jsou porovnávány na základě počátečního a koncového vrcholu, aby se zajistilo, že každá hrana má pouze jednu instanci v grafu.

Třída Edge je použita v kombinaci s třídou Node pro reprezentaci grafu. Třída Node reprezentuje vrchol v grafu a obsahuje seznam hran, které spojují tento vrchol s ostatními vrcholy. Třída TSP (Traveling Salesman Problem) používá třídy Edge a Node pro řešení problému obchodního cestujícího.

Třída Edge také obsahuje metodu setDistance, která slouží k nastavení vzdálenosti mezi počátečním a koncovým vrcholem hrany. Tato metoda je použita v třídě TSP při vytváření matice vzdáleností pro všechny hrany v grafu. Tato metoda využívá napojení na OpenRouteService pomocí API, které vrací dané vzdálenosti.

Pro získání vzdálenosti mezi vrcholy je v třídě TSP použita matice vzdáleností, která je vytvořena metodou createDistanceMatrix. Tato metoda vytváří matice vzdáleností pro všechny hrany v grafu. Pro každou hranu v grafu je vzdálenost uložena v instanci třídy Edge. Metoda createDistanceMatrix projde všechny hrany v grafu a uloží jejich vzdálenosti do matice vzdáleností.

Konstruktor třídy Edge přijímá dva objekty typu Node - počáteční a koncový vrchol hrany - a vytváří novou instanci třídy Edge. Třída také poskytuje metody pro získání a nastavení délky vzdálenosti hrany, a pro získání počátečního a koncového vrcholu. Program bere v úvahu nesymetričnost trasy a proto vytváří matici vzdáleností kompletní a zkoumá pro dva vrcholy vždy oba směry.

4.2.4 TSP.java

Toto je třída v jazyce Java, která implementuje algoritmus pro řešení problému obchodního cestujícího (Traveling Salesman Problem, TSP). V této implementaci je TSP řešen hrubou silou pomocí iterace přes všechny možné permutace vrcholů a výběrem té s nejkratší celkovou vzdáleností. Hrubá síla (brute force) je jednoduchá metoda, která spočívá v prohledání všech možných řešení. V případě problému obchodního cestujícího to znamená, že musíme projít všechny možné permutace vrcholů a najít tu s nejkratší vzdáleností. V této metodě počet permutací roste exponenciálně s počtem vrcholů, takže pro větší instance problému je výpočet velmi časově náročný a prakticky neřešitelný.

Třída obsahuje několik statických metod. Metoda getTsp bere jako vstupní parametr matici hran Edge, která reprezentuje vzdálenosti mezi vrcholy, a vrací objekt typu TspResult, který obsahuje minimální vzdálenost a optimální cestu.

Metoda nextPermutation generuje další permutaci pole, které se používá k iteraci přes všechny možné permutace vrcholů.

Metoda reverse obrací část pole, což se používá ke generování další permutace.

Metoda createDistanceMatrix bere jako vstupní parametr matici objektů typu Edge a vrací dvoudimenzionální pole hodnot typu Long, které reprezentují vzdálenosti mezi vrcholy.

TspResult je jednoduchá datová struktura, která obsahuje minimální vzdálenost a optimální cestu.

4.2.5 Repository.java

V Javě se obvykle třída Repository používá jako abstraktní vrstva, která poskytuje způsob správy operací jako jsou ukládání a načítání dat. Třída Repository funguje jako prostředník mezi aplikací a podkladovým mechanismem pro ukládání dat, jako je například databáze.

Třída Repository zapouzdřuje logiku potřebnou k interakci s datovým úložištěm, včetně dotazování na data, vkládání, aktualizování a mazání záznamů. Poskytuje konzistentní rozhraní pro interakci aplikace s úložištěm dat, skrývající složitosti podkladového mechanismu pro ukládání dat.

Použitím třídy Repository může aplikace zůstat oddělena od podrobností toho, jak jsou data uložena a načítána. To usnadňuje výměnu mechanismu ukládání dat, jako je například přechod z relační databáze na dokumentově orientovanou databázi, bez vlivu na kód aplikace.

Celkově řečeno, třída Repository je určena pro běžný přístup k organizaci logiky přístupu k datům v objektově orientovaném programování a poskytuje způsob správy operací ukládání a načítání dat strukturovaným a udržovatelným způsobem.

Třídu Repository si v tomto programu autor práce zavedl, aby si ulehčil práci s databází, nad kterou počítá minimální trasu a spouští algoritmus pro řešení této trasy. Tato třída má za funkci fungovat jako rozhraní pro přístup k databázi, která ukládá informace o uzlech v reálné bezdrátové síti.

Metoda persist() slouží k uložení vrcholu do databáze. Přijímá parametr typu Node, což je třída reprezentující vrchol v grafu. Po vytvoření nového objektu Session z

sessionFactory a začátku transakce se pomocí metody persist() objekt vrcholu uloží do databáze a transakce se provede a ukončí.

Metoda getById() slouží k získání vrcholu z databáze podle zadaného identifikátoru id. Používá se v ní dotazovací jazyk HQL (Hibernate Query Language), který je podobný SQL. Metoda delete() slouží k odstranění vrcholu z databáze podle zadaného identifikátoru id.

Celkově tedy tato třída poskytuje jednoduché rozhraní pro přístup k databázi a umožňuje uložit, získat a odstranit uzly v reálné bezdrátové síti. Hibernate zajišťuje, že data jsou uložena v databázi v požadovaném formátu a lze je snadno vyhledávat pomocí dotazů v jazyce HQL.

4.3 Výstupní data

Autor práce výsledná data shromažďuje z důvodu porovnání nově vypočtené optimální trasy vůči původní trase v tabulce s názvem „main data.xlsx“. V této tabulce jsou uvedeny jednotlivé denní trasy, které procházely optimalizačním algoritmem.

V tabulce je každý den reprezentován jedním řádkem. Každý řádek obsahuje datum, ve kterém byla daná trasa vykonána technikem, a jednotlivé servisy na trase. V dalším sloupci je poté uvedena vzdálenost trasy bez použití optimalizačního algoritmu. Následující sloupec obsahuje pořadí jednotlivých zastávek pro optimalizovanou trasu. V posledním sloupci je uvedena délka optimalizované trasy neboli minimální délka cesty, kterou mohl technik absolvovat.

Záznamů je v tabulce celkem šedesát, a tyto záznamy reprezentují sběr dat od začátku listopadu roku 2022 do konce března roku 2023. Trasa je v tabulce zaznamenána jako několik na jednom řádku po sobě následujících buněk, které obsahují označení jednotlivých zastávek na trase. Každá zastávka je reprezentována číslem s předponou, konkrétně „cl“ označující člena a „ap“ označující vysílač. Čísla jsou stejná, jako mají jednotliví členové, potažmo vysílače uvedené v systému firmy.

Délka trasy, kterou technik absolvoval, je počítána za použití napojení přes API do systému společnosti OpenRouteServis a je uvedena v metrech. To samé platí pro délku minimální trasy. Poslední údaj uvedený v této tabulce je pořadí vrcholů v optimální trase. To je uvedeno jako řetězec čísel označující ideální pořadí zastávek, které v souhrnu tvoří trasu s minimální délkou.

1	Datum	Počáteční pořadí						Vstup programu pro řešení TSP	Vzdálenost	Optimální trasa	Minimální vzdálenost
2	1.11	cl4405	cl4443	cl2625				cl0,cl4405,cl4443,cl2625	21273	2,3,1	20978
3	3.11	cl746	cl387	cl3802	ap247	cl1010		cl0,cl746,cl387,cl3802,ap247,cl1010	38578	5,1,2,3,4	33699
4	4.11	cl830	cl663	cl3416	ap18	ap87	cl1237	cl0,cl830,cl663,cl3416,ap18,ap87,cl1237	95459	1,2,6,5,4,3	51781
5	7.11	cl1302	cl4329	cl4817	cl1284	cl2966	ap87	cl0,cl1302,cl4329,cl4817,cl1284,cl2966,ap87	97265	6,2,5,4,1,3	81680
6	8.11	ap88	cl3114	cl3494	cl1855	cl3521		cl0,ap88,cl3114,cl3494,cl1855,cl3521	103699	1,4,2,5,3	61420
7	11.11	cl1224	ap151	ap199	cl264	cl518		cl0,cl1224,ap151,ap199,cl264,cl518	100359	4,1,3,5,2	73388
8	12.11	cl267	cl510	cl1275	cl3009	cl3152		cl0,cl267,cl510,cl1275,cl3009,cl3152	31852	5,3,2,4,1	29660
9	13.11	cl4248	cl5128	cl1239	cl2732	cl2353	cl4816	cl0,cl4248,cl5128,cl1239,cl2732,cl2353,cl4816	113197	5,2,1,3,6,4	80219
10	16.11	cl27	cl204	cl640	cl3740	cl4553	cl4854	cl0,cl27,cl204,cl640,cl3740,cl4553,cl4854	43223	1,5,4,6,3,2	28988
11	17.11	cl4522	cl3691	cl491	cl4167	ap246		cl0,cl4522,cl3691,cl491,cl4167,ap246	22157	2,5,1,4,3	18412
12	18.11	cl4581	ap240	cl2643	cl958	cl1162		cl0,cl4581,ap240,cl2643,cl958,cl1162	61284	2,1,4,5,3	54053
13	20.11	cl1402	cl1792	cl2113	cl3799			cl0,cl1402,cl1792,cl2113,cl3799	47573	2,1,4,3	31101
14	21.11	cl4764	cl84	cl434	cl904	cl898	cl1239	cl0,cl4764,cl84,cl434,cl904,cl898,cl1239	68893	4,5,6,2,3,1	57896
15	22.11	cl2635	cl2919	cl4256	cl2346	cl1079		cl0,cl2635,cl2919,cl4256,cl2346,cl1079	114053	2,3,4,5,1	89362
16	23.11	cl2132	cl3590	cl3116	ap146	cl365	cl791	cl0,cl2132,cl3590,cl3116,ap146,cl365,cl791	62541	3,1,2,4,6,5	51858
17	27.11	cl1318	cl1358	cl439	cl4852	cl3120	cl1583	cl0,cl1318,cl1358,cl439,cl4852,cl3120,cl1583	49959	6,4,5,1,3,2	37277
18	28.11	cl3670	cl3935	ap70	ap236	cl843		cl0,cl3670,cl3935,ap70,ap236,cl843	67837	2,4,5,1,3	44387
19	2.12	cl1974	cl120	cl3648	cl1416	cl2570	cl1112	cl0,cl1974,cl120,cl3648,cl1416,cl2570,cl1112	67411	5,3,6,2,4,1	57947
20	3.12	cl4019	cl3989	cl554	cl1428	cl2556	cl2465	cl0,cl4019,cl3989,cl554,cl1428,cl2556,cl2465	85675	2,1,5,6,4,	76368
21	5.12	cl2556	cl2450	cl2577	cl1907	cl3901		cl0,cl2556,cl2450,cl2577,cl1907,cl3901	62167	5,4,1,3,2	27995
22	6.12	cl1774	cl3056	cl3324	cl3674	cl729	cl4342	cl0,cl1774,cl3056,cl3324,cl3674,cl729,cl4342	89962	3,5,2,6,1,3	55262
23	10.12	cl1371	cl2090	cl2619	cl129	cl4704	ap126	cl0,cl1371,cl2090,cl2619,cl129,cl4704,ap126	34985	3,5,6,2,4,1	30375
24	11.12	cl4999	cl1676	cl1105	cl2661	cl3411		cl0,cl4999,cl1676,cl1105,cl2661,cl3411	72363	3,2,4,5,1	62343
25	12.12	cl3624	cl4677	ap193	cl3222	cl3277	cl4162	cl0,cl3624,cl4677,ap193,cl3222,cl3277,cl4162	39435	4,5,6,1,3,2	28970
26	13.12	cl4383	cl4946	cl411	ap200	ap9		cl0,cl4383,cl4946,cl411,ap200,ap9	62077	3,1,2,5,4	36507
27	20.12	cl4832	cl3624	cl2545	cl483	ap261		cl0,cl4832,cl3624,cl2545,cl483,ap261	65826	4,3,2,1,5	58243
28	21.12	ap280	cl12	cl90	cl70	cl306	cl2371	cl0,ap280,cl12,cl90,cl70,cl306,cl2371	14227	2,4,3,5,6,1	14120
29	22.12	cl659	cl2235	ap180	cl1204	cl4801		cl0,cl659,cl2235,ap180,cl1204,cl4801	51079	1,2,3,4,5	51079
30	23.12	cl659	cl2801	cl3117	ap174			cl0,cl659,cl2801,cl3117,ap174	37738	1,4,3,2	37307
31	2.1	cl621	cl4140	cl1136	cl1554	cl3237		cl0,cl621,cl4140,cl1136,cl1554,cl3237	40868	5,4,2,3,1	32778
32	4.1	cl1727	cl4015	cl4837	cl1156			cl0,cl1727,cl4015,cl4837,cl1156	60961	3,1,4,2	47981
33	9.1	cl1887	cl2260	cl1347	cl3369	cl2681		cl0,cl1887,cl2260,cl1347,cl3369,cl2681	81716	3,2,1,5,4	75357

Obrázek 10) Tabulka „main data.xlsx“ část 1. Zdroj: vlastní zpracování

33	9.1	cl1887	cl2260	cl1347	cl3369	cl2681		cl0,cl1887,cl2260,cl1347,cl3369,cl2681	81716	3,2,1,5,4	75357
34	11.1	cl1535	ap290	cl1527	cl2149	cl1185	cl4154	cl0,cl1535,ap290,cl1527,cl2149,cl1185,cl4154	42980	6,4,5,1,3,2	33104
35	16.1	cl1864	cl2434	cl1141	cl4208	cl2449	cl3483	cl0,cl1864,cl2434,cl1141,cl4208,cl2449,cl3483	103393	3,1,2,4,5,6	63108
36	18.1	cl1345	cl2570	ap58	cl1080	cl1633	cl3496	cl0,cl1345,cl2570,ap58,cl1080,cl1633,cl3496	63691	6,2,1,5,4,3	51184
37	19.1	cl652	cl451	cl1310	cl4734	cl2442		cl0,cl652,cl451,cl1310,cl4734,cl2442	37554	5,2,3,4,1	26584
38	24.1	cl1335	ap149	cl408	cl5135	cl783		cl0,cl1335,ap149,cl408,cl5135,cl783	50023	2,4,3,5,1	44493
39	25.1	ap297	cl4526	cl2223	cl3078	cl1361		cl0,ap297,cl4526,cl2223,cl3078,cl1361	46395	2,4,5,1,3	30576
40	30.1	cl3480	cl2167	cl4061	cl3056	cl3461		cl0,cl3480,cl2167,cl4061,cl3056,cl3461	80103	1,3,5,2,4	71740
41	2.2	cl4115	cl1541	ap218	cl699	cl3465		cl0,cl4115,cl1541,ap218,cl699,cl3465	61306	5,3,1,2,4	57415
42	6.2	cl2366	cl2990	ap1010	cl4703			cl0,cl2366,cl2990,ap1010,cl4703	102773	2,1,3,4	97233
43	7.2	cl2265	cl4153	cl1359	ap223	cl1907		cl0,cl2265,cl4153,cl1359,ap223,cl1907	100904	4,5,2,1,3	87666
44	8.2	cl2332	cl4647	cl4345	cl3725	cl4645		cl0,cl2332,cl4647,cl4345,cl3725,cl4645	40458	1,4,3,5,2	36276
45	13.2	cl1367	cl3183	cl3641	cl3829	ap1002		cl0,cl1367,cl3183,cl3641,cl3829,ap1002	94379	3,1,5,2,4	52633
46	16.2	cl2235	ap250	cl2073	cl1264	cl4217	cl3454	cl0,cl2235,ap250,cl2073,cl1264,cl4217,cl3454	42377	6,3,1,4,5,2	35829
47	20.2	cl456	cl4283	ap88	cl1475	cl3931		cl0,cl456,cl4283,ap88,cl1475,cl3931	90535	2,4,3,5,1	52343
48	21.2	cl2766	cl818	cl79	cl916	ap48	cl809	cl0,cl2766,cl818,cl79,cl916,ap48,cl809	76090	6,2,1,5,4,3	57382
49	27.2	cl4592	cl1554	cl2032	cl776	cl5050		cl0,cl4592,cl1554,cl2032,cl776,cl5050	31070	2,3,4,5,1	26601
50	28.2	cl1455	cl386	cl904	cl2358	cl2678	cl2154	cl0,cl1455,cl386,cl904,cl2358,cl2678,cl2154	29969	2,4,5,6,1,3	26248
51	3.3	cl3521	cl1193	cl2828	cl2984	cl893		cl0,cl3521,cl1193,cl2828,cl2984,cl893	37331	2,5,1,3,4	29768
52	6.3	ap11	cl3508	cl1237	cl4970	cl3683	cl694	cl0,ap11,cl3508,cl1237,cl4970,cl3683,cl694	105832	2,4,1,5,3,6	89530
53	7.3	cl1438	ap77	ap108	cl1132	cl1274		cl0,cl1438,ap77,ap108,cl1132,cl1274	67537	1,4,2,5,3	51511
54	15.3	cl350	cl408	cl1274	cl1210	cl3248	cl3525	cl0,cl350,cl408,cl1274,cl1210,cl3248,cl3525	97927	6,1,5,2,3,4	57777
55	16.3	cl3508	cl1224	cl1455	cl1156	cl3256		cl0,cl3508,cl1224,cl1455,cl1156,cl3256	80956	3,4,5,1,2	64121
56	17.3	cl5000	cl3036	cl694	cl4647	ap199	cl2959	cl0,cl5000,cl3036,cl694,cl4647,ap199,cl2959	82103	6,5,1,3,4,2	58350
57	18.3	cl3310	cl942	cl409	cl5112	cl4114	cl2483	cl0,cl3310,cl942,cl409,cl5112,cl4114,cl2483	84615	5,1,3,6,4,2	68519
58	20.3	cl2853	ap239	cl2054	cl5090	ap36	cl315	cl0,cl2853,ap239,cl2054,cl5090,ap36,cl315	43272	1,3,4,2,5,6	28940
59	21.3	cl2488	cl5108	cl2478	cl751	cl96	cl33	cl0,cl2488,cl5108,cl2478,cl751,cl96,cl33	72212	6,2,4,3,1,5	65473
60	22.3	cl1208	cl1347	cl1977	cl2537	cl3003		cl0,cl1208,cl1347,cl1977,cl2537,cl3003	76609	2,4,1,5,3	44721
61	15.3	cl3702	cl4229	cl1132	cl2450	cl3622		cl0,cl3702,cl4229,cl1132,cl2450,cl3622	68228	4,2,3,5,1	38955
62								Celková ujetá vzdálenost [km]	3916.314	Délka optimalizovaných tras [km]	2986.871
63								Vzdálenost ujetá za den průměrně [km]	65.2719	Délka optimalizované denní trasy průměrně [km]	49.78118333
64											
65								Teoretické zkrácení tras celkem [km]	929.443		
66								Zkrácení denních tras [km]	15.49071667		
67								procentuální úspora	23.73259652		

Obrázek 11) Tabulka „main data“ část 2. Zdroj: vlastní zpracování

4.4 Zhodnocení výstupních dat

Po sečtení délek všech 60 zkoumaných tras s původně sestaveným pořadím návštěv členů a vysílačů autorovi práce vyšla celková ujetá vzdálenost za dobu pěti měsíců v délce 3 916,31 kilometrů. Průměrná ujetá vzdálenost během jednoho dne je tedy 65,27 kilometrů.

Po sečtení délek nově vypočtených tras pro shodné seznamy obsluhovaných vrcholů autorovi práce vyšel součet 2 986,87 kilometrů. Po vydělení počtu kilometrů počtem dní vyšla průměrná délka trasy připadající na jeden den 49,78 kilometrů.

Po odečtení celkové délky optimalizovaných tras od celkové délky původních tras vyšla celková úspora 929,44 kilometrů. Průměrná denní úspora tedy činila 15,49 kilometrů při využití řešení úlohy obchodního cestujícího. Při zapojení algoritmu pro vyřešení optimální denní trasy by došlo ke zkrácení trasy o téměř 24 %.

Jak již bylo na začátku této práce uvedeno, spolek proplácí svým technikům náklady na provoz vozidla pro servisní činnost v rámci obsluhované sítě. Podle informací, které autor práce získal při rozhovoru s vedením spolku, spolek proplácí svým technikům 10 korun na kilometr, který ujedou během své práce v rámci sítě. Tato částka zahrnuje jak příspěvek na benzín a amortizaci vozidla, tak další složky, které jsou stanoveny dohodou mezi technikem a vedením spolku. Na základě těchto čísel by implementace algoritmu pro řešení problému obchodního cestujícího mohla potenciálně firmě ušetřit až 9 294,4 korun během období pěti měsíců a průměrná denní úspora by činila 154,9 korun.

Z pohledu malé až středně velké firmy by tyto úspory mohly být velice znatelné a mohly by se použít na zlepšování služeb pro členy spolku. Z tohoto důvodu by se podle názoru autora práce měl spolek snažit o implementaci algoritmu pro optimalizaci trasy techniků a tím i o následné snížení provozních nákladů.

Zároveň by spolek mohl dosáhnout snížení provozních nákladů díky ušetření času jednotlivých techniků. Jak již bylo v práci uvedeno, tak spolek vyplácí správcům sítě hodinovou sazbu podle práce odvedené v rámci sítě, a nikoli pevný měsíční plat. Díky tomuto faktu by snížení počtu ujetých kilometrů a s tím i spjaté snížení času, který technik stráví na cestě, přispělo ke snížení provozních nákladů firmy snížením mzdových nákladů techniků.

Autor práce po rozhovoru s ostatními techniky v síti a na základě vlastních zkušeností odhaduje, že se průměrná rychlost, které technik dosahuje v autě během své servisní činnosti, pohybuje okolo 30 km/hod. Tato hodnota je poměrně nízká, ale většina servisů se provádí v Plzni a okolí v rušném denním provozu, a proto se často stává, že se technik pohybuje velice pomalu. Tento faktor bohužel nemůže spolek ovlivnit. Při průměrné rychlosti 30 km/hod by spolek zkrátil čas technika v práci až o 30,98 hodiny v průběhu sledovaných pěti měsíců.

Po rozhovoru s vedením spolku autor stanovil celkové náklady na jednu hodinu práce technika na 500 korun po započítání složek jako je hodinová mzda, zdravotní a sociální pojištění, benefity a dalších peněžních i nepeněžních náklady, a proto by hypoteticky došlo k celkové úspoře za práci techniků čítající až 15 490 korun. Autor práce neobdržel od vedení spolku přesný výčet jednotlivých nákladů na jednoho technika, ale pouze finální částku, kterou technik spolek stojí a se kterou se ve vnitřních procesech spolku kalkuluje.

Popřípadě by se dal čas uspořený efektivnějším pohybem technika investovat do více možných servisů za jeden den. I toto řešení by poté z celkového hlediska spolku mohlo přinést významné úspory.

Celkové snížení nákladů na provoz při implementaci algoritmu pro optimalizaci trasy v rámci posledních pěti měsíců by v součtu mohlo dosahovat částky až 24 784,4 koruny.

4.5 Závěr praktické části

Trasa, kterou technik objíždí v momentální organizaci práce ve firmě, je tvořena v několika krocích. Prvotní návrh trasy je tvořen operátorkou telefonní linky podle blízkosti jednotlivých vrcholů na mapě. Tento odhad je prováděn bez využití plánovacího softwaru, ale lidově řečeno „od oka“. Tímto způsobem je určena prvotní navrhovaná trasa pro technika. Tato trasa poté prochází dalšími úpravami, které mají za následek její prodloužení. Faktorů, které vedou k prodloužení trasy navržené operátorkou telefonní linky, je několik. Některé z těchto faktorů můžou být:

- Člen není ve stanovený čas k dispozici – Vždy v předchozí den probíhá kontakt všech členů a ověření, že v plánovaný čas budou dostupní a technik se u nich může zastavit na servis. Pokud nastane situace, kdy člen není dostupný v plánovaný čas servisu, tak se celá trasa změní, aby se ten den navštívili všechny plánované členové a vysílače, které byli v plánu. Tento přístup poté může působit velké rozdíly mezi optimální trasou a reálně absolvovanou trasou.
- Lidská chyba při sestavování prvotní trasy -- Nesprávné určení prvotní plánované trasy má za následek prodloužení trasy, kterou technik v daný den absolvuje.
- Nečekané komplikace na trase – může se stát, že na trase vznikne uzavírka, případně nehoda, a to má opět za následek prodloužení trasy a odklonění od

ideální trasy. S tímto typem však spolek předem počítat nemůže a tento důvod zhoršení celkové délky trasy bude hrát roli vždy.

4.5.1 Navrhovaná řešení

Spolek má dle autora práce dvě možnosti, jak přistupovat k problematice prodlužování tras popsané v předchozím odstavci. Zde autor uvádí možná řešení problému:

- zpoplatnění individuálních požadavků na čas servisu
- změnit systém sestavování tras.

4.5.2 Zpoplatňování požadavků na individuální čas servisu

Z praxe a z rozhovorů se správci sítě a s vedením spolku, autor vyrozuměl, že průměrně zhruba dva členové denně mají speciální přání na čas a mohou jen ve specifický čas. Proto se musí kvůli těmto členům měnit trasa. To vede k prodloužení trasy a k dalším nákladům z pohledu spolku.

Jako řešení by autor práce spolku doporučil zpoplatnit členům požadavek na individuální čas přiměřenou peněžní částkou. Při průměrných dvou žádostech denně a při vzorku šedesáti dnů, pokud by spolek chtěl pokrýt pouze náklady způsobené najetými kilometry navíc, tak by spolek musel zpoplatnit žádost o výběr individuální času pro servis částkou 77,45 koruny. Tato částka vyšla po vydělení nákladů vzniklých v souvislosti s najetými kilometry navíc oproti optimální trase počtem individuálních požadavků za 60 dní, kterých bylo zhruba 120. Zpoplatnění tohoto požadavku touto částkou by spolku v horizontu posledních pěti měsíců přineslo 9 294 korun. Podle názoru autora práce by si však spolek mohl dovolit zpoplatnit výběr individuálního času na servis částkou 100 korun což by spolku přineslo zhruba 12 000 korun za dobu posledních pěti měsíců. Pro členy spolku by podle autora této práce tato částka nepředstavovala velký problém.

Za předpokladu, že by si spolek chtěl nechat tímto způsobem hradit i náklady, které vznikají tím, že spolek navíc správcům proplácí i čas strávený v autě během přesunu mezi servisy, měl by spolek zpoplatnit členům možnost vlastního výběru času částkou 206,53 korun. Tato částka opět vyšla jako podíl součtu nákladů vzniklých ujetými kilometry navíc a nákladů spojených s časem technika počtem požadavků za 60 dní. Spolek by tím tedy pokryl náklady vzniklé kilometry najetými navíc a zároveň i čas strávený technikem navíc v autě kvůli delším trasám. Autor práce by však doporučil spolku poplatek

zaokrouhlit na částku 250 korun, kterou by spolek pokryl všechny náklady, které vznikly navíc.

Autor práce spolku by doporučil tento poplatek vždy zvýšit. U pokrývání pouze nákladů vzniklých najetými kilometry navíc autor doporučuje částku stanovit na 100 korun a v případě pokrývání vzdálenosti i času, který technik strávil za volantem by autor doporučil stanovit částku 250 korun. Tento přístup autor doporučuje z důvodu, aby se spolek mohl alespoň z části bránit proti vzniku neočekávaných komplikací při pohybu techniků v sítí a s tím spojeným nákladům. Navíc podle vlastních zkušeností autora většina členů preferuje platby, které jsou zaokrouhleny na celé sto koruny nebo padesáti koruny, a mnohdy i sami členové spolku mají tendenci zaokrouhlovat částku směrem nahoru, aby se vyhnuli zbytečné manipulaci s mincemi malých hodnot. Další možností z tohoto důvodu by bylo tuto částku stanovit na hodnotě 200 korun, což by spolku dovolilo hradit z velké části náklady, které spolku vznikají a zároveň by spolek provedl zaokrouhlení na celé sto koruny, což by mohlo připadat sympatičtější členům spolku. Spolek by však tímto ztratil možnost případně pokrýt náklady vzniklé z důvodu neočekávaných problémů.

4.5.3 Rozdílný systém plánování tras

Autor práce navrhnul spolku plánovat trasy vždy na dva nebo více dnů dopředu. Spolek vytváří trasu vždy ze členů, které jsou ve frontě nejdéle, avšak téměř nikdy se nestane, že by fronta na servis byla prázdná. Pokud by tedy spolek začal plánovat trasy na více dnů dopředu, tak by to spolku dovolilo sestavit trasy tak, aby si všichni členové mohli určit svůj individuální čas a spolek by mohl plánovat optimální trasy. Tento přístup by vedl k úspoře nákladů a spolek by mohl minimalizovat vliv určování individuálních časů na provedení servisů ze strany členů.

Spolek by tímto přístupem neeliminoval všechny náklady, které spolku vznikají, ale pouze by omezil tvorbu nákladů navíc. Toto řešení by autor práce shledával jako vhodné, ale rozpracování by bylo již nad rámec této práce. Navíc se autor práce obává, že by cena implementace takového systému, který by byl na navržení složitý, přesáhla úspory, které by spolku program přinesl. Proto toto řešení autor práce spolku navrhl jako možné, ale nemůže přínos tohoto řešení podložit žádnými konkrétními čísly.

5 Implementace navrženého řešení

Autor práce navrhl vedení spolku dvě různé implementace řešení problému. Spolek si může vybrat buď implementaci programového řešení a snažit se tím minimalizovat přebytečné náklady na provoz vozidel a s tím i mzdové náklady na techniky, kteří tyto kilometry stráví za volantem. Nebo se spolek může rozhodnout pro přístup, že momentální systém plánování jednotlivých tras ponechá a začne účtovat členům menší příplatek za určení vlastního času na servis. Potažmo se spolek může ještě rozhodnout pro kombinaci obou řešení.

5.1 Implementace programového řešení

Implementace programového řešení pro výpočet problému obchodního cestujícího se vztahuje k procesu nasazení nového softwarového řešení v organizaci. Během implementace je nutné zajistit, aby byly splněny požadavky projektu a že nové řešení je bez chyb a v souladu s potřebami organizace. Implementace softwaru může být složitý proces a často vyžaduje úzkou spolupráci mezi vývojáři a organizací.

Pokud by spolek chtěl implementovat programové řešení, které by spolku pomáhalo optimalizovat trasy vždy pouze na den dopředu, tak by to podle názoru autora nemělo představovat pro spolek problém.

Program, který autor napsal v programovacím jazyce Java je velice jednoduchý a ze strany spolku by se museli udělat jen minimální změny. Jak již autor uváděl, upravit tento program, aby vypisoval i informace jako na příklad telefonní číslo a emailovou adresu by bylo jednoduché. Autor si data stahoval na svůj osobní počítač a pokud byl do systému zadán nový člen, tak autorova databáze není přímo napojena na databázi spolku a proto by se neaktualizovala podle této změny. Tento problém by byl dle názoru autora největší překážkou u implementace navrženého řešení. I přes to, že tento problém je nad rámec schopností autora této práce, tak ve spolku jsou zaměstnáni technici, kteří by toto byli schopni propojit a proto by to pro spolek nemělo představovat velký problém.

5.2 Účtování poplatků

Druhým způsobem, jakým se spolek může snažit eliminovat přebytečné náklady na pohyb technika, je nikoli snaha o to tyto náklady eliminovat, ale pouze přenést vzniklé náklady na koncového uživatele kvůli kterému musí technik jet delší trasu, než je nezbytně nutné.

Jak již autor práce stanovil v předchozí části této práce, aby spolek eliminoval veškeré náklady spojené s prodlužováním tras kvůli přání členů, tak by poplatek za vlastní čas činil minimálně 206,53 koruny. Jak již autor práce uváděl, tak by však doporučil cenu příplatku stanovit na 250 korun.

Jak již bylo v této práci uvedeno, spolek za své služby v mnoha případech nevybírá poplatek za práci. Z vlastní zkušenosti a z rozhovorů se členy jiných konkurenčních spolků v Plzni a okolí stojí návštěva technika z jakéhokoli důvodu mezi 300 a 600 korunami. Pokud by se tedy spolek rozhodl zavést tento autorem navrhovaný poplatek za možnost určení vlastního času pro návštěvu technika, tak by člen spolku, který tuto možnost využije, platil stále méně než za servis u většiny konkurentů na trhu. Tato cena by však mohla přesvědčit některé členy, aby se přizpůsobili času stanovenému spolkem a tím by se také přispělo ke snižování nákladů na provoz.

Někteří členové by však mohli tento poplatek shledávat jako příliš a mohla by hrozit ztráta členů. To by poté vedlo k finančnímu poškození spolku v delším časovém úseku.

5.3 Další možná řešení

Problém má i další možnosti řešení, než jen autorem navrhované řešení. Další možnosti řešení, která jsou možná jsou například:

- Zpoplatnění všech servisů bez rozdílu – Při zavedení poplatku za každý servis pro všechny členy spolku, by spolek mohl financovat kilometry, které technici urazí navíc z důvodu určování individuálních časů. Navíc by tento přístup přinesl spolku peníze, které by spolek mohl investovat do budování sítě pro své členy. Avšak z důvodu, že rozhodnutí, že si členové své nutné servisy nemusí hradit je jedním z hlavních důvodů spokojenosti mnoha členů a jednou ze základních politik spolku by toto řešení autor této práce nedoporučoval.
- Implementace komplexního automatizovaného systému. Pokud by se spolek rozhodl implementovat systém, který by dovoloval organizování členů podle časových možností specifikovaných jednotlivými členy a plánovat optimální trasy podle těchto časových možností v horizontu několika dní dopředu tak, aby tyto plánované cesty vždycky dosahovaly určitého stupně optimalizace, by mohl spolek alespoň z části eliminovat náklady vzniklé na servis tímto způsobem. Navržení a implementace takového řešení je však složitějším problémem a zřejmě

by vyžadovalo i spolupráci s dalšími společnostmi působícími v tomto oboru. Toto řešení by spolku dovolovalo snížit náklady spojené se servisy v síti, ale zase by spolku způsobilo poměrně velké prvotní náklady na zavedení takového systému včetně nákladů na údržbu tohoto systému. Je však otázkou, jestli je počet členů v síti a počet servisů, které připadají na jeden den dostatečný, aby mohl ospravedlnit náklady vzniklé implementací takového systému.

Závěr

Tato práce se věnuje problematice řešení konkrétního ekonomického problému, se kterým se potýká spolek poskytující služby související s internetovým připojením svých členů. Jde o optimalizaci pohybu technika v síti a tím zmenšení nákladů nutných na provádění servisních prací v síti. Autor této práce dospěl k závěru, že nejlepší možné řešení pro tento konkrétní spolek je kombinace řešení problému obchodního cestujícího za využití poměrně jednoduchého programového řešení napsaného v jazyce Java a zavedení poplatku, který by si hradil člen, pokud by požadoval individuálně určený čas návštěvy technika navzdory optimálně stanovené trase. Řešení navrhované autorem je poměrně jednoduché, přičemž objem servisů v této konkrétní síti pravděpodobně není dostatečně velký na to, aby si spolek mohl dovolit nechat navrhnout a implementovat složité programové řešení, které by spolku dovolilo kombinovat individuální požadavky členů s optimálním sestavováním tras v horizontu více dní předem. Autor zároveň zmínil i možnost trasy vůbec neoptimalizovat, a naopak zavést poplatek pro všechny servisy stejný. Takové řešení by však mohlo způsobit pokles počtu členů, a proto se toto řešení spolku nedoporučuje.

Seznam použitých zdrojů

Bělohlávek R., Vychodil V. (2006). Diskrétní Matematika Pro Informatiky. Univerzita Palackého [online]. Dostupné 13.4.2023 z: http://phoenix.inf.upol.cz/esf/ucebni/DM1.pdf a http://phoenix.inf.upol.cz/esf/ucebni/DM2.pdf

Bezdrátová komunikace v infrastruktuře internetových sítí (n.d.). Dostupné 17.4.2023 z: https://ujezd.net/bezdratova-komunikace-v-infrastrukture-internetovych-siti

History of Java (n.d.). Dostupné 16.4.2023 z: https://www.javatpoint.com/history-of-java

JetBrains (n.d.). Dostupné 17.4.2023 z: https://www.jetbrains.com/

Kaliaganov, M. (2021). *Optimalizace údržby pozemních komunikací* [Bakalářská práce, škoda Auto Vysoká Škola]. Dostupné 16.4.2023 z: https://theses.cz/id/eu3sqy/zaverecna_prace.pdf

Krátky, R. (2020). *Hamiltonovské grafy* [Diplomová práce, Univerzita Hradec Králové]. Dostupné 17.4.2023 z: https://theses.cz/id/wy704j/STAG92896.pdf

Luner, P. (n.d.). *Jemný úvod do genetických algoritm*ů. Dostupné 17.4.2023 z: https://cgg.mff.cuni.cz/~pepca/prg022/luner.html

MATAI Rajesh, SURYA Singh a MURARI Lal Mittal (2010). *Travelling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches*. [online]. ISBN DOI: 10.5772/12909. Dostupné 17.4.2023 z: https://www.intechopen.com/chapters/12736

Dijkstrův algoritmus (n.d.). Dostupné 17.4.2023 z: https://portal.matematickabiologie.cz/index.php?pg=zaklady-informatiky-pro-biology--teoreticke-zaklady-informatiky--teorie-grafu--optimalizacni-ulohy-nad-grafy--dijkstruv-algoritmus

Moscato, P., Cotta, C., & Mendes, A. (2004). *Memetic Algorithms In New Optimization Techniques in Engineering* (pp. 53–85). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-39930-8_3

O nás (n.d.). Dostupné 16.4.2023 z: https://boknet.cz/o-nas

Services (n.d.). Dostupné 16.4.2023 z: https://openrouteservice.org/services/

Smith, Clint P.E., and Daniel Collins. (2014). *Wireless Networks: Design and Integration for LTE, EVDO, HSPA, and WiMAX.* 3rd ed. New York: McGraw-Hill Education. https://www.accessengineeringlibrary.com/content/book/9780071819831

Yu, V. F., Susanto, H., Jodiawan, P., Ho, T.-W., Lin, S.-W., & Huang, Y.-T. (2022). A Simulated Annealing Algorithm for the Vehicle Routing Problem With Parcel Lockers. IEEE Access, 10, 20764–20782. https://doi.org/10.1109/access.2022.3152062

What we do. (n.d.). Dostupné 17.4.2023 z: https://heigit.org/what-we-do/

Wired Communication Media (n.d.). Dostupné 16.4.2023 z: https://www.geeksforgeeks.org/wired-communication-media/

Co je to API. (n.d.). Dostupné 16.4.2023 z: https://www.rascasone.com/cs/blog/co-je-api

Waterfall Model. (n.d.). Dostupné 16.4.2023 z: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

Seznam obrázků

Seznam příloh

Příloha A: Zdrojový kód

Třída APP:

```
package cz.zcu.sip;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class App {

    public List<Node> getNodesFromFile(File file, Set<String> request)
{
        if (!file.exists()) {
            System.err.println("File " + file.getAbsolutePath() + "
does not exist.");
            System.exit(-1);
        }

        List<Node> nodes = new ArrayList<>();

        try (BufferedReader bufferedReader = new BufferedReader(new
FileReader(file))) {
            String line;
            while ((line = bufferedReader.readLine()) != null) {
                Node n = new Node(line);
                if (request.contains(n.toInputString())) {
                    nodes.add(n);
                    request.remove(n.toInputString());
                }
            }

        } catch (IOException e) {
            e.printStackTrace();
            System.exit(-1);
        }
        if (!request.isEmpty()) {
            System.err.println("Failed to find GPS for: ");
            for (String s : request) {
                System.err.println(s);
            }
        }

        return nodes;
    }


    public static void getDistance(Edge edge) throws
InterruptedException {
        String f = edge.getFrom().getGPS();
        String t = edge.getTo().getGPS();
        Thread.sleep(1501);
        try {
//            URL url = new
URL("https://api.openrouteservice.org/v2/directions-driving-car?" +
//
```

```
"api_key=5b3ce3597851110001cf62485dbeed399ffd4dfdb63e2b985aef90dc&star
t=" + f + "&end=" + t);
            URL url = new
URL("https://api.openrouteservice.org/v2/directions/driving-
car?api_key=5b3ce3597851110001cf62485dbeed399ffd4dfdb63e2b985aef90dc&s
tart=" + f + "&end=" + t);
            HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
            conn.setRequestMethod("GET");
            conn.setRequestProperty("Accept", "application/json,
application/geo+json, application/gpx+xml, img/png; charset=utf-8");

            if (conn.getResponseCode() != 200) {
                throw new RuntimeException("Failed : HTTP error code :
" + conn.getResponseCode());
            }
            StringBuilder response = new StringBuilder();
            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(conn.getInputStream()))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }
            }
            conn.disconnect();
            String distanceString =
extractProperty(response.toString(), "distance");
            if (distanceString != null) {
                distanceString = distanceString.substring(0,
distanceString.indexOf("."));
                edge.setDistance(Integer.parseInt(distanceString));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }


    public static Edge[][] createDistanceMatrix(List<Node> nodes)
throws InterruptedException {
        int n = nodes.size();
        Edge[][] dist = new Edge[n][n];
        for (int i = 0; i < n; i++) {
            Node from = nodes.get(i);
            for (int j = 0; j < n; j++) {
                Node to = nodes.get(j);
                Edge edge = new Edge(from, to);
                if (i == j) {
                    edge.setDistance(0);
                    dist[i][j] = edge;
                    continue;
                }
                getDistance(edge);
                dist[i][j] = edge;
                Edge opposite = new Edge(to, from);
                getDistance(opposite);
                dist[j][i] = opposite;
            }
        }
        return dist;
```

```
    }

    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String line = scanner.nextLine();
        String[] words = line.split(",");
        App app = new App();
        File file = new
File("src/main/resources/v_address_point.csv");
        Set<String> wordsSet = new HashSet<>();
        Collections.addAll(wordsSet, words);
        List<Node> nodesFromFile = app.getNodesFromFile(file,
wordsSet);
        System.out.println(nodesFromFile);

        Edge[][] matrix = createDistanceMatrix(nodesFromFile);
        List<Node> traveledNodes = new ArrayList<>();
        long distance = 0;
        for (int i = 0; i < matrix.length; i++) {
            int pointer = i+1;
            if(pointer>=matrix.length){
                pointer = 0;
            }
            Edge[] edges = matrix[i];
            Edge e= edges[pointer];
            traveledNodes.add(e.getFrom());
            distance+=e.getDistance();
            if(i==matrix.length-1){
                traveledNodes.add(e.getTo());
            }
        }

        TSP.TspResult tsp = TSP.getTsp(matrix);
        for (Node node : tsp.path) {
            System.out.println(node);

        }


        System.out.println("distance : " + tsp.distance);

        for (Node traveledNode : traveledNodes) {
            System.out.println(traveledNode);
        }

        System.out.println("traveled distance: "+ distance);

    }

    public static String extractProperty(String json, String
propertyName) {
        Pattern pattern = Pattern.compile("\"" + propertyName +
"\":\\s*([^,\\}\\s]+)");
        Matcher matcher = pattern.matcher(json);
        if (matcher.find()) {
            return matcher.group(1);
        }
        return null;
```

```
    }
```

Třída TSP.java:

```
package cz.zcu.sip;

import java.util.*;


public class TSP {

    public static TspResult getTsp(Edge[][] matrix) {
        int n = matrix.length;

        int[] path = new int[n - 1];
        for (int i = 1; i < n; i++) {
            path[i - 1] = i;
        }

        long minDist = Long.MAX_VALUE;
        int[] minPath = new int[n - 1];

        do {
            long currDist = matrix[0][path[0]].getDistance();
            for (int i = 0; i < n - 2; i++) {
                currDist += matrix[path[i]][path[i +
1]].getDistance();
            }
            currDist += matrix[path[n - 2]][0].getDistance();

            // Handle asymmetric distances
            if (matrix[path[n - 2]][0].getDistance() !=
matrix[0][path[n - 2]].getDistance()) {
                currDist += matrix[0][path[n - 2]].getDistance() -
matrix[path[n - 2]][0].getDistance();
            }

            if (currDist < minDist) {
                minDist = currDist;
                System.arraycopy(path, 0, minPath, 0, n - 1);
            }
        } while (nextPermutation(path));


        List<Node> order = new ArrayList<>();
        order.add(matrix[0][0].getFrom());
        for (int i : minPath) {
            order.add(matrix[i][i].getFrom());
        }
        order.add(matrix[0][0].getFrom());
        return new TspResult(order, minDist);

    }


    public static boolean nextPermutation(int[] array) {
        int i = array.length - 2;
        while (i >= 0 && array[i] >= array[i + 1]) {
            i--;
        }
```

```
        if (i < 0) {
            return false;
        }
        int j = array.length - 1;
        while (array[j] <= array[i]) {
            j--;
        }
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
        reverse(array, i + 1, array.length - 1);
        return true;
    }

    public static void reverse(int[] array, int i, int j) {
        while (i < j) {
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
            i++;
            j--;
        }
    }


    public static long[][] createDistanceMatrix(Edge[][] edges) {
        int n = edges.length;
        long[][] dist = new long[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                Edge edge = edges[i][j];
                if (edge != null) {
                    dist[i][j] = edge.getDistance();
                } else {
                    dist[i][j] = Long.MAX_VALUE;
                }
            }
        }
        return dist;
    }

    public static class TspResult {
        public Long distance;
        public List <Node> path;

        public TspResult(List<Node> order, long minDist) {
            path = order;
            distance=minDist;

        }
    }
}
```

Třída Edge.java:

```
package cz.zcu.sip;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Objects;

public class Edge {
    private final Node from;
    private final Node to;
    private Long distance;
    public Edge(Node from, Node to) {
        this.from = from;
        this.to = to;
    }

    public Long getDistance() {
        return distance;
    }

    public Node getFrom() {
        return from;
    }

    public Node getTo() {
        return to;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Edge edge = (Edge) o;
        return from.equals(edge.from) && to.equals(edge.to);
    }

    @Override
    public int hashCode() {
        return Objects.hash(from, to);
    }


    public void setDistance(long distance) {
        this.distance = distance;
    }
}
```

Třída Node.java:

```
package cz.zcu.sip;


import java.util.Objects;


public class Node {

    public String toInputString () {

        if(this.GPS.contains("NULL")){
            return "";
        }

        String result = "";
        if (isAp) {
            result += "ap";
        } else {
            result += "cl";
        }
        result += id;



        return result;
    }

    public Node(String line) {
        line = line.replaceAll("\"", "");
        String[] words = line.split(",");
        this.isAp = words[0].equals("ap");

        if (this.isAp) {
            this.id= Integer.valueOf(words[2]);
        } else {
            this.id = Integer.valueOf(words[1]);
        }


        this.GPS = words[5]+ "," +words[4];
        this.name = words[3];

    }

    private String name;

    private String GPS;

    private Integer id;

    public boolean isAp(boolean b) {
        return isAp;
    }

    public void setAp(boolean ap) {
        isAp = ap;
    }

    @Override
```

```
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Node node = (Node) o;
        return isAp == node.isAp && name.equals(node.name) &&
id.equals(node.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, id, isAp);
    }

    @Override
    public String toString() {
        return "Node{" +
                "name='" + name + '\'' +
                ", GPS='" + flipGPS() + '\'' +
                ", id=" + id +
                ", isAp=" + isAp +
                '}';
    }

        private String flipGPS(){
            String[] split = GPS.split(",");
            return split[1]+","+split[0];

        }
    private boolean isAp;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getGPS() {
        return GPS;
    }

    public void setGPS(String GPS) {
        this.GPS = GPS;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

}
```

Třída Repository.java:

```
package cz.zcu.sip;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;
import org.hibernate.service.ServiceRegistry;

public class Repository {
    private final SessionFactory sessionFactory;

    public Repository(Configuration configuration) {
        ServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder().applySettings(configuration.getProper
ties()).build();
        sessionFactory =
configuration.buildSessionFactory(serviceRegistry);
    }

    public void persist(Node n) {
        try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();
            session.persist(n);
            transaction.commit();
        }
    }

    public Node getById(Integer id) {
        try (Session session = sessionFactory.openSession()) {
            Query<Node> query = session.createQuery("select n from
Node n where n.vs=:id", Node.class);
            query.setParameter("id", id);
            return query.getResultStream().findFirst().orElse(null);

        }


    }

    public void delete(Integer id) {
        try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();
            Query query = session.createQuery("delete from Node where
vs= :id");
            query.setParameter("id", id);
            query.executeUpdate();
            transaction.commit();
        }
    }
}
```

Abstrakt

APA 7: Šíp, O. (2023). *Návrh na zlepšení logistických činností v podniku zaměřeném na poskytováním internetových služeb* [Bakalářská práce, Západočeská univerzita v Plzni].

Klíčová slova: optimalizace, teorie grafů, problém obchodního cestujícího

Tato práce se zabývá optimalizací pohybu správců mezi členy vybraného spolku po pozemních komunikacích. Efektivita pohybu je řešena pomocí přístupu hrubé síly na řešení problému obchodního cestujícího. Pro řešení minimální teoretické trasy si autor napsal program v programovacím jazyce Java. Data se kterými se v této bakalářské práci počítá autor získal po udělení přístupu do vnitřního systému vybraného ekonomického subjektu. Teoretický ekonomický přínos zavedení takového optimalizačního algoritmu je vypočítaný porovnáním délek skutečně absolvovaných tras s optimalizovaných tras. Na konec autor navrhl vybranému ekonomickému subjektu možné řešení. Kromě řešení, které by autor preferoval, byly spolku navržená i jiná řešení.

Abstract

APA 7: Šíp, O. (2023). *Proposal for improving logistical operations inside business focused on providing internet service* [Bachelor Thesis, University of West Bohemia].

Klíčová slova: optimization, graph theory, traveling salesman problem

This work deals with optimizing the movement of technicians between members of a selected economic entity along roads. The efficiency of movement is addressed using the brute force approach to solving the traveling salesman problem. To solve the minimum theoretical route, the author wrote a program in the Java programming language. The data used in this bachelor's thesis was obtained after being granted access to the internal system of the selected economic entity. The theoretical economic benefit of introducing such optimization algorithm is calculated by comparing the lengths of the actual routes taken with the optimized routes. Finally, the author proposed a possible solution to the selected economic entity. In addition to the solution that the author would prefer, other solutions were also proposed to the economic entity.