VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

OPTIMÁLNÍ PLÁNOVÁNÍ TRASY PRO ELEKTROMOBILY
OPTIMAL PATH PLANNING FOR ELECTRIC VEHICLES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Filip Horák**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jakub Kůdela, Ph.D.**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Filip Horák**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Jakub Kůdela, Ph.D.**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Optimální plánování trasy pro elektromobily

Stručná charakteristika problematiky úkolu:

Diplomová práce se bude zabývat plánováním optimální trasy pro elektromobily, s přihlédnutím na časovou a cenovou náročnost zvolené trasy a výběr dobíjecích stanic.

Cíle diplomové práce:

Popsat problém plánování trasy.
Provést rešerši aktuálně používaných metod pro tento problém.
Pro vybranou metodu vytvořit software implementaci.

Seznam doporučené literatury:

ARSLAN, O., YILDIZ, B., KARASAN, O. E. Minimum cost path problem for Plug-in Hybrid Electric Vehicles. Transportation Research Part E. 2015, 123-141.

SWEDA, T. M., DOLINSKAYA, I. S., KLABJAN D. Adaptive Routing and Recharging Policies for Electric Vehicles. Transportation Science. 2017, 1-23.

KHULLER, S., MALEKIAN, A., MESTRE, J. To fill or not to fill: The gas station problem. ACM Transactions on Algorithms. 2011, Article No.: 36.

ABSTRAKT

Tato diplomová práce se zabývá plánováním optimální trasy pro elektromobily. V první, teoretické části této práce probíhá uvedení problematiky a popis několika heuristických metod, které byly pro řešení optimalizačních úloh použity. Praktická část práce se zabývá software implementací popsaných metod. Na závěr je provedeno několik experimentů a porovnání výsledků jednotlivých metod.

ABSTRACT

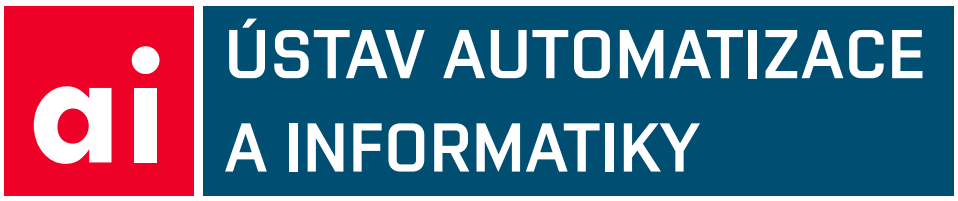
This master's thesis is about optimal route planning for electric vehicles. The first, theoretical part of this work introduces the issue and describes several heuristic methods that have been used to address optimization tasks. A practical part of the thesis is based around software implementation of the methods described earlier. Finally, several experiments and comparisons of obtained results are carried out.

KLÍČOVÁ SLOVA

Dijkstrův algoritmus, A*, dynamické programování, celočíselné programování, plánování cesty, elektromobil, plug-in hybrid.

KEYWORDS

Dijkstra's algorithm, A*, dynamic programming, integer programming, route planning, electric vehicle, plug-in hybrid vehicle.



2021

BIBLIOGRAFICKÁ CITACE

HORÁK, Filip. *Optimální plánování trasy pro elektromobily.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2021, 69 s. Diplomová práce. Vedoucí práce: Ing. Jakub Kůdela, Ph.D.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 21. 5. 2021 .....

Filip Horák

PODĚKOVÁNÍ

Srdečně děkuji vedoucímu této práce Ing. Jakubovi Kůdelovi Ph.D. za jeho odborné vedení, čas a cenné rady, které mi během psaní práce neváhal poskytnout. Speciální díky pak věnuji rodičům, kteří mi během studia nepřestávali býti oporou. V neposlední řadě bych rád poděkoval také přítelkyni a blízkým přátelům, kteří mě během studia doprovázeli.

OBSAH

1 ÚVOD

Během posledních pár let se počet registrovaných vozidel poháněných (ať už kompletně nebo pouze z části) elektrickou energií každým dnem zvyšuje. S rostoucím počtem vozidel narůstá také potřeba použití dobíjecích stanic. Jednou z řady překážek, které uživatelé vozidel s elektrickým pohonem musí čelit, je plánování trasy tak, aby v blízkosti cesty delší vzdálenosti byly přístupné dobíjecí stanice. Další nepříjemnosti mohou nastat, pokud řidič přijede k dobíjecí stanici a plány mu zkazí naplněná kapacita vozidel dobíjecí stanice. V případě použití plug-in hybrid vozidla si řidič nemusí dělat starosti, pokud mu dojde baterie a v bezprostřední blízkosti se nenachází žádná dobíjecí stanice, jelikož může jet dále na spotřebu paliva. Aby však mohl plně využít potenciál vozidla, je vhodné baterii pravidelně dobíjet.

Pro problematiku plánování cesty vozidel s přihlédnutím na doplňování pohonných hmot v průběhu cesty byla postupem let publikována řada přístupů využívajících různé metody řešení. Jednou z hojně používaných metod řešení takové úlohy je použití dynamického programování. Aplikaci dynamického programování lze naleznout v článku *To Fill or Not to Fill: The Gas Station Problem* [14], kde je sice model vozidla poháněný palivem, avšak obdobný přístup je možné použít pro elektromobil. Další použití dynamického programování za účelem optimalizace plánování trasy modelu plug-in hybrid vozidla je obsaženo v článku *Minimum cost path problem for Plug-in Hybrid Electric Vehicles* [3], kde je řešena deterministická úloha nalezení cesty mezi dvěma zadanými body a optimalizace dobíjení a dotankování na zvolené trase. V článku *Adaptive Routing and Recharging Policies for Electric Vehicles* [21] je naopak popsáno řešení stochastické úlohy, kde se přihlíží na neznámou obsazenost dobíjecích stanic a časy čekání na uvolnění kapacity stanice. Dalším možností, jak je k řešení problematiky doplnění pohonných hmot plug-in hybrid automobilu možné přistupovat, je použití složitějšího matematického modelu, přihlížejícího k cenám elektřiny a paliva v dostupných stanicích. Použití takového modelu je popsáno v článku *Modeling the Impacts of Electricity Tariffs on Plug-In Hybrid Electric Vehicle Charging, Costs, and Emissions* [20] a také v již dříve uvedeném článku [3]. K úloze je možné také přistoupit způsobem uvedeným v publikaci *Electric Vehicle Charging Warning and Path Planning Method Based on Spark* [9], kde byl pro optimalizaci cesty a dobíjení baterie elektromobilu použit Dijkstrův algoritmus v kombinaci s paralelním zpracováním dat prostřednictvím frameworku `Apache Spark`. Tato metoda řešení přihlíží ke stavu vozovky a jejím úsekům. Umožňuje přizpůsobovat spotřebu vozidla v závislosti na typu vozovky, jestli se jedná o hlavní silnici nebo vedlejší, dálnici či křižovatku.

Cílem této práce, bylo popsat problematiku hledání optimální cesty elektřinou poháněných vozidel a následně vytvořit software implementaci pro tyto úlohy.

V této práci byly řešeny deterministické úlohy, jejichž podstatou je nalezení trasy mezi dvěma body v daném prostředí a minimalizace nákladů na provedení této cesty.

Nejprve je čtenář seznámen s teorií týkající se grafů a způsoby jejich prohledávání a optimalizačními metodami. Dále je v práci popsána aplikace Dijkstrova algoritmu a algoritmu A* pro nalezení přípustné cesty v dopravní síti s dobíjecími stanicemi a následná optimalizace hodnot nabíjení na získané trase za pomocí dynamického programování. Posléze jsou v práci popsány a prakticky použity také metody představené článkem [3]. Všechny tyto metody, včetně vytváření simulačního prostředí, byly zakomponovány do naprogramovaného uživatelského rozhraní, jehož funkce je v práci rovněž vysvětlena. Na závěr je v diplomové práci demonstrováno použití uvedených metod na několika různých grafech a modelech automobilů a jejich vzájemné porovnání.

2 PLÁNOVÁNÍ CESTY A OPTIMALIZACE

Pro řešení problematiky plánování cesty je nutné nejprve formulovat prostředí, ve kterém se cesta bude vyhledávat. Vhodnou strukturou, jež se hojně používá při řešení úloh spojených s plánováním a optimalizací cesty je graf. Graf je možné prohledávat různými algoritmy a hledat v něm cesty mezi jednotlivými uzly.

2.1 Graf

Graf je uspořádaná dvojice $G = (N, A)$, kde N vyjadřuje neprázdnou konečnou množinu uzlů grafu a A udává konečnou množinu hran mezi uzly grafu. Prvky množiny A mohou mít různý tvar dle orientace hran. Uspořádaná dvojice (i, j) udává orientovanou hranu mířící z uzlu i do uzlu j. Naopak dva prvky v množinových závorkách $\{i, j\}$ označují hranu, jež nemá daný směr.

Jako sousedící se označují takové dva uzly grafu, které jsou navzájem propojeny hranou. Cestou se nazývá konečná posloupnost sousedících uzlů mezi dvěma zvolenými uzly.

K hranám či uzlům grafu lze přidávat číselné nebo jiné hodnoty, které mohou reprezentovat např. doby trvání, délku úseku, propustnosti potrubí, pravděpodobnosti událostí apod.

Text a obrázky této podkapitoly byly čerpány z publikací [23, 26].

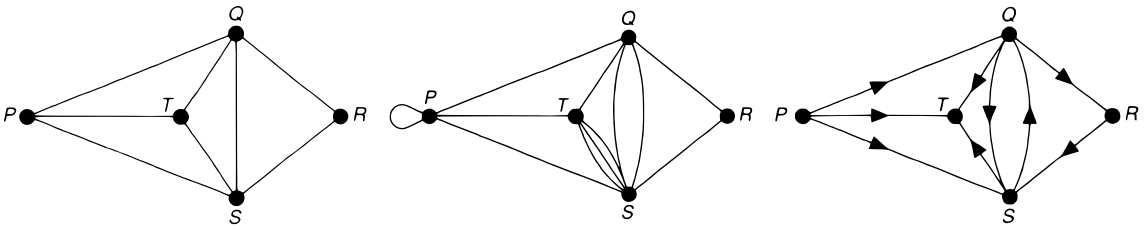
2.1.1 Vlastnosti grafu

Graf obsahuje násobné hrany, pokud se v něm vyskytují dvě a více shodně vedoucích hran mezi jednou dvojicí uzlů. Takový graf nese pojmenování multigraf (viz obr. 1b). Jestliže se v grafu nachází hrana, jež vychází ze stejného uzlu jako do kterého vchází, jedná se o smyčku. Smyčka je na obr. 1b vyobrazena jako součást multigrafu, ale výskyt smyček není na multigrafy omezen. Graf neobsahující násobné hrany nebo smyčky se nazývá prostý graf a k vidění se nachází na obr. 1a.

Graf se dále rozlišuje dle orientace hran. Jako orientovaný se označuje takový graf, jehož hrany vedou pouze v jednom směru a lze jej vidět na obr. 1c. Cesta v orientovaném grafu, jejíž počáteční uzel je zároveň koncový, se nazývá cyklus. Na obr. 1c je možné tento cyklus vidět mezi uzly $Q \to S \to Q$ a $Q \to R \to S \to Q$.

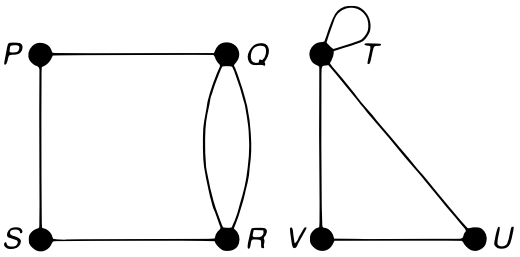
Všechny příklady grafů uvedené na obr. 1 tvoří jeden celek a mezi každými dvěma uzly lze nalézt alespoň jednu cestu, která je propojuje. Takové grafy jsou nazývané souvislé. V případě, že je graf tvořený z více celků, jež nejsou vzájemně propojeny hranami, jedná se o graf nesouvislý (viz obr. 2).

Dva grafy jsou izomorfní, jestliže se liší pouze nakreslením nebo pojmenováním uzlů a hran. Na obr. 3 je ukázka grafu, který je izomorfní s grafem na obr. 1a.

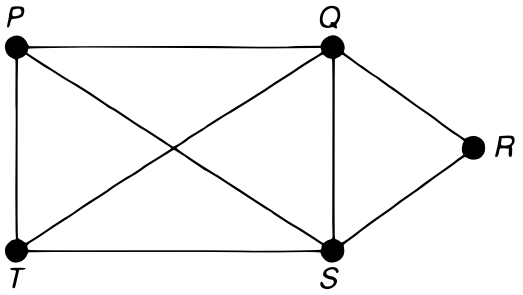


(a) Prostý graf (b) Multigraf se smyčkou (c) Orientovaný prostý graf

Obr. 1: Ukázka grafů s různými hranami [26]



Obr. 2: Nesouvislý graf [26]



Obr. 3: Izomorfní graf ke grafu na obr. 1a [26]

Jestliže ke grafu existuje rovinné nakreslení, jedná se o rovinný neboli planární graf. Rovinné nakreslení je takové, kde je graf nakreslený bez křížení hran. Graf na obr. 3 je tedy také rovinný, jelikož je možné jej překreslit do podoby na obr. 1a.

V této práci se pracuje zejména s konečnými neorientovanými grafy bez násobných hran a smyček a pokud není specifikováno jinak, označením graf se rozumí právě tento graf.

2.2 Metody prohledávání grafu

Metody prohledávání grafů lze dle využití znalostí o úloze dělit na úlohy neinformované, které znalosti o úloze nevyužívají (např. metoda prohledávání do šířky, metoda prohledávání do hloubky, apod.) a informované, jež se řídí dostupnými znalostmi o problému (např. best-first search, algoritmus A*, apod.) [6]. Pro hledání nejkratší cesty se dále používá Dijkstrův algoritmus nebo Floyd-Warshallův algoritmus [23].

V této práci byly použity algoritmy A*, Dijkstrův a Floyd-Warshallův, které jsou blíže popsány níže.

2.2.1 Algoritmus A*

Algoritmus A* je grafový algoritmus sloužící k nalezení nejkratší cesty v grafu vedoucí mezi dvěma zadanými uzly. Poprvé jej představili P. Hart, N. Nilsson a B. Ra-

phael ve článku *A Formal Basis for the Heuristic Determination of Minimum Cost Paths* [13]. Vstupem algoritmu je ohodnocený graf, počáteční a cílový uzel. Výstupem je optimální cesta za předpokladu, že řešení úlohy existuje. Kromě výše zmíněného článku bylo v této podkapitole čerpáno také z publikací [6, 19].

Algoritmus používá prioritní frontu, ve které jsou uzly určené k expanzi seřazené dle hodnotící funkce:

$$f(i) = g(i) + h(i), \tag{2.1}$$

kde $g(i)$ je funkce udávající nejmenší dosud zjištěnou vzdálenost mezi počátečním uzlem a uzlem i a $h(i)$ je heuristická funkce představující odhad vzdálenosti z uzlu i do koncového uzlu. Tato funkce musí být přípustná, aby bylo zaručené nalezení optimálního řešení. Funkce $h(i)$ je přípustná, platí-li

$$0 \leq h(i) \leq h^*(i) \tag{2.2}$$

pro všechny stavy i, kde $h^*(i)$ je skutečná vzdálenost mezi uzlem i a koncovým uzlem. Čím více se blíží odhadovaná hodnota ke skutečnosti, tím menší část grafu se prohledává a při $h(i) = h^*(i)$ algoritmus expanduje pouze uzly na cestě k cílovému uzlu. Další vlastností, kterou může heuristická funkce disponovat je monotónnost. Funkce $h(i)$ je monotónní, jestliže

$$h(i) \leq h(j) + d_{ij}, \tag{2.3}$$

kde d_{ij} je hodnota přechodu z uzlu i do uzlu j. Je-li nerovnost (2.3) splněna, algoritmus bude expandovat každý uzel maximálně jednou.

Heuristickou funkci lze zvolit libovolně, avšak při hledání nejkratší cesty je vhodné vycházet z teorie metrických prostorů. Jako jedna z možností, která je zároveň použita v této práci, se nabízí euklidovská vzdálenost pozice i od cílové pozice.

Postup funkce algoritmu je následující. Algoritmus při inicializaci zapíše počáteční uzel do prioritní fronty uzlů určených k expanzi (*OPEN*). Následně je v každé iteraci z fronty *OPEN* odebrán uzel s nejmenší hodnotou $f(i)$, zapíše se do seznamu již expandovaných uzlů (*CLOSED*) a pro všechny jeho sousední uzly se spočtou hodnoty funkcí $f(i)$ a $g(i)$. Jestliže tyto uzly nejsou ve frontě *OPEN* ani v seznamu *CLOSED*, jsou přidány do fronty *OPEN*. Pokud se již v prioritní frontě jeden nebo více uzlů nachází, avšak s ohodnocením větším než právě vypočtené $f(i)$, aktualizuje se ohodnocení $f(i)$ a změní se jméno rodičovského uzlu v zápisu uzlů. Algoritmus výpočet ukončí nalezením nejkratší cesty, jestliže je z fronty *OPEN* při výběru zvolen cílový uzel. Délka nejkratší cesty odpovídá hodnotě $g(i)$ cílového uzlu. V případě, že algoritmus výpočet ukončí z důvodu vyprázdnění fronty *OPEN*, řešení pro danou úlohu neexistuje. Postup hledání je detailnější popsán v algoritmu 1.

Algoritmus 1 Algoritmus A*

```
1: function AStar(G = (N, A), start, target)
2:     CLOSED ← ∅
3:     OPEN ← start
4:     for each node i ∈ N do
5:         g(i) ← ∞
6:         f(i) ← ∞
7:     g(start) ← 0
8:     f(start) ← h(start)
9:     while OPEN is not empty do
10:        current ← PopMinF(OPEN)
11:        if current = target then
12:            break
13:        CLOSED.add(current)
14:        for each neighbor ∈ Neighbors(current) do
15:            if neighbor ∈ CLOSED then
16:                continue
17:            temp_g ← g(current) + d_{current,neighbor}
18:            if neighbor ∈ OPEN and temp_g < g(neighbor) then
19:                OPEN.remove(neighbor)
20:            if neighbor ∈ CLOSED and temp_g < g(neighbor) then
21:                CLOSED.remove(neighbor)
22:            if neighbor ∉ OPEN and neighbor ∉ CLOSED then
23:                OPEN.add(neighbor)
24:                g(neighbor) ← temp_g
25:                f(neighbor) ← g(neighbor) + h(neighbor)
26:                predecessor(neighbor) ← current
27:    return failure

28: function GetPath(G = (N, A), start, target)
29:    path ← empty vector
30:    current ← target
31:    while current is not target do
32:        path.add(current)
33:        current ← predecessor(current)
34:    path.add(target)
35:    return path
```

2.2.2 Dijkstrův algoritmus

Dijkstrův algoritmus sloužící k nalezení nejkratší cesty poprvé popsal E. W. Dijkstra v jeho článku *A note on two problems in connexion with graphs* [8]. Vstupem algoritmu je kladně ohodnocený graf a počáteční uzel. Výstupem je nejkratší možná cesta z počátečního uzlu do ostatních uzlů grafu za předpokladu, že daná cesta existuje. Jestliže je žádoucí nalézt nejkratší cestu pouze mezi dvěma uzly, součástí vstupních hodnot bude i cílový uzel. Algoritmus se v tomto případě ukončí, jakmile nalezne nejkratší cestu do zadaného uzlu, pokud taková cesta existuje. Při psaní textu této podkapitoly bylo mimo článek [8] vycházeno také z [7, 23].

Algoritmus je založen na postupném zpřesňování odhadů délky nejkratší cesty z počátku k ostatním uzlům. Algoritmus používá prioritní frontu, ve které jsou jednotlivé uzly seřazeny dle odhadu vzdálenosti mezi počátečním a daným uzlem. Při expanzi uzlu se provede ohodnocení jeho sousedů a nově získaná hodnota se porovná se současnou vzdálenosti do daného uzlu pomocí nerovnosti

$$g(i) + d_{ij} < g(j), \tag{2.4}$$

kde $g(i)$ a $g(j)$ jsou aktuální odhady nejkratších vzdáleností do uzlů i a j. Jestliže je nerovnost pravdivá, stane se $g(i) + d_{ij}$ novým odhadem $g(j)$.

Proces hledání nejkratší cesty pomocí Dijkstrova algoritmu je následující. Nejprve se inicializuje odhad vzdálenosti počátečního bodu s na $g(s) = 0$. Odhad vzdáleností pro zbylé uzly se inicializuje na nekonečno. Poté algoritmus v každé iteraci odebere z fronty uzel s nejmenším odhadem vzdálenosti a ten expanduje. Pro každý sousední uzel se provede porovnání odhadů vzdáleností dle vztahu (2.4) a případná aktualizace hodnot. Algoritmus končí výpočet, jakmile vyprázdní frontu uzlů. V případě zadání cílového uzlu je výpočet ukončen pokud je z fronty odebrán cílový uzel. Detailněji je celý proces popsán algoritmem 2 [2].

2.2.3 Floyd-Warshallův algoritmus

Floyd-Warshallův algoritmus slouží k nalezení nejkratší cesty mezi všemi dvojicemi uzlů v grafu. Je založen na dynamickém programování, které je blíže popsané v podkapitole 2.3.2. Algoritmus je pojmenován podle R. W. Floyda a S. Warshalla, kteří jej nezávisle na sobě popsali ve svých článcích *Algorithm 97: Shortest Path* [11] a *A Theorem on Boolean Matrices* [24]. Vstupem algoritmu je ohodnocený graf. V grafu se mohou nacházet i záporně ohodnocené hrany, ale pro správnou funkci algoritmu nesmí graf obsahovat záporně ohodnocené cykly. Výstupem algoritmu jsou nejkratší možné cesty a jejich délky mezi všemi dvojicemi uzlů. Obsah této kapitoly byl vytvořen za pomocí článků představených výše a publikací [2, 7, 23].

Algoritmus metodou postupného zlepšování zvětšuje s každou iterací množinu vnitřních vrcholů, které jsou uvažovány na cestách, dokud neobsáhne všechny možné

Algoritmus 2 Dijkstrův algoritmus

```
1: function DIJKSTRA(G = (N, A), start, target)
2:     for each node i ∈ N do
3:         g(i) ← ∞
4:     g(start) ← 0
5:     predecessor(start) ← None
6:     unvisited_nodes ← N
7:     while unvisited_nodes is not empty do
8:         current ← PopMinG(unvisited_list)
9:         if current = target then
10:            break
11:        for each neighbor ∈ Neighbors(current) do
12:            if g(current) + d_{current,neighbor} < g(neighbor) then
13:                g(neighbor) ← g(current) + d_{current,neighbor}
14:                predecessor(neighbor) ← current

15: function GETPATH(G = (N, A), start, target)
16:    path ← empty vector
17:    current ← target
18:    while current is not target do
19:        path.add(current)
20:        current ← predecessor(current)
21:    path.add(start)
22:    return path
```

vrcholy. V každé iteraci algoritmu, se pomocí nerovnosti (2.5) porovná současná nejkratší známá cesta t_{ij} mezi dvojicí uzlů i a j s cestou vedoucí skrz uzel k. Pokud je tato podmínka splněna, délka cesty t_{ij} je aktualizována na hodnotu $t_{ik} + t_{kj}$.

$$t_{ij} > t_{ik} + t_{kj} \tag{2.5}$$

Při výpočtu algoritmus nejprve inicializuje hodnoty vzdáleností mezi dvojicemi uzlů na hodnoty $t_{ii} = 0$ a $t_{ij} = d_{ij}$. V případě, že neexistuje hrana spojující uzly i a j, je $d_{ij} = \infty$. Následně se iterativně prochází všechny dvojice uzlů a za pomocí nerovnosti (2.5) se aktualizují délky nejkratších cest. Při aktualizaci délek cest mezi uzly je také příhodné ukládat předchůdce uzlu j do vhodné datové struktury, aby bylo možné získat posloupnost uzlů nejkratší trasy. Detailněji je celý postup, včetně funkce pro rekonstrukci cesty mezi dvěma zadanými uzly popsán algoritmem 3.

Algoritmus 3 Floyd-Warshallův algoritmus

```
1: function FLOYD-WARSHALL(G = (N, A))
2:     distances ← |N| × |N| matrix
3:     predecessors ← |N| × |N| matrix
4:     for each pair (i, j) ∈ distances do
5:         distances(i, j) ← ∞
6:         predecessors(i, j) ← j
7:     for each node i ∈ N do
8:         distances(i, i) ← 0
9:     for each arc (i, j) ∈ A do
10:        distances(i, j) ← weight(i, j)
11:    for each k from 0 to |N| do
12:        for each i from 0 to |N| do
13:            for each j from 0 to |N| do
14:                if distances(i, j) > distances(i, k) + distancess(k, j) then
15:                    distances(i, j) ← distances(i, k) + distances(k, j)
16:                    predecessors(i, j) ← predecessors(i, k)

17: function GETPATH(predecessors, start, target)
18:    path ← empty vector
19:    current ← target
20:    while current is not target do
21:        path.add(current)
22:        current ← predecessors(current, target)
23:    path.add(target)
24:    return path
```

2.3 Optimalizační metody

Řešením optimalizačního problému je minimalizace či maximalizace jeho účelové funkce, jejíž proměnné mohou být ohraničeny omezujícími podmínkami [25]. Mezi matematické metody, které lze při řešení optimalizačních úloh použít patří např. metody matematického programování (lineární programování, nelineární programování, celočíselné programování, apod.), metody založené na matematické teorii procesů (dynamické programování), metody teorie her, simulační metody, atd. [10]. V této podkapitole jsou blíže představeny metody celočíselného a dynamického programování, které byly zároveň použity v této práci.

2.3.1 Celočíselné programování

Úlohy matematického programování, jejichž některé nebo všechny rozhodovací proměnné mají požadavky na celočíselnost jsou nazývány úlohami celočíselného programování. Obecná formulace této úlohy lze popsat jako:

$$\text{minimalizace} \quad f(x_0, x_1, ..., x_n) \tag{2.6}$$

za podmínek

$$g_i(x_0, x_1, ..., x_n) \leq 0, \quad i = 0, 1, ..., m, \tag{2.7}$$

$$x_j \in M_j \subseteq \mathbb{Z}, \quad j \in J, \tag{2.8}$$

kde $J \neq \emptyset$, $J \subseteq \{0, 1, ...n\}$ a M_j je množina celočíselných hodnot, kterých může proměnná x_j nabývat.

Celočíselné úlohy lze dle charakteru funkcí rozdělit na lineární a nelineární. Dále lze úlohy klasifikovat dle rozsahu podmínky celočíselnosti na:

- úlohy úplně celočíselné – podmínka celočíselnosti se vztahuje na všechny proměnné
- úlohy částečně celočíselné – podmínka celočíselnosti se týká pouze některých proměnných
- úlohy bivalentního programování – proměnné mohou nabývat pouze hodnot nula nebo jedna

Při řešení celočíselných úloh lze použít vhodnou metodu lineárního nebo nelineárního programování a upravit získané výsledky na celá čísla. Při využití tohoto způsobu ale hrozí, že nastane příliš velká chyba ve srovnání s hodnotou účelové funkce v celočíselném optimálním řešení, nebo že získané řešení nebude přípustné. Pro řešení komplikovanějších úloh je vhodnější zvolit některou z metod celočíselného programování jako např. metody sečných nadrovin nebo metodu větví a mezí. K řešení celočíselných úloh lze dále použít metody dynamického programování, různé heuristické metody, nebo speciální metody pro řešení problému určitého typu. Obsah podkapitoly byl napsán za pomocí výukových materiálů [10, 15].

2.3.2 Dynamické programování

Dynamické programování slouží k optimalizaci dynamických úloh, které lze popsat také jako úlohy řízení procesů probíhajících v čase. Metody dynamického programování je však možné použít také k řešení problémů, v nichž čas přímo nevystupuje a může do nich být uměle zaveden. Informace dostupné v této kapitole byly čerpány z knihy [5] a výukových materiálů [10, 15].

Obecnou úlohu dynamického programování je možné formulovat diskrétním systémem

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, ..., N - 1, \tag{2.9}$$

kde x_k je stav systému v diskrétním čase k, u_k je řídící rozhodnutí v čase k, w_k představuje náhodný parametr, N udává počet vykonání řídících rozhodnutí a f_k je funkce popisující systém a jeho chování při přechodu mezi jednotlivými fázemi. Cenová funkce $g_k(x_k, u_k, w_k)$ narůstá v každém časovém okamžiku k a celková cena za všechny přechody je dána vztahem

$$g_N + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k). \tag{2.10}$$

Jednotlivé přechody mezi stavy lze popsat jako posloupnost rozhodnutí

$$\pi = \{\mu_0, ..., \mu_{N-1}\}, \tag{2.11}$$

kde μ_k přiřazuje jednotlivým stavům x_k řídící rozhodnutí $u_k = \mu_k(x_k)$.

Přístupy dynamického programování jsou založené na využití rekurentních vztahů a opírají se přitom o princip optimality, jehož autorem je R. Bellman [4]. Bellmanův princip optimality udává, že jakákoli zbývající část optimální strategie $\pi^* = \{\mu_0^*, \mu_1^*, ..., \mu_{N-1}^*\}$ je rovněž optimální, pokud proces začíná ve stavu, do nějž se dostane v důsledku předcházející části optimální strategie. Z tohoto principu tak vyplývá, že optimální řídící strategii lze získat po částech. Nejprve se získá optimální řešení pro poslední stav a dále se postupuje zpětně v čase a počítají se optimální řešení pro jednotlivé stavy tak dlouho, než se postoupí až k počátečnímu stavu. Algoritmus dynamického programování pro obecný problém je dán vztahy

$$J_N(x_N) = g_N(x_N), \tag{2.12}$$

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k)\}, \quad k = 0, 1, ...N-1, \tag{2.13}$$

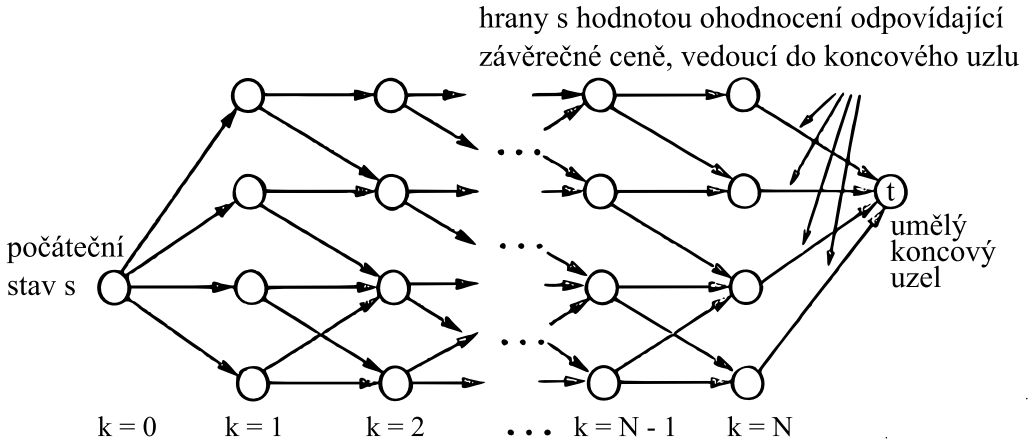
kde $J_k(x_k)$ je cena za přechod ze stavu x_k v čase k do koncového stavu a E_{w_k} značí operátor střední hodnoty. Výstupem algoritmu je optimální cena $J_0(x_0)$.

Pro deterministickou úlohu, kde stavový prostor S_k je konečný $\forall k$, je možné v jakémkoliv stavu x_k přiřadit rozhodovací proměnnou u_k k přechodu ze stavu x_k do stavu $f_k(x_k, u_k)$ za cenu $g_k(x_k, u_k)$. Z toho vyplývá, že deterministický problém s konečným počtem stavů lze reprezentovat pomocí grafu, kde hrany představují přechody mezi stavy a jejich ceny. Závěrečná fáze je pak modelována přidáním umělého koncového uzlu t. Pro každý stav x_N v N-té etapě existuje do konečného uzlu t přechod s cenou $g_N(x_N)$. Graf reprezentující popsaný stavový prostor lze vidět na obr. 4. Posloupnosti řídících rozhodnutí odpovídají cestám z počátečního stavu s do jednoho ze stavů končících v N-té fázi. Řešení této úlohy je tedy možné získat nalezením nejlevnější cesty mezi uzly s a t. Algoritmus dynamického programování je pak možné vyjádřit rovnicí:

$$J_N(i) = a_{it}^N, \quad i \in S_N, \tag{2.14}$$

$$J_k(i) = \min_{j \in S_{k+1}} [a_{ij}^k + J_{k+1}(j)], \quad i \in S_k, \quad k = 0, 1, ...N-1, \tag{2.15}$$

kde a_{ij}^k označuje cenu přechodu k-té etapy ze stavu $i \in S_k$ do stavu $j \in S_{k+1}$, a_{it}^N je konečná cena stavu $i \in S_N$ a má hodnotu $a_{it}^N = g_N(i)$. Pro hrany, mezi kterými neexistuje žádné řídící rozhodnutí mezi stavy i a j platí $a_{ij}^k = \infty$. Výstupem tohoto algoritmu je optimální cena $J_0(s)$ a její hodnota je rovna délce nejkratší cesty z uzlu 0 do uzlu t.



Obr. 4: Graf zobrazující deterministický systém s konečným počtem stavů [5]

Předchozí algoritmus popsaný rovnicemi (2.14) a (2.15) postupuje zpětně v čase. Vzhledem k tomu, že optimální cesta z s do t je po obrácení směrů přechodů rovna optimální cestě z t do s, lze odvodit algoritmus dynamického programování, který bude postupovat v čase dopředu a má tvar

$$\tilde{J}_N(i) = a_{sj}^0, \quad j \in S_1, \tag{2.16}$$

$$\tilde{J}_k(j) = \min_{i \in S_{N-k}} [a_{ij}^{N-k} + \tilde{J}_{k+1}(i)], \quad j \in S_{N-k+1}, \quad k = 1, 2, ...N-1, \tag{2.17}$$

kde optimální cena zaujímá hodnotu $\tilde{J}_0(t) = \min_{i \in S_N}[a_{ij}^N + \tilde{J}_1(i)]$. Vztah mezi výsledky zpětného a dopředného algoritmu je pak možné popsat rovností $J_0(s) = \tilde{J}_0(t)$.

3 FORMULACE A ŘEŠENÍ PROBLÉMU

Při řešení problematiky optimálního plánování trasy pro elektromobily bylo nejprve nutné vytvořit pracovní prostředí v podobě rovinného grafu. Následně se specifikovaly parametry pro daný model jako např. výskyt dobíjecích stanic, počáteční a cílový bod, maximální a minimální hodnoty baterie, apod. Poté už je možné zahájit hledání a optimalizaci cesty vybranou metodou.

Vzhledem k deterministickému charakteru úloh řešených v této práci a seznamu doporučené literatury, byly pro modely plug-in hybrid vozidel zvoleny metody, které ve svém článku *Minimum cost path problem for Plug-in Hybrid Electric Vehicles* popsali O. Arslan, B. Yıldız a O. E. Karaşan [3]. Pro řešení úloh s modely elektromobilů bylo vybráno řešení pomocí Dijkstrova algoritmu a algoritmu A* v kombinaci s dynamickým programováním, neboť dohromady jsou tyto metody schopny nalézt velmi dobré výsledky v poměrně krátkém čase.

3.1 Tvorba simulačního prostředí

Jak už bylo zmíněno výše, simulační prostředí má podobu rovinného grafu. Ten se vytvoří náhodným generováním bodů ve vymezeném prostoru. Tímto způsobem je možné vytvářet grafy s různě velkým počtem uzlů a délek hran. Vstupními parametry, které je možné uživatelsky zadat jsou:

- Délky stran x a y prostoru, v němž se bude graf vytvářet
- Počet uzlů v grafu
- Minimální rozestup mezi každými dvěma uzly

Následně se provede Delaunayova triangulace, čímž se zaručí propojení uzlů hranami tak, že výsledný graf bude planární [23].

Do jednotlivých uzlů lze pak ukládat vlastnosti, jako přítomnost dobíjecí nebo čerpací stanice, či hodnotu funkce $g(i)$. Hranám je možné kromě jejich ohodnocení přiřadit také barvu, což je vhodné při vyznačení nalezené optimální cesty mezi dvěma body.

3.2 Optimalizace trasy elektromobilu Dijkstrovým algoritmem

Při optimalizaci trasy elektromobilu pomocí Dijkstrova algoritmu se vyhledává nejkratší cesta ve vstupním grafu $G = (N, A)$ při daném nastavení těchto parametrů modelu:

- s - počáteční uzel
- t - cílový uzel
- $\underline{P}$ - minimální hodnota nabití baterie [kWh]

- $\overline{P}$ - maximální hodnota nabití baterie [kWh]
- P_s - počáteční hodnota nabití baterie [kWh]
- P_t - konečná hodnota nabití baterie [kWh]
- ε - průměrná spotřeba energie [kWh/100km]
- c_i^e - cena elektřiny v uzlu i [Kč/kWh]
- c^{st} - cena za zastavení [Kč]
- c^{dep} - cena za opotřebení vozidla [Kč/km]
- s^e - vektor boolean hodnot udávající výskyt dobíjecí stanice v daném uzlu

Pokud by byl Dijkstrův algoritmus aplikován již na zadaný graf, mohla by nastat situace, kdy se na nalezené cestě nenacházejí vhodně rozmístěné dobíjecí stanice a model by po cestě nemohl do cíle dorazit z důvodu vybití baterie. Proto je vhodné před začátkem hledání optimální cesty nejprve vytvořit tzv. „*meta-network*“ původního grafu, ve kterém se má optimální trasa nalézt. Použití *meta-networku* v tomto případě zajistí, že nejkratší cesta nalezena Dijkstrovým algoritmem povede skrze uzly s dobíjecími stanicemi a po hranách, jejichž délku je model elektromobilu schopen ujet. *Meta-network* je však možné použít také k urychlení optimalizace cesty ve velkých grafech při použití výpočetních metod uvedených v kapitole 3.4 [3].

Meta-network se vytvoří nalezením nejkratší cesty v grafu mezi počátečním, cílovým a všemi ostatními uzly, v nichž se vyskytuje dobíjecí stanice baterie a následným nahrazením těchto tras jedním přechodem. Aby byla nová hrana spojující původní uzly do *meta-networku* přidána, musí splňovat podmínku:

$$P - \varepsilon \cdot d_{ij} \geq P_{min}, \tag{3.1}$$

kde P vyjadřuje současnou hodnotu nabití baterie a P_{min} udává minimální hodnotu nabití pro uzel, do kterého přechod směřuje. Předpokládá se, že model baterii může nabít do plné kapacity při každé návštěvě dobíjecí stanice. Parametr P tak nabývá hodnoty $\overline{P}$ jestliže $i \neq s$ nebo P_s, pokud $i = s$. Proměnná P_{min} slouží k zajištění dostatečné hodnoty nabití v koncovém uzlu t a její výchozí hodnota je $P_{min} = \underline{P}$ a při $j = t$ je $P_{min} = \max\{\underline{P}, P_t\}$. Porovnání vstupního grafu s jeho *meta-networkem* je k vidění na obrázcích 5 a 6, kde přítomnost stanic je ve výchozím grafu znázorněna žlutým zbarvením uzlů a počáteční a cílový uzel jsou dány modrou a červenou značku okolo uzlu.

Jakmile je *meta-network* vytvořen, aplikuje se na něj Dijkstrův algoritmus s úkolem najít optimální trasu mezi body s a t. Může se stát, že *meta-network* je nesouvislý graf a cesta mezi dvěma zadanými body neexistuje. V takovém případě je model neřešitelný a pro jeho řešení je nutné upravit některé ze zadaných parametrů (např. přidáním dobíjecí stanice). Pokud je nejkratší cesta nalezena, provede se rekonstrukce výsledné trasy zpět do původního grafu.

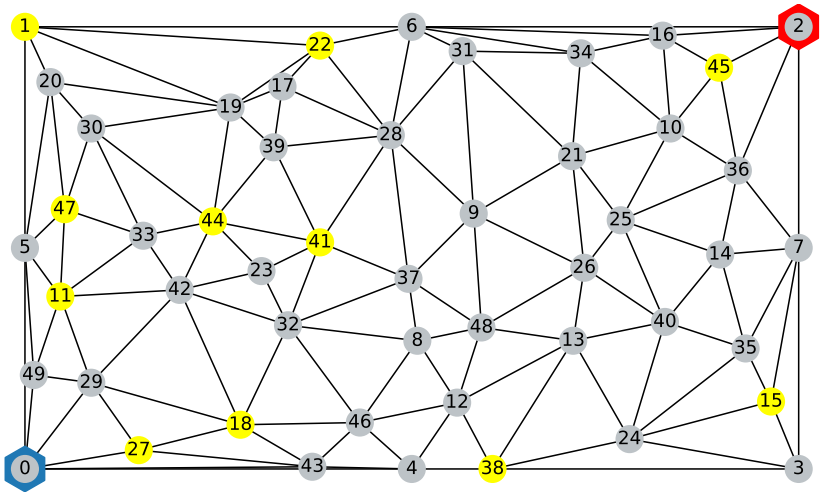
Následně je možné provézt optimalizaci nabíjení na získané trase. Tato operace se provede pomocí algoritmu dynamického programování pro diskrétní konečný

stavový prostor. Nejprve se provede diskretizace intervalu hodnot úrovně nabití $\langle \underline{P}, \overline{P} \rangle$ na konečný počet prvků. Poté se zpětným algoritmem dynamického programování napočítají ceny přechodů a řídící rozhodnutí pro jednotlivé stavy a nakonec se z těchto hodnot vybere optimální strategie dobíjení v jednotlivých uzlech. Celková cena trasy se spočítá jako

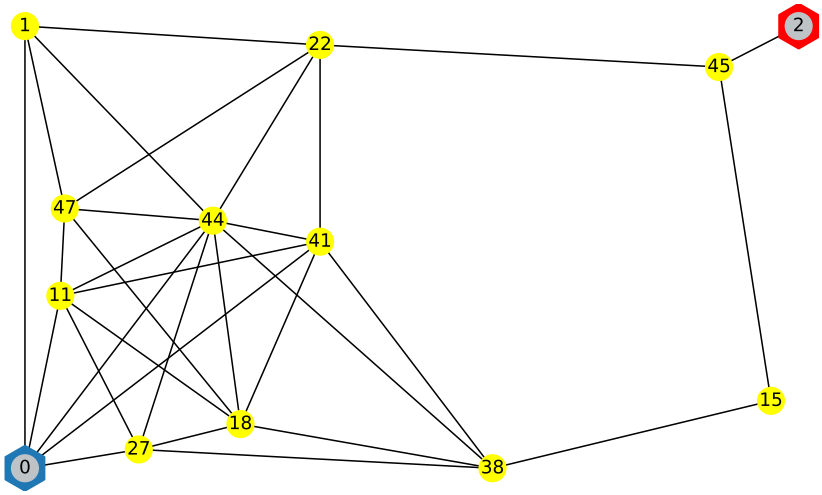
$$\sum_{i \in N} c_i^e \cdot e_i^+ + \sum_{(i,j) \in A} d_{ij} \cdot c^{dep} + \sum_{i \in N} c^{st} \cdot v_i, \tag{3.2}$$

kde e_i^+ označuje množství doplněné energie v uzlu i a v_i udává, jestli se v uzlu i zastavilo za účelem dobití baterie a může nabývat hodnot $v_i = 1$, jestliže platí $e_i^+ > 0$, v opačném případě $v_i = 0$.

Implementace algoritmu dynamického programování je podrobněji znázorněna v algoritmu 4. Vstupem je vektor uzlů nacházející se na optimální trase, vektor cen dobíjení v jednotlivých uzlech, vektor vzdáleností jednotlivých hran mezi uzly a dále parametry spojené s baterií a cenou představené na začátku této kapitoly.



Obr. 5: Ukázka grafu s dobíjecími stanicemi



Obr. 6: *Meta-network* grafu na obr. 6

Algoritmus 4 DP algoritmus pro optimalizaci nabíjení na zadané cestě

function GetChargingCost($path, costs, distances, \underline{P}, \overline{P}, P_s, P_t, \varepsilon, c^{st}, c^{dep}$)
  $energy\_distances \leftarrow [distances \cdot \varepsilon/100;\ 0]$
  $n \leftarrow length(path)$
  $discretization \leftarrow (\overline{P} - \underline{P}) \cdot 10 + 1$
  $x_s \leftarrow linspace(\underline{P}, \overline{P}, discretization)$
  $J \leftarrow discretization \times n$ zero matrix
  $\mu \leftarrow discretization \times n$ zero matrix
  for each i **from** 0 **to** $discretization$ **do**
    if $P_t \leq x_s(i)$ **then**
      $J(i, n-1) \leftarrow 0$
    else
      $J(i, n-1) \leftarrow costs(n-1) \cdot (P_t - x_s(i)) + c^{st}$
      $\mu(i, n-1) \leftarrow P_t - x_s(i)$
  for each k **from** $n-1$ **to** 0 **do**
    for each i **from** 0 **to** $discretization$ **do**
      $u_s \leftarrow x_s(i + 0 : discretization) - x_s(i)$
      $J_s \leftarrow length(u_s) \times 1$ zero vector
      $g_s \leftarrow [0;\ costs(k) \cdot u_s(1 : end) + c^{st}]$
      $x_{next} \leftarrow x_s(i) + u_s - energy\_distances(k)$
      for each j **from** 0 **to** $length(u_s)$ **do**
        if $x_{next} < \underline{P}$ **then**
          $J_s \leftarrow \infty$
        else
          $idx \leftarrow$ vector of indices of all $x_s \leq x_{next}(j)$
          $J_s(j) \leftarrow g_s(j) + J(idx(end), k+1)$
      $[minval, minidx] \leftarrow min(J_s)$
      $J(i, k) \leftarrow minval$
      $\mu(i, k) \leftarrow u_s(minidx)$
  $s \leftarrow n + 1 \times 1$ zero vector
  $s(0) \leftarrow (P_s - \underline{P}) \cdot 10$
  $x_{real} \leftarrow n + 1 \times 1$ zero vector
  $x_{real}(0) \leftarrow x_s(s(0))$
  $u_{real} \leftarrow n \times 1$ zero vector
  for each i **from** 0 **to** n **do**
    $u_{real}(i) \leftarrow \mu(s(i), i)$
    $x_{next} = x_{real}(i) + u - energy\_distances(i)$
    $idx \leftarrow$ vector of indices of all $x_s \leq x_{next}$
    $x_{real}(i+1) \leftarrow x_s(idx(end))$
    $s(i+1) \leftarrow idx(end)$

3.3 Optimalizace trasy elektromobilu algoritmem A*

Postup nalezení cesty a optimalizace nabíjení pomocí algoritmu A* je obdobný jako v kapitole 3.2. *Meta-network* je rovněž vytvořen prohledáváním grafu Dijkstrovým algoritmem a následně se v něm s využitím algoritmu A* hledá nejkratší cesta. Pokračuje se rekonstrukcí nalezené cesty do výchozího grafu a optimalizací nabíjení pomocí dynamického programování.

3.4 Metody Arslan

Článek [3] jmenovaný na počátku této kapitoly pojednává o metodách optimalizace trasy nejen pro elektromobily, ale také pro plug-in hybrid vozidla (PHEV). V článku jsou postupně popsány vlastnosti modelu, způsob generování *meta-networku*, čtyři metody řešení a porovnání výsledku jednotlivých postupů.

Minimalizační úloha nákladů na cestu plug-in hybrid vozidla (MCPP-PHEV) je v článku [3] definována, jako instance $\langle V, X, s, t, P_s, P_t, G_s, G_t \rangle$, kde V je instance vozidla, kterou tvoří vektor $\langle \overline{P}, \underline{P}, \overline{G}, \underline{G}, \varepsilon, \rho \rangle$ a X je instance dopravní sítě tvořena n-ticí $\langle N, A, s^e, s^g, c^e, c^g, d \rangle$. Výstupem této úlohy je řešení v podobě trojice $\langle x, e^+, g^+ \rangle$. Jednotlivé parametry úlohy a jejich význam jsou blíže specifikovány níže v podkapitole 3.4.1. Pro přiblížení modelu reálnému vozidlu je možné úlohu rozšířit navíc o cenu za opotřebení vozidla, cenu za zastavení a vliv degradace baterie. Takovou úlohu je možné označit jako rozšířenou (E-MCPP-PHEV).

V této práci je na následujících stránkách popsána funkce, vlastnosti a následná aplikace jednotlivých metod. Implementace popsaných postupů v této práci se od článku liší zanedbáním vlivu degradace baterie. Takto bylo rozhodnuto z důvodu, že jedna z okrajových podmínek týkající se parametru degradace baterie obsahuje proměnnou s druhou mocninou, což by tuto lineární úlohu celočíselného programování změnilo na úlohu kvadratického programovaní, jejíž výpočetní náročnost je mnohonásobně vyšší.

3.4.1 Matematický model E-MCPP-PHEV

Prvním popsaným způsobem řešení je matematický model minimalizačního problému hledání nejkratší cesty. Parametry a proměnné použité k vyřešení modelu jsou prezentovány níže. Předpokládá se, že všechny parametry jsou známé a řídící rozhodnutí jsou na nich závislé.
Seznam parametrů:

- N, A - seznamy uzlů a hran grafu
- s, t - počáteční a cílový uzel

- s_i^e, s_i^g - nabývá hodnotu 1 pokud se v uzlu i nachází dobíjecí nebo tankovací stanice, jinak 0
- $\overline{P}, \underline{P}$ - maximální a minimální hodnota kapacity baterie [kWh]
- $\overline{G}, \underline{G}$ - maximální a minimální hodnota kapacity nádrže [l]
- P_s, P_t - počáteční a koncová hodnota kapacity baterie [kWh]
- G_s, G_t - počáteční a koncová hodnota kapacity nádrže [l]
- ε - průměrná spotřeba energie [kWh/100km]
- ρ - průměrná spotřeba paliva [l/100km]
- d_{ij} - délka hrany (i, j) [km]
- c_i^e - cena elektřiny v uzlu i [Kč/kWh]
- c_i^g - cena paliva v uzlu i [Kč/l]
- c^{st} - cena za zastavení [Kč]
- c^{dep} - cena za opotřebení vozidla [Kč/km]

Seznam proměnných:

- e_i^α, e_i^β - hodnota nabití v uzlu i při příjezdu a odjezdu vozidla [kWh]
- e_i^+ - přírůstek hodnoty nabití v uzlu i [kWh]
- g_i^α, g_i^β - hodnota paliva v uzlu i při příjezdu a odjezdu vozidla [l]
- e_i^+ - přírůstek hodnoty paliva v uzlu i [l]
- x_{ij} - nábývá hodnoty 1, pokud je hrana (i, j) součástí optimální cesty, jinak 0
- v_i - nábývá hodnoty 1, pokud se v uzlu i dobíjí baterie, jinak 0
- r_i - nábývá hodnoty 1, pokud se v uzlu i doplňuje palivo a/nebo dobíjí baterie, jinak 0
- d_{ij}^{cd}, d_{ij}^{cs} - ujetá vzdálenost v režimu spotřeby energie baterie (*„charge depleting“* – CD) a spotřeby paliva (*„charge sustaining“* – CS) po hraně (i, j) [km]

Provádí se minimalizace účelové funkce

$$\sum_{i \in N} c_i^e \cdot e_i^+ + \sum_{i \in N} c_i^g \cdot g_i^+ + \sum_{(i,j) \in A} d_{ij} \cdot c^{dep} + \sum_{i \in N} c^{st} \cdot r_i \tag{3.3}$$

za následujících okrajových podmínek:

$$\sum_{j:(i,j) \in A} x_{ij} - \sum_{j:(j,i) \in A} x_{ji} = \begin{cases} 1, & i = s \\ -1, & i = t \\ 0, & \forall i \in N / \{s, t\} \end{cases}, \tag{3.4}$$

$$e_i^\beta = e_i^\alpha + s_i^e \cdot e_i^+, \quad \forall i \in N, \tag{3.5}$$

$$M \cdot (x_{ij} - 1) \leq e_j^\alpha - e_i^\beta + \varepsilon \cdot d_{ij}^{cd} \leq M \cdot (1 - x_{ij}), \quad \forall (i, j) \in A, \tag{3.6}$$

$$\underline{P} \leq e_i^\alpha \leq \overline{P}, \quad \forall i \in N, \tag{3.7}$$

$$\underline{P} \leq e_i^\beta \leq \overline{P}, \quad \forall i \in N, \tag{3.8}$$

$$e_i^+ \leq v_i \cdot \overline{P}, \quad \forall i \in N, \tag{3.9}$$

$$v_i \leq r_i, \quad \forall i \in N, \tag{3.10}$$

$$e_s^\alpha = P_s, \tag{3.11}$$

$$e_t^\alpha \geq P_t, \tag{3.12}$$

$$g_i^\beta = g_i^\alpha + s_i^g \cdot g_i^+, \quad \forall i \in N, \tag{3.13}$$

$$M \cdot (x_{ij} - 1) \leq g_j^\alpha - g_i^\beta + \rho \cdot d_{ij}^{cs} \leq M \cdot (1 - x_{ij}), \quad \forall (i,j) \in A, \tag{3.14}$$

$$\underline{G} \leq g_i^\alpha \leq \overline{G}, \quad \forall i \in N, \tag{3.15}$$

$$\underline{G} \leq g_i^\beta \leq \overline{G}, \quad \forall i \in N, \tag{3.16}$$

$$g_i^+ \leq r_i \cdot \overline{G}, \quad \forall i \in N, \tag{3.17}$$

$$g_s^\alpha = G_s, \tag{3.18}$$

$$g_t^\alpha \geq G_t, \tag{3.19}$$

$$d_{ij}^{cs} + d_{ij}^{cd} = d_{ij}, \quad \forall (i,j) \in A, \tag{3.20}$$

$$x_{ij}, v_k, r_k \in \{0,1\}; d_{ij}^{cd}, d_{ij}^{cs}, e_k^\alpha, e_k^\beta, e_k^+, g_k^\alpha, g_k^\beta, g_k^+ \geq 0, \quad \forall k \in N, \forall (i,j) \in A \tag{3.21}$$

Účelová funkce (3.3) minimalizuje cenu přechodů a skládá se z ceny za elektřinu a palivo, ceny za opotřebení vozidla a ceny za zastavení. Okrajová podmínka (3.4) vynucuje výskyt řešení na cestě z uzlu s do t. Omezení (3.5) jsou rovnice rovnováhy elektrické energie pro uzly, které udávají, že hodnota nabití při odjezdu z uzlu i se musí rovnat součtu hodnoty nabití při příjezdu a přírůstku hodnoty nabití v uzlu i. Omezení (3.6) jsou rovnice rovnováhy elektrické energie pro hrany, jež jsou součástí cesty z s do t. Pro hrany mimo cestu jsou omezení uvolněna odpovídající hodnotou konstanty M, které se pro případ elektrické energie získá výpočtem:

$$M = \overline{P} - \underline{P} + \varepsilon \cdot max(d), \tag{3.22}$$

kde $max(d)$ představuje výběr hodnoty nejdelší hrany. Hodnota této konstanty je volena tak, aby vyhovovala délkám všech hran a přitom nebyla zbytečně veliká, což by mohlo vézt k prodloužení výpočtu. Naopak příliš malá hodnota konstanty M by mohla vést k nepřesným výsledkům. Omezení (3.7) a (3.8) udávají horní a spodní mez pro hodnoty nabití baterie při příjezdu a odjezdu z uzlu. Omezení (3.9) přiřazují binární proměnné v_i hodnotu 1, jestliže byla baterie dobíjena v uzlu i. Omezení (3.10) vyžadují přiřazení proměnné r_i hodnotu 1, pokud je $v_i = 1$. Tím se zaručí nárůst ceny za zastavení v případě, že vozidlo zastaví v uzlu i z důvodu dobití baterie. Omezení (3.11) a (3.12) přiřazují hodnoty nabití baterie v uzlech s a t. Omezení (3.13)-(3.19) jsou protějšky omezení (3.5)-(3.12) pro případ paliva. Omezení (3.20) zaručuje, že suma ujetých vzdáleností na CD a CS režimy je rovna délce hrany, pokud se hrana nachází na cestě mezi dvěma zadanými uzly. Omezení (3.21) jsou definiční obory hodnot jednotlivých proměnných.

3.4.2 Heuristika dynamického programování s diskrétní aproximací (DH)

V této podkapitole je představena heuristika pro řešení úlohy (3.3)-(3.21) založená na dynamickém programování. Nejprve je zde definován seznam stavů úrovně nabití a paliva. Dále jsou představeny Bellmanovy rovnice, které by měly být splněny minimálními cenami přechodů, aby bylo možné usnadnit metodiku řešení dynamického programování. Nakonec je zde prezentována transformace grafu, jež vede ke zjednodušení výpočtu těchto rovnic.

V případě této heuristiky je stav definován jako trojice $\langle i, \sigma, \lambda \rangle$ reprezentující příjezd do uzlu $i \in N$ s úrovní nabití $\sigma \in [\underline{P}, \overline{P}]$ kWh a $\lambda \in [\underline{G}, \overline{G}]$ l paliva. Takto definovaný stav se značí pomocí $\omega_i^{\sigma,\lambda}$. V případě, že nezáleží na specifické hodnotě i, σ nebo λ, lze stav zapisovat jako ω_i nebo pouze ω.

Jestliže je dána instance E-MCPP-PHEV $\langle V, X, s, t, P_s, P_t, G_s, G_t \rangle$, její řešení $\langle x, e^+, g^+ \rangle$ obsahuje cestu z uzlu s do t, kterou lze odvodit z vektoru x. Ze známých hodnot doplnění energie (e^+) a paliva (g^+) v uzlech je možné odvodit jednotlivé vzdálenosti hran ujeté na režimy CD a CS. Spolu se zadaným P_s a G_s, vektory x, e^+ a g^+ udávají stavy úrovně nabití a hladiny paliva v nádrži při příjezdu do uzlů na vypočtené trase. Z toho plyne, že pro každé řešení E-MCPP-PHEV, existuje unikátní sekvence stavů, které představují tohle řešení. Lze také říci, že E-MCPP-PHEV úlohy mají nespočet proveditelných řešení. Jelikož každé z těchto řešení se jedinečně mapuje na posloupnost stavů, je také nespočetný příslušný stavový prostor. Tento nespočetný stavový prostor je však možné aproximovat konečným spočetným stavovým prostorem, což je hlavní myšlenkou algoritmu DH.

Získání konečného stavového prostoru se provede pomocí diskretizačních parametrů $\xi, \tau \in \mathbb{N}$. Využitím těchto parametrů se získají množiny $\Sigma = \{\sigma_0, \sigma_1, ..., \sigma_\xi\}$ a $\Lambda = \{\lambda_0, \lambda_1, ..., \lambda_\tau\}$, kde $\sigma_0 = \underline{P}$, $\lambda_0 = \underline{G}$, $\sigma_k = \sigma_0 + k \cdot \frac{\overline{P}-\underline{P}}{\xi}, \quad \forall k \in \{1, 2, ..., \xi\}$ a $\lambda_l = \lambda_0 + l \cdot \frac{\overline{G}-\underline{G}}{\tau}, \quad \forall l \in \{1, 2, ..., \tau\}$. Každé σ_i tak reprezentuje interval úrovně nabití $[\sigma_i, \sigma_{i+1}], \quad \forall i \in \{0, 1, ..., \xi-1\}$, kde σ_ξ udává plně nabitou baterii. Zastoupení intervalů pro každé λ je obdobné.

Pro danou instanci E-MCPP-PHEV a diskretizační parametry ξ a τ je možné definovat diskrétní stavový prostor Ω jako:

$$\Omega = \{(\omega_i^{\sigma,\lambda} | i \in N - \{s, t\}), \sigma \in \Sigma, \lambda \in \Lambda\} \cup \{\omega_s^{P_s,G_s}, \omega_t^{P_t,G_t}\}. \tag{3.23}$$

Kardinalita tohoto stavového prostoru je ohraničena součinem $n \cdot (\xi+1) \cdot (\tau+1)$, kde n je konečný počet uzlů vstupního grafu. Využitím diskrétního stavového prostoru Ω v algoritmu DH vzniká aproximační chyba, kterou lze zmenšovat výběrem vyšších hodnot diskretizačních parametrů.

Vstupem funkce minimálních nákladů přechodu $f : \Omega \times \Omega \to \mathbb{R}^+$ jsou dva stavy $\omega_i^{\sigma,\lambda}$, $\omega_j^{\overline{\sigma},\overline{\lambda}}$ a jejím výstupem je minimální cena přechodu z uzlu i s hodnotou nabití baterie σ kWh a λ litry paliva do uzlu j s nabitím baterie alespoň $\overline{\sigma}$ kWh

a úrovní paliva nejméně $\overline{\lambda}$ l. Při výpočtu hodnoty $f(\omega_i^{\sigma,\lambda}, \omega_j^{\overline{\sigma},\overline{\lambda}})$ je bráno v úvahu pouze kolik je nutné dobít energie a natankovat paliva v uzlu i. Níže jsou popsány čtyři různé případy, které mohou nastat, a jak pro každý z nich spočítat hodnotu nákladů. Pro každou možnost je také stanovena podmínka řešitelnosti. Pokud tato podmínka není splněna, cena nákladů pro daný případ je rovna nekonečnu. Parametr $d^\star$ označuje nejkratší délky cest.

- Případ 1: Žádné dobíjení baterie a tankování paliva
  Podmínka řešitelnosti: Současná hodnota nabití a úroveň paliva jsou dostatečné na přechod z uzlu i do uzlu j za splnění podmínek konečného stavu:

$$\sigma \geq \overline{\sigma}, \lambda \geq \overline{\lambda} \quad \text{a} \quad \frac{(\sigma - \overline{\sigma})}{\varepsilon} + \frac{(\lambda - \overline{\lambda})}{\rho} \geq d_{ij}^\star. \tag{3.24}$$

  Celkové náklady: Jedinou vzniklou složkou nákladů je v tomto případě cena za opotřebení vozidla: $f_1(\omega_i^{\sigma,\lambda}, \omega_j^{\overline{\sigma},\overline{\lambda}}) = c^{dep} \cdot d_{ij}^\star$.
- Případ 2: Tankování paliva bez dobíjení baterie
  Podmínka řešitelnosti: Současná hodnota nabití a plná nádrž paliva jsou dostatečné na přechod z uzlu i do uzlu j za splnění podmínek konečného stavu:

$$s_i^g = 1, \sigma \geq \overline{\sigma} \quad \text{a} \quad \frac{(\sigma - \overline{\sigma})}{\varepsilon} + \frac{(\overline{G} - \overline{\lambda})}{\rho} \geq d_{ij}^\star. \tag{3.25}$$

  Celkové náklady: Přechod s minimálními náklady vyžaduje nejprve využít současnou hodnotu nabití baterie $(\sigma - \overline{\sigma})$. Ujetá část délky hrany se spočte pomocí $d_{ij}^{cd} = \min\{d_{ij}^\star, \frac{(\sigma-\overline{\sigma})}{\varepsilon}\}$. Zbylá vzdálenost, kterou je nutné urazit v CS režimu se pak spočte jako $d_{ij}^{cs} = d_{ij}^\star - d_{ij}^{cd}$. Dále je nutné v uzlu i dotankovat tolik paliva, aby byla pokryta ujetá vzdálenost a aby do cílového uzlu vozidlo dorazilo s $\overline{\lambda}$ litry paliva. Množství paliva, které se v uzlu i doplní se spočte rovnicí $g_i^+ = (d_{ij}^{cs} \cdot \rho + \overline{\lambda} - \lambda)^+$. Podmínkou řešitelnosti (3.25) je také zaručeno, že doplněné palivo bude v rozmezí $0 \leq g_i^+ \leq \overline{G} - \lambda$. Jednotlivé cenové složky tvořící celkové náklady jsou pouze ceny za palivo, opotřebení vozidla a zastavení: $f_2(\omega_i^{\sigma,\lambda}, \omega_j^{\overline{\sigma},\overline{\lambda}}) = c_i^g \cdot g_i^+ + c^{dep} \cdot d_{ij}^\star + c^{st}$.
- Případ 3: Dobíjení baterie bez tankování paliva
  Podmínka řešitelnosti: Plné nabití baterie a současná úroveň paliva jsou dostatečné na přechod z uzlu i do uzlu j za splnění podmínek konečného stavu:

$$s_i^e = 1, \lambda \geq \overline{\lambda} \quad \text{a} \quad \frac{(\overline{P} - \overline{\sigma})}{\varepsilon} + \frac{(\lambda - \overline{\lambda})}{\rho} \geq d_{ij}^\star. \tag{3.26}$$

  Celkové náklady: U tohoto případu se potřebné hodnoty spočtou analogicky k případu 2. Množství energie, o kterou je potřeba baterii v uzlu i dobít je $e_i^+ = (d_{ij}^{cd} \cdot \varepsilon + \overline{\sigma} - \sigma)^+$, kde $d_{ij}^{cd} = d_{ij}^\star - d_{ij}^{cs}$ a $d_{ij}^{cs} = \min\{d_{ij}^\star, \frac{(\lambda-\overline{\lambda})}{\varepsilon}\}$. Celkové náklady se pak skládají z ceny za elektrickou energii, ceny za opotřebení vozidla a ceny za zastavení: $f_3(\omega_i^{\sigma,\lambda}, \omega_j^{\overline{\sigma},\overline{\lambda}}) = c_i^e \cdot e_i^+ + c^{dep} \cdot d_{ij}^\star + c^{st}$.

- Případ 4: Dobíjení baterie a současné tankování paliva
  Podmínka řešitelnosti: Plné nabití baterie a plná nádrž paliva jsou dostatečné na přechod z uzlu i do uzlu j za splnění podmínek konečného stavu:

$$s_i^e = 1, s_i^g = 1 \quad \text{a} \quad \frac{(\overline{P} - \overline{\sigma})}{\varepsilon} + \frac{(\overline{G} - \overline{\lambda})}{\rho} \geq d_{ij}^{\star}. \tag{3.27}$$

  Celkové náklady: V tomto případě je nejprve definována nadměrná vzdálenost $\bar{d_{ij}}$, kterou nelze pokrýt současnými hodnotami nabití baterie a množství paliva před dobíjením nebo tankováním v uzlu i, pomocí rovnice:

$$\bar{d_{ij}} = (d_{ij}^{\star} - d_{ij}^{cd} - d_{ij}^{cs})^{+}, \tag{3.28}$$

  kde $d_{ij}^{cd} = \frac{(\sigma - \overline{\sigma})^{+}}{\varepsilon}$ a $d_{ij}^{cs} = \frac{(\lambda - \overline{\lambda})^{+}}{\rho}$. Na základě cen elektrické energie a paliva v uzlu i je také nutné určit, zda-li bude cenově výhodnější pokrýt vzdálenost $\bar{d_{ij}}$ na režim CD nebo CS:
  - Jestliže platí $(c_i^e \cdot \varepsilon) < (c_i^g \cdot \rho)$, je levnější využití režimu CD. Nejprve se provede kontrola, je-li plně nabitá baterie dostatečná k pokrytí potřebné vzdálenosti $\bar{d_{ij}}$. Platí-li $(d_{ij}^{cd} + \bar{d_{ij}}) < \frac{\overline{P} - \overline{\sigma}}{\varepsilon}$, na pokrytí potřebné vzdálenosti stačí plně nabitá baterie a hodnoty o kolik je nutné dobít baterii a doplnit palivo se spočtou jako $e_i^{+} = \bar{d_{ij}} \cdot \varepsilon + (\overline{\sigma} - \sigma)^{+}$ a $g_i^{+} = (\overline{\lambda} - \lambda)^{+}$. V opačném případě budou tyto hodnoty $e_i^{+} = (\overline{P} - \sigma)$ a $g_i^{+} = \left((d_{ij}^{\star} - \frac{\overline{P} - \overline{\sigma}}{\varepsilon}) \cdot \rho + \overline{\lambda} - \lambda\right)^{+}$.
  - Pokud vychází cenově levněji použít režim CS, výpočty potřebné k získání hodnot e_i^{+} a g_i^{+} jsou symetrické k výpočtům pro režim CD popsané výše.

  Jakmile jsou získány hodnoty doplnění baterie a paliva e_i^{+} a g_i^{+}, lze spočítat celkové náklady na přechod, které se skládají z ceny elektrické energie a paliva, ceny za opotřebení vozidla a ceny za zastavení:
  $f_4(\omega_i^{\sigma,\lambda}, \omega_j^{\overline{\sigma},\overline{\lambda}}) = c_i^e \cdot e_i^{+} + c_i^g \cdot g_i^{+} + c^{dep} \cdot d_{ij}^{\star} + c^{st}$.

Z těchto 4 případů je pro každou dvojici stavů vybrán přechod s minimální hodnotou funkce přechodu:

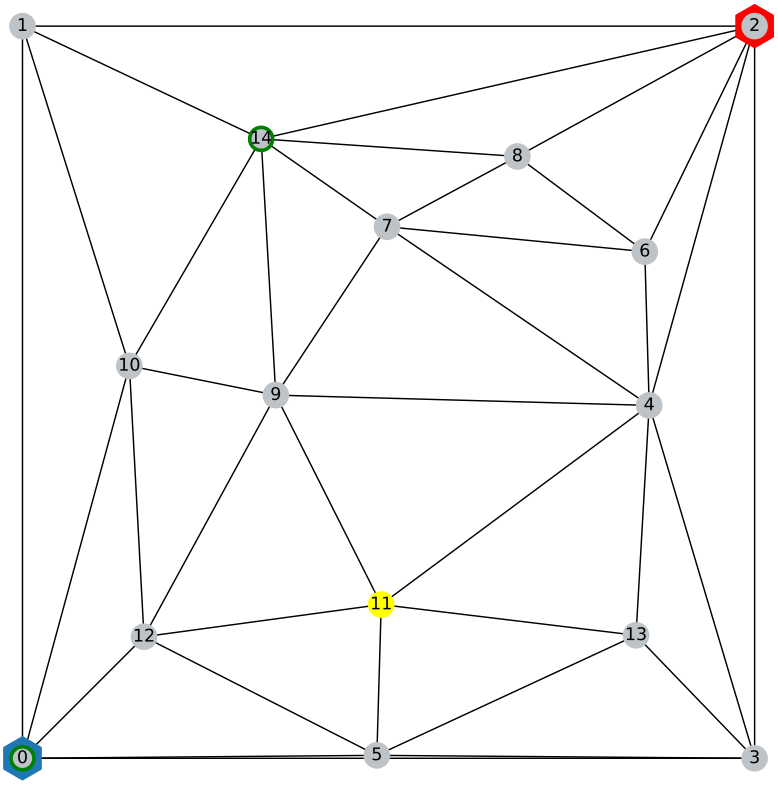
$$f(\omega, \overline{\omega}) = \min_{i \in \{1,2,3,4\}} \{f_i(\omega, \overline{\omega})\}. \tag{3.29}$$

Funkce $\pi : \Omega \to \mathbb{R}$ se nazývá hodnotová funkce a $\pi(\omega_i^{\sigma,\lambda})$ je optimální hodnota řešení E-MCPP-PHEV instance $\langle V, X, s, t, P_s, P_t, G_s, G_t \rangle$. Následující rovnice jsou založené na rovnicích dopředné varianty DP algoritmu (2.16) a (2.17):

$$\pi(\omega_s^{P_s,G_s}) = 0, \quad \pi(\omega) = \min_{\overline{\omega} \in \Omega} \{\pi(\overline{\omega}) + f(\omega, \overline{\omega})\} \quad \forall \omega \in \Omega. \tag{3.30}$$

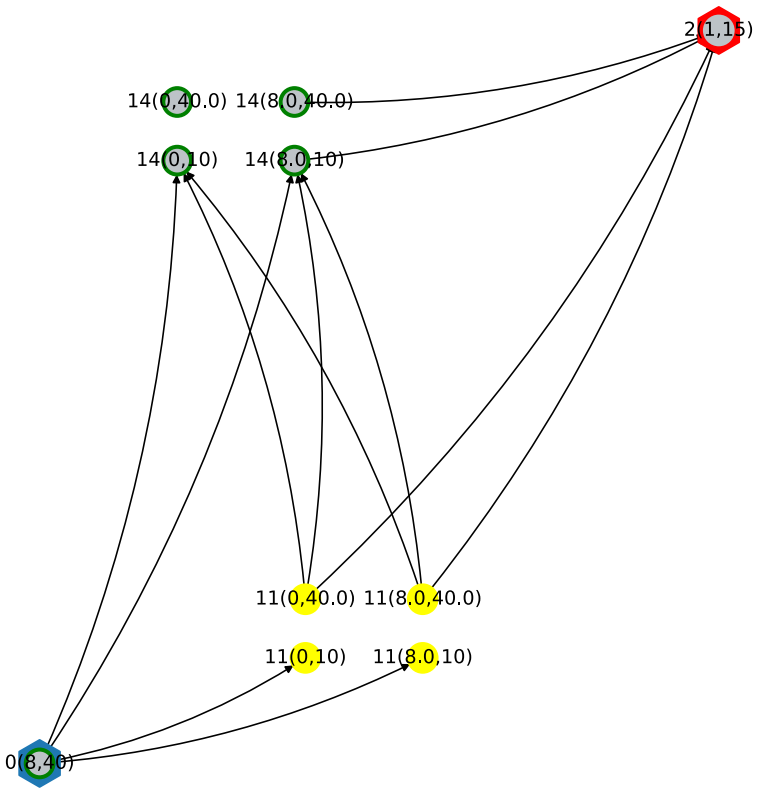
Vyřešení těchto rovnic je základ funkce DH algoritmu. Jejich výpočet lze zredukovat tvorbou tzv. DH-grafu $\tilde{G} = (\Omega, \tilde{A})$, kde množinu uzlů představuje diskrétní stavový prostor Ω a množina hran $\tilde{A}$ obsahuje hrany mezi stavy ω_i a $\omega_j \in \Omega$ s ohodnocením $f(\omega_i, \omega_j)$, pokud je funkce ceny konečná. Jakmile je tento graf získán, nalezne se

nejlevnější cesta mezi stavy $\omega_s^{P_s,G_s}$ a $\omega_t^{P_t,G_t}$, což je dostačující k vyřešení E-MCPP-PHEV úlohy, jelikož jednotlivé přechody mezi stavy v grafu obsahují také informace o dobíjení a tankování.



Obr. 7: Ukázka grafu s dobíjecí a čerpací stanicí

Při tvorbě DH-grafu lze původní graf nejprve zredukovat pouze na uzly, mezi kterými se hledá nejkratší cesta, uzly obsahující dobíjecí nebo čerpací stanici a nejkratší cesty mezi nimi. Následně se vytvoří diskrétní stavy pro jednotlivé uzly a nakonec se provede ohodnocení přechodů podle rovnice (3.29). Na obr. 8 je k vidění ukázka DH grafu, který byl vygenerován pro graf na obr. 7, kde zelené kolečko okolo uzlu značí přítomnost čerpací stanice. Pro zpřehlednění a také ušetření výpočetního času, nejsou do DH-grafu přidávány hrany vedoucí do uzlu, jež je od cíle vzdálenější než výchozí uzel příslušného přechodu. Pro vytvoření tohoto grafu byly použity hodnoty diskretizace $\xi = \tau = 1$. Popisky uzlů grafu na obr. 8 znázorňují jednotlivé diskrétní stavy, kde čísla před závorkami označují uzly původního grafu, ke kterému se stavy vztahují a hodnoty v závorkách pak postupně představují diskrétní hodnoty úrovně nabití baterie a stavu paliva v nádrži. Z obrázku je patrné, že cesta z počátečního uzlu $s = 0$ do cílového uzlu $t = 2$ je tvořena posloupností stavů $0(8, 40) \rightarrow 14(8, 10) \rightarrow 2(1, 15)$. Při tvorbě DH-grafu jsou do jednotlivých přechodů mezi uzly ukládány také uzly, skrz které v původním grafu cesta prochází. Z těchto dat je poté možné rekonstruovat celou cestu, jež pro případ grafu na obr. 7 povede přes uzly: $0, 10, 14$ a 2.



Obr. 8: DH graf ke grafu na obr. 7

3.4.3 Rozšíření DH heuristiky (DHE)

Tato metoda vychází z DH algoritmu popsaného výše v podkapitole 3.4.2. Diskrétní aproximací reálných hodnot elektrické energie a paliva je způsobeno, že pomocí DH metody řešení není vždy možné získat optimální hodnoty dobíjení a dotankování v jednotlivých uzlech, ačkoliv výsledná trasa být optimální může. Postup výpočtu algoritmem DHE probíhá tak, že se nejprve získá optimální trasa s využitím algoritmu DH a následně se vytvoří podgraf, tvořený pouze uzly a hranami na optimální cestě. Na získaném podgrafu se provede výpočet matematického modelu popsaného v podkapitole 3.4.1. Vzhledem k tomu, že podgraf je zpravidla mnohem menší, než originální graf, je výpočet matematického modelu velmi rychlý.

3.4.4 Heuristika nejkratší cesty (SP)

Tato heuristika je založená na nalezení nejkratší cesty mezi dvěma uzly a následnou optimalizací pouze rozhodnutí týkající se dobíjení a tankování. Nalezení nejkratší cesty je provedeno obdobně, jako v podkapitole 3.2. Nejprve je vytvořen *meta-network* grafu, skládající se pouze z počátečního a cílového uzlu a uzlů, které obsahují dobíjecí a nebo čerpací stanici. Poté je v *meta-networku* nalezena nejkratší cesta za pomocí Dijkstrova algoritmu. Na závěr se vytvoří subgraf původního grafu, který je tvořený nejkratší možnou cestou a provede se na něm výpočet matematického modelu, obdobně jako v podkapitole 3.4.3.

4 SOFTWARE IMPLEMENTACE

V této kapitole je představena aplikace *pathfinder*, která je praktickým výstupem této práce. Program slouží ke generování simulačního prostředí a jeho následnému prohledávání. Simulační prostředí zde představuje dopravní síť s dobíjecími a čerpacími stanicemi, ve které se pohybuje model elektromobilu. Jak simulační prostředí, tak parametry modelu jsou uživatelsky nastavitelné. Na následujících stránkách jsou popsány technologie použité ke zhotovení software a samotné uživatelské prostředí aplikace.

4.1 Použité technologie

Aplikace byla implementována s využitím programovacího jazyka `Python` doplněným o několik dostupných knihoven, jako např. `NetworkX`, `tkinter`, `matplotlib`, atd. Pro řešení složitějších matematických modelů byl do programu zakomponován řešitel `Gurobi`. Na dalších řádcích této podkapitoly jsou použité technologie podrobněji popsány.

4.1.1 Programovací jazyk `Python`

`Python` [18] je interpretovaný objektově orientovaný programovací jazyk umožňující dynamické psaní programu, práci s moduly, třídami, výjimkami a dynamickými datovými typy. Jednou z dalších výhod tohoto programovacího jazyka je velmi přehledná a intuitivní syntaxe. `Python` je možné použít na většině operačních systémů, včetně `Windows`, `macOS`, `Linux` a mnoha dalších unixových variantách.

Distribuce `Pythonu` zahrnuje bohatou standardní knihovnu modulů, které pokrývají oblasti jako např. internetové protokoly, zpracování textových řetězců, software inženýrství a rozhraní operačních systémů. V případě potřeby modulů jiného charakteru, je k dispozici ke stažení rozsáhlé množství knihoven poskytovaných třetími stranami.

Jazyk je velmi univerzální a lze jej aplikovat na rozsáhlou oblast problémů. Jeho použití je vhodné zejména na tvorbu webových aplikací, tvorbu uživatelských rozhraní, řešení matematických modelů, vizualizaci dat, apod. Díky své jednoduchosti je také vhodným programovacím jazykem pro začátečníky [17].

`NetworkX`

`NetworkX` [16] je knihovna umožňující tvorbu a práci s grafy. Podporuje jak grafy prosté, tak multigrafy, orientované grafy a smyčky. Grafy je možné náhodně generovat, vytvářet z předem zadaných dat, importovat a exportovat. Uzly mohou být reprezentovány různými datovými typy, jako např. textem nebo obrázky. Dále do

uzlů lze, stejně jako do hran, ukládat různá data a vlastnosti. Knihovna poskytuje také několik funkcí a algoritmů, mezi které patří Delaunayova triangulace, prohledávací algoritmy, funkce pro převedení grafu na jeho rovinné zobrazení, atd. Bohaté jsou také možnosti vizualizace grafů.

`tkinter`

Jeden z balíčku modulů dodávaných ve standardní knihovně je `tkinter` [18]. Tato knihovna slouží ke tvorbě grafického rozhraní. Nabízí k použití celou řadu widgetů v podobě tlačítek, vstupních textových polí, výběrových seznamů, apod. Knihovna dále disponuje celou řadou možností, jak s widgety pracovat, vzájemně je propojovat nebo jim přiřazovat úkoly a funkce.

Tvorba uživatelského rozhraní za pomocí této knihovny probíhá umisťováním jednotlivých widgetů pomocí mechanismu správy geometrie. V praxi může tento postup probíhat tak, že se vytvoří widget hlavního okna aplikace, do kterého se pak budou umisťovat další widgety dle potřeby.

`matplotlib`

Knihovna `matplotlib` [22] se používá zejména k vizualizaci různých dat, nejčastěji pomocí dvourozměrného grafu. Její možnosti využití však nejsou omezeny pouze na zmíněné uplatnění. Díky širokým možnostem nastavení zobrazení je `matplotlib` možné aplikovat na širokou škálu problémů. Lze pomocí ní vytvářet grafy různých druhů a tvarů, přidávat do zobrazení popisky v podobě textu nebo matematických rovnic, vykreslovat a dále pracovat s obrázky, atd.

4.1.2 `Gurobi Optimizer`

`Gurobi Optimizer` [12] je software určený k řešení složitých matematických modelů. Jedná se o jeden z nejrychlejších a nejúčinnějších nástrojů k řešení úloh matematického programování. Je možné pomocí něj řešit jak úlohy lineárního programování, tak úlohy celočíselného nebo kvadratického programování. Mezi metody výpočtu, které řešitel může použít patří např. simplexová metoda, metoda větví a mezí, metody sečných nadrovin, atd.

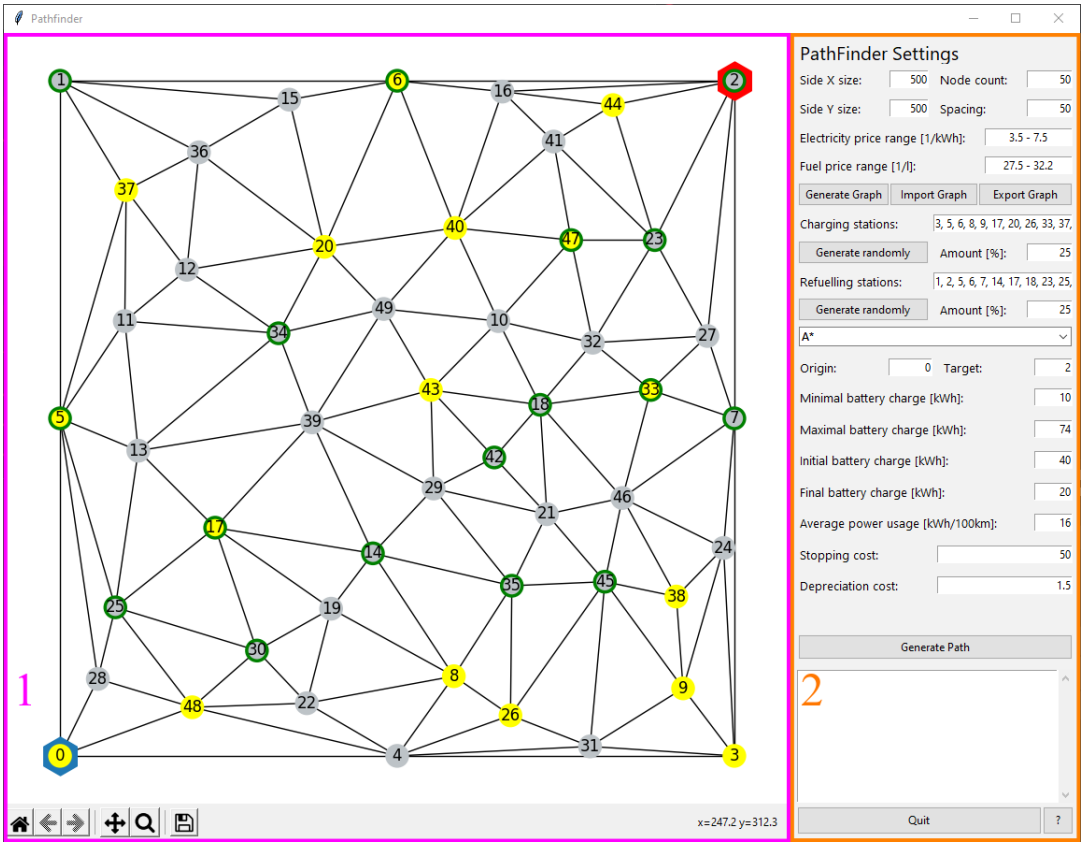
Software lze používat s využitím modelovacích jazyků, konkrétně `AMPL` či `GAMS`, nebo jej lze aplikovat za pomocí jednoho z mnoha podporovaných programovacích jazyků, jako `C`, `C++`, `C#`, `Java`, `Python`, `Visual Basic`, `MATLAB` nebo `R`. Zvláštní přínos pro uživatele nabízí volba programovacího jazyka `Python`, pro který je dostupná knihovna `Gurobi Python Environment` umožňující snazší práci s modelem a využití získaných výsledků v dalších aplikacích.

Aby bylo možné s `Gurobi` pracovat v plném rozsahu, je nutné vlastnit licenci tohoto software. Pro studeny a akademické pracovníky je dostupná bezplatná

licence, která svému uživateli umožňuje využívat všechny funkce, které software nabízí. Přístup k této licenci je nutný ke správnému fungování všech součástí aplikace *pathfinder*.

4.2 Grafické rozhraní

Uživatelské rozhraní se skládá ze dvou částí: panelu, kde probíhá vykreslování grafu a panelu nastavení. Pohled na celé uživatelské rozhraní je vyobrazen na obr 9.



Obr. 9: Uživatelské rozhraní

1 – Panel vykreslování grafu

Tato sekce grafického rozhraní slouží kromě vizualizace grafu a jeho dat také k interakci se získaným zobrazením.

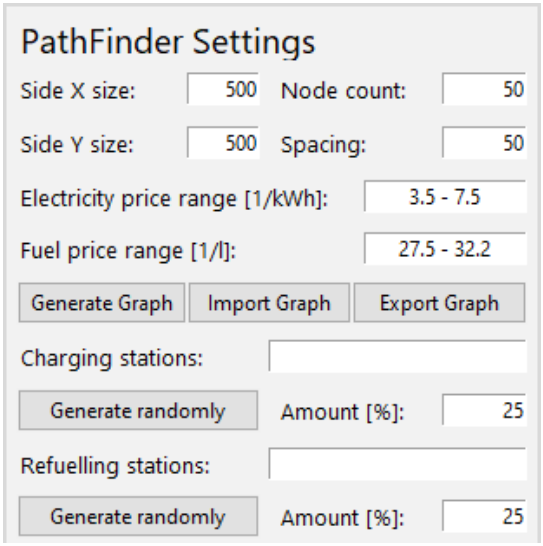
Uzly grafu zde mohou být dle jejich vlastností označeny několika různými barvami. Modré a červené ohraničení uzlu značí počáteční a cílový uzel, mezi kterými se hledá cesta, naopak zbarvení uvnitř uzlu reprezentuje výskyt dobíjecí (žlutá výplň) nebo čerpací (zelený kruh) stanice. Kliknutím kurzorem na uzel lze do něj umístit stanici. Dvojité kliknutí levým tlačítkem myši přidává dobíjecí stanici a dvojitý klik pravým tlačítkem do uzlu uloží stanici čerpací.

V dolní části panelu se nachází lišta s tlačítky (vlevo) a souřadnice současné polohy kurzoru myši v prostoru grafu (vpravo). Lišta s tlačítky je k vidění na obr. 10. Význam jednotlivých tlačítek v pořadí zleva doprava je následující. První tlačítko slouží k navrácení grafu do výchozího pohledu. Další dvě tlačítka nesou příkazy *zpět* a *vpřed* a umožňují přepínat mezi jednotlivými pohledy grafu. Následující tlačítko umožňuje manipulovat s pohledem grafu prostřednictvím posunu kurzoru myši. Tlačítko se symbolem lupy umožňuje přiblížení grafu a poslední tlačítko lze použít k uložení současného pohledu do souboru.

Obr. 10: Detail tlačítek ovládání pohledu grafu

2 – Panel nastavení

Tento panel obsahuje funkce k nastavení jednotlivých parametrů grafu a implementovaných heuristik. Nachází se zde také log událostí, tlačítko vypnutí software a tlačítko zobrazující manuál k programu.



Obr. 11: Detail části panelu nastavení týkající se parametrů grafu

V části panelu nastavení vyobrazené na obr. 11 se nastavují parametry spojené s tvorbou grafu sloužícího jako simulačního prostředí. Kolonky *Side X size* a *Side Y size* určují délky stran dvourozměrného prostoru, do kterého je možné generovat uzly grafu s náhodnou polohou. Vstup *Node count* udává, kolik uzlů bude do prostoru vloženo a *Spacing* vymezuje minimální rozestup mezi jednotlivými uzly. Pod těmito vstupy se postupně nachází kolonky pro nastavení intervalů cen za elektřinu a palivo. Při generování uzlů grafu se pak ceny dobíjení a paliva jednotlivých uzlů

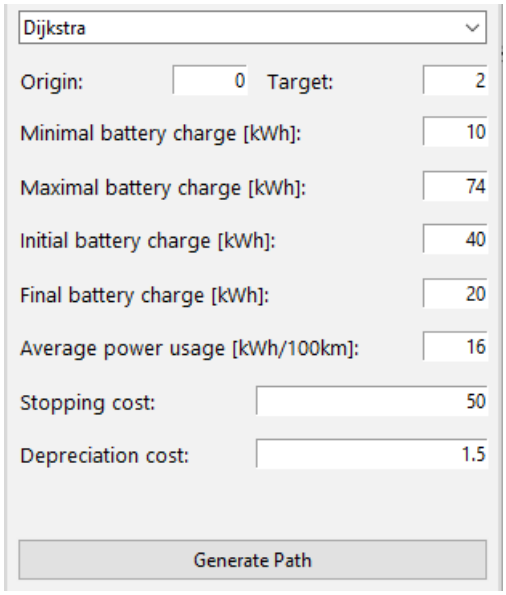
vyberou náhodně z těchto intervalů. Následuje řádek se třemi tlačítky, kde tlačítko *Generate graph* spustí vytváření grafu dle parametrů nastavených výše. Tlačítko *Import graph* slouží k načtení grafu ze souboru a tlačítko *Export graph* naopak slouží k uložení vygenerovaného grafu do souboru v počítači. Pod těmito tlačítky se nachází vstup pro zadávání čísel uzlů, do kterých se umístí dobíjecí stanice. Dobíjecí stanice je rovněž možné generovat náhodně pomocí tlačítka níže, jež náhodně umístí dobíjecí stanici do množství uzlů specifikované v políčku *Amount* v sekci pro nastavení výskytu dobíjecích stanic. Pod tímto tlačítkem se vyskytuje sekce pro nastavení výskytu čerpacích stanic s obdobnou funkčností.

V horní části panelů na obrázcích 12a a 12b se nachází výběrový seznam s heuristikami pro optimalizaci trasy. Na výběr je v seznamu celkem šest metod výpočtu, z toho dvě jsou určené pro optimalizaci trasy elektromobilů:

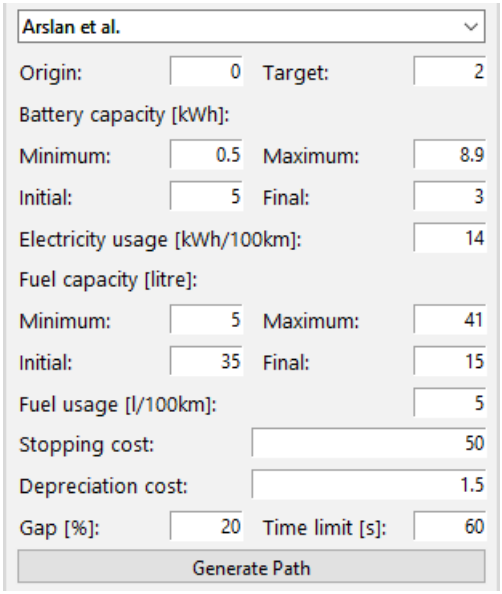
- Optimalizace trasy pomocí Dijkstrova algoritmu a dynamického programování (viz kapitola 3.2)
- Optimalizace trasy pomocí algoritmu A* a dynamického programování (viz kapitola 3.3)

Zbylé čtyři heuristiky je možné využít na plánování cesty jak elektromobilů, tak plug-in hybridů:

- Optimalizace trasy využitím matematického modelu (viz kapitola 3.4.1)
- Optimalizace trasy za pomocí algoritmu DH (viz kapitola 3.4.2)
- Optimalizace trasy užitím algoritmu DHE (viz kapitola 3.4.3)
- Optimalizace trasy algoritmem SP (viz kapitola 3.4.4)



(a) Nastavení parametrů pro Dijkstrův algoritmus

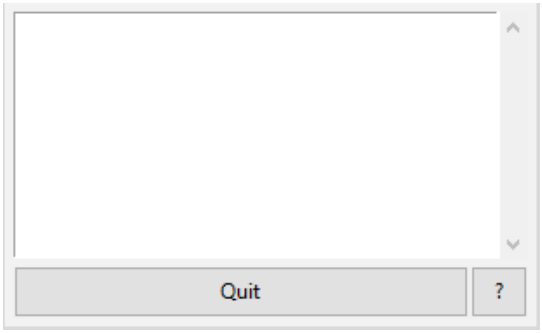


(b) Nastavení parametrů matematického modelu

Obr. 12: Ukázka sekcí pro nastavení parametrů heuristik

Dále se v těchto sekcích vyskytují kolonky sloužící pro nastavení parametrů příslušné heuristiky. Vstupy níže v těchto sekcích odpovídají parametrům, které byly zavedené v kapitolách jednotlivých heuristik. Tlačítkem *Generate path* v dolní části nastavení každé heuristiky se spustí výpočet. Obdobně vypadají panely nastavení pro ostatní dostupné algoritmy.

V dolní části panelu nastavení, jež je zároveň k vidění na obr. 13 se vyskytuje log událostí, kde se průběžně vypisují zprávy týkající se chodu programu nebo výsledky provedených výpočtů. Dvojklik levým tlačítkem myši v prostoru logu otevře nové okno s jeho dosavadním obsahem, čímž je zajištěné pohodlné čtení informací, které log poskytuje. Pod tímto logem se nachází tlačítko *Quit*, které inicializuje vypnutí programu a tlačítko se symbolem otazníku, jehož zmáčknutí způsobí otevření okna s manuálem programu.



Obr. 13: Detail logu událostí

5 ZHODNOCENÍ VÝSLEDKŮ

V této kapitole budou postupně představeny výsledky porovnání popsaných metod. Bylo provedeno srovnání využití různých algoritmů pro vytvoření *meta-networku*, porovnání metod optimalizace pro elektromobily a porovnání jednotlivých metod uvedených v kapitole 3.4.

5.1 Porovnání algoritmů pro tvorbu *meta-networku* grafu

Tato podkapitola obsahuje výsledky srovnání využití Dijkstrova algoritmu, algoritmu A* a Floyd-Warshallova algoritmu pro vytvoření *meta-networku* z různě velkých výchozích grafů obsahující dobíjecí stanice.

Experiment byl prováděn vždy na třech různých grafech pro jeden počet uzlů N v grafu. Pro každý graf byly náhodně vygenerovány čtyři varianty procentuálního zastoupení dobíjecích stanic a to 25%, 50%, 75% a 100%. V konečné aplikaci není nutné vytvářet *meta-network* pro případ výskytu dobíjecí stanice ve všech uzlech, ale zde je pro účely porovnání účinnosti jednotlivých algoritmů tento případ zaveden. Výjimkou je algoritmus DH, kde je z důvodů požadavků algoritmu *meta-network* generován odlišným způsobem, který pro ostatní algoritmy není použit z důvodu lehce větší časové náročnosti, avšak jeho výhodou je rychlý výpočet pro větší množství stanic.

Na každý graf byl použit model s nastavením parametrů odpovídajících tabulce 1. Parametry kapacity baterie a spotřeby energie byly inspirovány vlastnostmi vozidla *Tesla Model 3* [1].

Tab. 1: Nastavení parametrů pro tvorbu *meta-networku*

$\underline{P}$	$\overline{P}$	P_s	P_t	ε
10kWh	74kWh	74kWh	20kWh	16kWh/100km

V tabulkách 2, 3 a 4 jsou vyobrazeny získané hodnoty časů výpočtu pro počet uzlů ve výchozím grafu $N = 100$, $N = 500$ a $N = 1000$. Z hodnot lze vyčíst, že na testovaných grafech si nejrychleji počínal Dijkstrův algoritmus a naopak výpočet pomocí algoritmu A* trval nejdéle. Z tohoto důvodu nebyly časy výpočtu algoritmem A* pro varianty grafů s větším počtem uzlů měřeny, neboť tak vysoká časová náročnost v porovnání s mnohem nižšími časy Dijkstrova a Floyd-Warshallova algoritmu postrádá praktické využití a není potřeba ji dále prozkoumávat. Dále je možné vidět, že časová náročnost algoritmu Floyd-Warshall je až na menší výchylky obdobná pro různé hustoty dobíjecích stanic a jeden počet uzlů. Pro grafy s menším počtem uzlů

a vyšším zastoupením dobíjecích stanic byl, jak je vidět v tabulce 2, efektivnější algoritmus Floyd-Warshall, avšak s minimálním rozdílem oproti Dijkstrovému algoritmu, který je i u větších grafů schopen sestavit *meta-network* v kratším časovém intervalu.

Tab. 2: Porovnání jednotlivých algoritmů tvorby *meta-networku* pro $N = 100$

N = 100				
Množství stanic:	25%	50%	75%	100%
Dijkstrův algoritmus	0.13s	0.25s	0.41s	0.55s
Algoritmus A*	0.53s	1.69s	3.81s	6.95s
Algoritmus Floyd-Warshall	0.36s	0.37s	0.38s	0.41s

Tab. 3: Porovnání jednotlivých algoritmů tvorby *meta-networku* pro $N = 500$

N = 500				
Množství stanic:	25%	50%	75%	100%
Dijkstrův algoritmus	11.76s	21.45s	33.45s	46.74s
Algoritmus A*	6m 31s	>10min	neměřeno	
Algoritmus Floyd-Warshall	58.61s	57.80s	57.24s	60.97s

Tab. 4: Porovnání jednotlivých algoritmů tvorby *meta-networku* pro $N = 1000$

N = 1000				
Množství stanic:	25%	50%	75%	100%
Dijkstrův algoritmus	1m 22s	2m 37s	4m 14s	5m 20s
Algoritmus A*	neměřeno			
Algoritmus Floyd-Warshall	8m 6s	8m 2s	8m 9s	8m 11s

Získané výsledky vychází z vlastností a funkčnosti jednotlivých algoritmů. Pro tvorbu *meta-networku* byla použita varianta Dijkstrova algoritmu, která se nezastaví v okamžiku nalezení cesty do cílového uzlu a místo toho ohodnotí vzdálenosti do všech ostatních uzlů z jednoho výchozího. Toto ohodnocení a následné hledání cesty mezi stanicemi se děje pro N uzlů. Oproti tomu algoritmus Floyd-Warshall vždy hledá cesty mezi všemi uzly ať už je tam dobíjecí stanice, či nikoliv, což vysvětluje přibližně stejné časy pro různé hodnoty poměru stanic. Algoritmus A* hledá

pomocí heuristické funkce nejkratší cestu mezi dvěma uzly, což se při generování *meta-networku* děje pro každé dvojice příslušných uzlů. Není vyloučeno, že je možné provézt úpravy algoritmů A* a Floyd-Warshall tak, aby se snížila jejich časová náročnost, avšak cílem bylo porovnat výchozí podobu algoritmů a jejich využití, nikoliv je rozšiřovat.

5.2 Porovnání algoritmů optimalizace trasy elektromobilů

Na následujících řádcích této podkapitoly proběhne porovnání výsledků metod optimalizace trasy pro elektromobily. Porovnání proběhne mezi Dijkstrovým algoritmem, algoritmem A* a matematickým modelem popsaným v článku [3].

Optimalizace algoritmy prohledávání grafu probíhala následujícím způsobem. Nejprve byl vygenerován *meta-network* pro zadaný graf, poté byla nalezena nejkratší cesta Dijkstrovým/A* algoritmem a následně se provedla optimalizace nabíjení na získané trase pomocí algoritmu dynamického programování. Jelikož nalezené nejkratší cesty mezi dvěma uzly byly stejné pro oba algoritmy, jsou i výsledky týkající se optimalizace nabíjení totožné pro tyto dvě metody. Stejně tak čas generování *meta-networku* je pro tyto dva algoritmy jednotný, neboť jej stačí vytvořit pouze jednou a postupně jej prohledat oběma metodami. Výpočet algoritmem dynamického programování probíhal zvlášť pro oba algoritmy, a tak je výsledný čas představený zde průměrem naměřených hodnot pro obě iterace.

Aby bylo možné posoudit výsledky nabíjení v jednotlivých uzlech, získané pomocí dynamického programování, byla provedena také optimalizace pomocí matematického modelu. To bylo provedeno ve dvou iteracích pro každé prostředí a model vozidla. Nejprve byl matematický model spuštěn pro obdobný čas, za který získaly výsledek metody prohledávání grafu a pro druhou iteraci byl procesu výpočtu matematickým modelem stanoven maximální čas na půl hodinu.

Optimalizace probíhala pro tři různé modely elektromobilů, jejichž parametry odpovídají vozidlům *Tesla Model Y*, *Peugeot e-2008* a *Mercedes-Benz EQC* a jsou k vidění v tabulce 5 [1]. Výpočty byly prováděny pro tři náhodně vytvořená prostředí různých velikostí.

Tab. 5: Parametry modelů elektromobilu

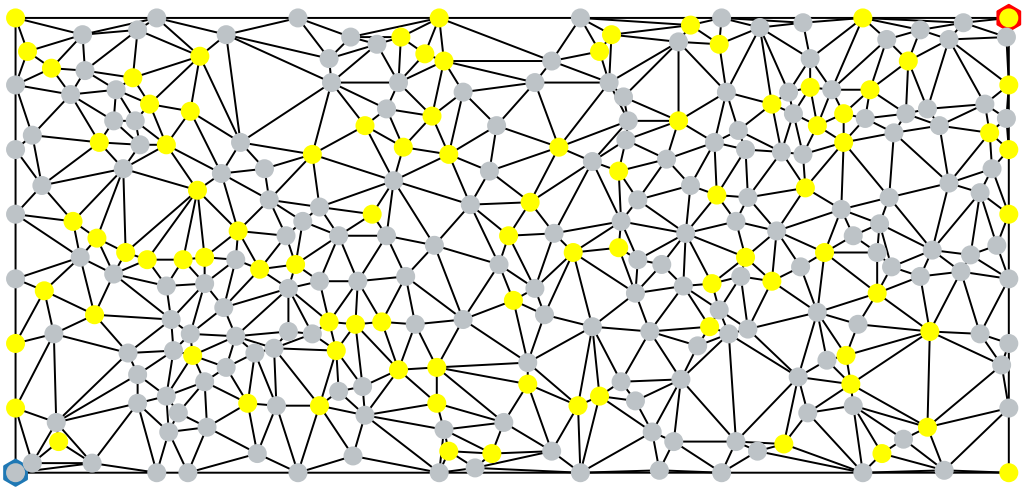
	$\underline{P}$	$\overline{P}$	P_s	P_t	ε
Tesla Model Y	2.5	75	38	38	14.4
Peugeot e-2008	5	50	25	25	18.3
Mercedes-Benz EQC	8	88	44	44	21.3

Česká Republika

První z porovnání bylo realizováno na prostředí, které se svými vlastnostmi (rozlohou, počtem větších měst) podobá České Republice. Parametry vytvořeného grafu jsou vyobrazeny v tabulce 6 a samotný graf lze spatřit na obr. 14.

Tab. 6: Parametry prostředí *„Česká Republika“*

Poměr stran	N	Minimální rozestup	Dobíjecí stanice	c^{st}	c^{dep}
415 × 190km	273	8km	30%	50	1



Obr. 14: Graf prostředí *„Česká Republika“*

Výsledky pro jednotlivé metody optimalizace pro zmíněné prostředí jsou k nalezení v tabulce 7, kde C^e vyjadřuje celkovou cenu dobíjení, C^{dep} vyjadřuje celkovou cenu za opotřebení vozidla a zároveň vzhledem k $c^{dep} = 1$ udává také celkovou ujetou vzdálenost vozidlem, C^{st} označuje celkovou cenu zastavení a C je hodnota celkové ceny odpovídající součtu všech dílčích cenových složek. Dále jsou v tabulce dostupné časové údaje zpracování jednotlivých výpočtů, kde t označuje časové hodnoty běhu heuristik, t_{m-n} udává čas, za který byl vytvořen *meta-network* a t_{DP} je čas, za který byla provedena optimalizace na získané trase s využitím algoritmu dynamického programování. Součet časů t, t_{m-n} a t_{DP} udává celkový čas výpočtu pro algoritmy prohledávání grafu, kdežto pro výpočty matematickým modelem je celkový čas trvání výpočtu obsažen ve sloupci t. Čísla obsažená v závorkách vedle označení výpočtu matematickým modelem udávají dosaženou *gap* výpočtu, jejíž hodnota je určena jako $gap = |z_P - z_D|/|z_P|$, kde z_P je nalezená dolní mez nejlepšího řešení a z_D je hodnota aktuálně nejlepšího řešení, jež nalezl řešitel `Gurobi`.

Tab. 7: Výsledky metod optimalizace trasy elektromobilů v prostředí *„Česká Republika“*

	Algoritmus	C^e	C^{dep}	C^{st}	C	t	t_{m-n}	t_{DP}
Tesla	Dijkstra	262.50	477.44	100	839.94	4.99ms	2.61s	13.38s
	A*					4.21ms		
	MM (42.6%)	254.29	483.24	100	837.53	15s		
	MM (41.5%)	254.39	482.78	100	837.17	30m		
Peugeot	Dijkstra	345.76	479.99	150	975.75	4.50ms	2.51s	4.58s
	A*					2.90ms		
	MM (50.7%)	343.82	479.99	150	973.81	7s		
	MM (48.7%)	343.82	479.99	150	973.81	30m		
Mercedes	Dijkstra	388.83	477.44	100	966.27	4.89ms	2.43s	15.98s
	A*					3.88ms		
	MM (50.1%)	387.28	477.44	100	964.72	18s		
	MM (49.1%)	380.44	483.24	100	963.68	30m		

Na první pohled je z výsledků uvedených v tabulce 7 patrné, že algoritmus A* našel nejkratší cestu v získaném *meta-networku* v kratším časovém okamžiku, než Dijkstrův algoritmus. Avšak vzhledem k tomu, že oba algoritmy nalezli nejkratší trasu v řádech jednotek milisekund, nehraje tento čas kromě porovnání těchto dvou algoritmů větší roli. Dále je možné si všimnout, že celkové ceny jsou u všech tří modelů vozidel obdobné pro všechny testované heuristiky. Odchylku hodnot získaných prostřednictvím dynamického programování od hodnot spočtených matematickým modelem je možné zdůvodnit diskretizací úrovní nabití algoritmem dynamického programování. Lze říci, že pro tento případ jsou dostačující výsledky získané algoritmy prohledávání grafu a dynamického programování a matematického modelu s časem výpočtu v řádech vteřin, neboť matematický model s vyhrazeným časem půl hodiny nenašel výsledky natolik rozdílné, aby bylo možné ospravedlnit jeho časové náklady. Z výsledků lze také vyčíst, že modely vozidel jely celkem po čtyřech různých trasách, což bylo způsobeno různými parametry vozidel.

U výsledků modelu vozidla typu *Tesla Model Y* lze vidět, že volené trasy se liší pro všechny testované heuristiky. Algoritmy prohledávání grafu nalezly nejkratší cestu, avšak ne zcela optimální z hlediska cen dobíjení v jednotlivých uzlech. Matematický model provádějící půlhodinový výpočet nalezl s velmi zanedbatelným rozdílem kratší cestu, avšak s mírně zvýšenými náklady na dobíjení.

Model vozidla typu *Peugeot e-2008* má pro testované prostředí nejvyšší cenové náklady, což je možné zdůvodnit jeho poměrem kapacity baterie a spotřeby

energie oproti ostatním vozidlům. Všechny testované heuristiky volily stejnou cestu, která byla zároveň zvolena pouze tímto modelem. Z faktorů vysoké celkové ceny a volené cesty lze předpokládat, že vybraná trasa nebyla pro ostatní dva typy vozidel zcela optimální a byla pro tento model vozidla zvolena z důvodu jeho parametrů.

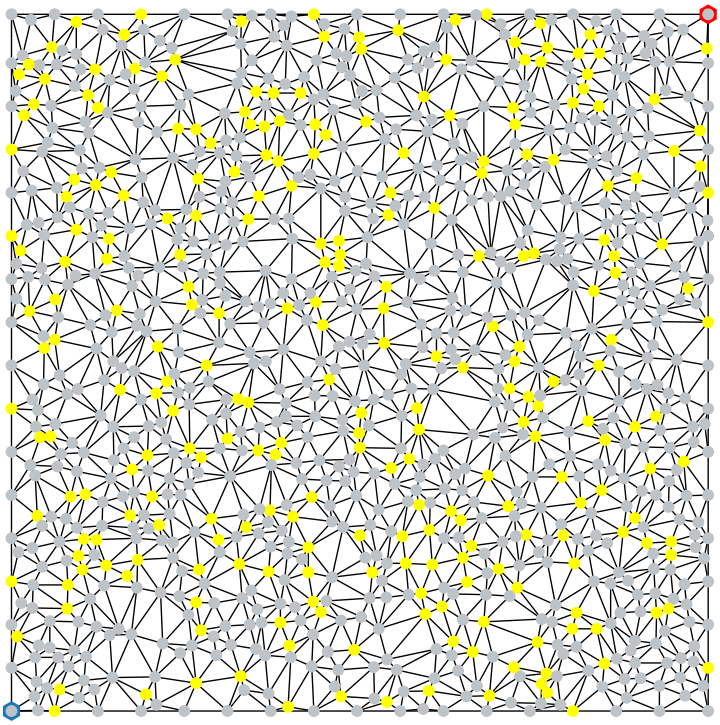
U hodnot získaných pro model vozidla typu *Mercedes-Benz EQC* je možné vidět, že jak algoritmy prohledávání grafu, tak matematický model s časem výpočtu 18s zvolily stejnou nejkratší trasu, jako algoritmy prohledávání grafu u vozidla značky *Tesla*. Oproti tomu matematický model s vymezeným časem 30m zvolil jako optimální cestu shodnou s cestou nalezenou matematickým modelem s časem výpočtu 15s rovněž pro vozidlo značky *Tesla*.

Simulační prostředí s počtem uzlů $N = 1000$

Pro další experiment bylo zvoleno prostředí s počtem uzlů $N = 1000$ a rozlohou, jejíž velikost pravděpodobně překračuje standardní délku cesty, kterou uživatelé elektromobilů podniknou. Cílem však bylo otestovat efektivitu testovaných heuristik ve větším prostředí. Ostatní parametry tohoto grafu jsou uvedeny v tabulce 8 a vytvořené prostředí je k vidění na obr. 15.

Tab. 8: Parametry simulačního prostředí s počtem uzlů $N = 1000$

Poměr stran	N	Minimální rozestup	Dobíjecí stanice	c^{st}	c^{dep}
1000 × 1000km	1000	15km	25%	50	1



Obr. 15: Graf prostředí s počtem uzlů $N = 1000$

Výsledky porovnání jednotlivých modelů vozidel pro toto testované prostředí lze vidět v tabulce 9. Celkové ceny u tohoto experimentu se liší o významně větší rozdíly oproti předchozímu testování, které probíhalo na podstatně menším grafu a to jak rozlohou, tak počtem uzlů. Celkové ceny nabývají různě velkých hodnot pro všechny testované modely a stejně tak trasy volené jednotlivými modely vozidel nebyly stejné pro žádné dva modely.

Tab. 9: Výsledky metod optimalizace trasy elektromobilů v simulačním prostředí s počtem uzlů $N = 1000$

	Algoritmus	C^e	C^{dep}	C^{st}	C	t	t_{m-n}	t_{DP}
Tesla	Dijkstra	1261.51	1501.44	200	2962.94	32.8ms	1m 26s	9.20s
	A*					35.3ms		
	MM (44.63%)	979.66	1539.18	200	2718.84	1m 35s		
	MM (44.10%)	979.66	1539.18	200	2718.84	30m		
Peugeot	Dijkstra	1504.23	1526.74	450	3480.97	27.2ms	1m 23s	5.90s
	A*					30.6ms		
	MM (55.10%)	1324.67	1575.31	500	3399.98	1m 29s		
	MM (53.23%)	1334.55	1563.00	450	3347.56	30m		
Mercedes	Dijkstra	1808.89	1503.72	300	3612.60	28.3ms	1m 28s	13.93s
	A*					31.0ms		
	MM (–%)	cesta nenalezena				1m 40s		
	MM (53.38%)	1437.20	1550.65	300	3287.84	30m		

Výsledky modelu vozidla *Tesla Model Y* lze označit jako nejlepší z hlediska hodnoty celkové ceny, což odpovídá rozdílů mezi parametry tohoto vozidla a parametry zbylých dvou porovnávaných modelů. Poměrně velký rozdíl lze vidět, mezi hodnotami cen získaných algoritmem dynamického programování a oběma iteracemi matematického modelu. V tomto případě se jako nejefektivnější volba jeví výpočet pomocí matematického modelu, jemuž bylo umožněno hledat optimální řešení kratší časový okamžik, neboť hodnoty jednotlivých cen získané matematickým modelem s časem zpracování půl hodiny jsou totožné.

V případě modelu vozidla *Peugeot e-2008* byla každou testovanou heuristikou zvolena jiná trasa. Rozdíly mezi hodnotami celkových cen pro jednotlivé metody však nejsou tak rozdílné, jako tomu bylo u předchozího modelu vozidla. V tomto případě byl nalezen nejlepší výsledek matematickým modelem, který měl na výpočet půl hodinu, avšak získaná hodnota se příliš neliší od hodnoty získané matematickým modelem v kratším časovém okamžiku a lze těžko říct, zda-li je vyvážena podstatně delším časem výpočtu.

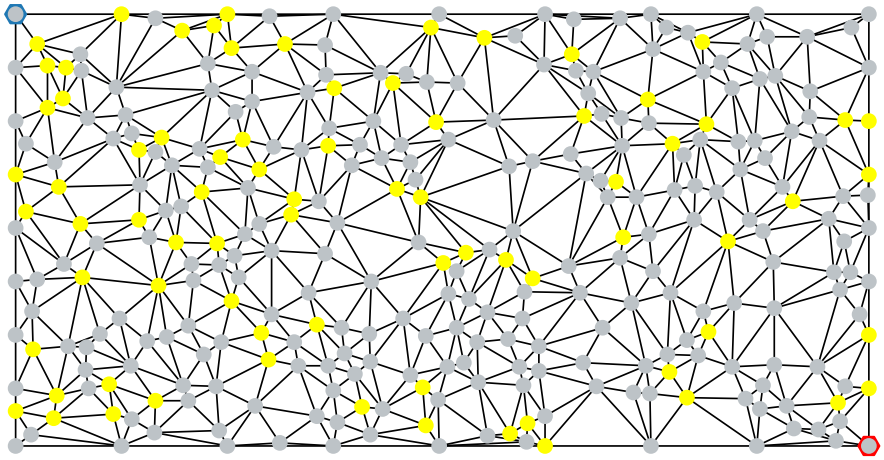
U modelu vozidla *Mercedes-Benz EQC* jsou výsledky dostupné pouze pro metodu optimalizace pomocí grafových algoritmů a dynamického programování a matematický model s výpočetní dobou půl hodiny, neboť v zadaném kratším časovém okamžiku nebyl matematický model schopen nalézt jakoukoliv přípustnou cestu. Cesta nalezena algoritmy prohledávání grafu je jednou z nejkratších nalezených tras v tomto prostředí, avšak dobíjení na volené trase zcela jistě nelze považovat za optimální, což se i přes shodnou četnost nabíjení na obou volených cestách promítlo velkým rozdílem mezi celkovými cenami.

Finsko

Posledním prostředí, vyobrazené na obr. 16, na kterém byly testované modely elektromobilů je graf svou rozlohou připomínající Finsko a trasa mezi dvěma zadanými body je svou délkou podobná cestě z města Tampere do vesnice Kilpisjärvi. Detailní parametry prostředí jsou k vidění v tabulce 10.

Tab. 10: Parametry simulačního prostředí *„Tampere-Kilpisjärvi“*

Poměr stran	N	Minimální rozestup	Dobíjecí stanice	c^{st}	c^{dep}
420 × 830km	309	15km	25%	50	1



Obr. 16: Graf prostředí *„Tampere-Kilpisjärvi“* (otočený o 90°)

Myšlenkou tohoto experimentu bylo provézt porovnání modelů na grafu s počtem uzlů podobným, jako v prostředí *„České Republiky“*, avšak s větší rozlohou a vzdálenostmi mezi jednotlivými uzly. Z výsledků v tabulce 11 lze vyčíst, že celkové ceny získané grafovými algoritmy jsou velmi podobné cenám spočtených matematickým modelem s časem na výpočet omezeným na několik vteřin. Matematický model s časem trvání optimalizace půl hodiny pak nalezl pro všechny tři testované

modely vozidel lepší řešení, avšak ne s tak velkým rozdílem jako tomu bylo u hodnot celkových cen v tabulce 9.

Model vozidla *Tesla Model Y* dosáhl obdobných výsledků za pomocí kombinace grafových algoritmů a dynamického programování a matematického modelu omezeným na kratší čas výpočtu, lišící se pouze diskretizací úrovní nabití.

Trasy zvolené pro model vozidla *Peugeot e-2008* se liší pro všechny testované metody. Lze si také povšimnout poměrně kratšího času optimalizace trasy algoritmem dynamického programování, oproti zbylým dvěma modelům vozidel.

Pro model elektromobilu *Mercedes-Benz EQC* byla algoritmy prohledávání grafu a matematickým modelem s omezeným časem výpočtu zvolena stejná trasa, jako u vozu typu *Peugeot e-2008*. Hodnoty celkových cen se pro tyto dvě heuristiky liší opět pouze z důvodu diskretizace. Ceny spočtené všemi použitými metodami jsou stejně jako v případech dvou předchozích modelů vozidel velmi podobné.

Tab. 11: Výsledky metod optimalizace trasy elektromobilů v simulačním prostředí *„Tampere-Kilpisjärvi“*

	Algoritmus	C^e	C^{dep}	C^{st}	C	t	t_{m-n}	t_{DP}
Tesla	Dijkstra	610.05	979.10	150	1739.15	6.91ms	3.04s	18.66s
	A*					2.17ms		
	MM (43.1%)	607.18	979.10	150	1736.29	21.07s		
	MM (41.3%)	577.45	986.60	150	1714.05	30m		
Peugeot	Dijkstra	905.77	997.17	300	2202.94	3.80ms	2.90s	6.35s
	A*					2.68ms		
	MM (55.0%)	892.76	1013.33	300	2206.09	9.05s		
	MM (49.5%)	854.46	1005.57	300	2160.03	30m		
Mercedes	Dijkstra	930.27	979.10	200	2109.37	6.07ms	3.21s	22.61s
	A*					2.81ms		
	MM (52.9%)	926.66	979.10	200	2105.76	25.07s		
	MM (50.5%)	890.64	986.60	200	2077.24	30m		

5.3 Porovnání různých stupňů aproximací u metod řešení DH a DHE

V této podkapitole byla porovnána efektivita algoritmů DH a DHE pro modely PHEV při nastavení stupně aproximace na hodnotu 1, 2, 4 a 8. Porovnávání bylo provedeno v náhodně generovaných grafech s počtem uzlů $N = 50$, $N = 100$ a $N = 150$, jejichž kompletní specifikace lze nalézt v tabulce 12. Byly testovány tři různá nasta-

vení parametrů modelu, které byly inspirovány plug-in hybrid vozidly *Škoda Superb iV*, *Audi A7 55 TFSIe* a *Ford Kuga* [1]. Nastavení parametrů modelu jednotlivých vozidel je k vidění v tabulce 13.

Tab. 12: Parametry prostředí pro porovnávání algoritmů DH a DHE s různými hodnotami stupňů aproximace

Poměr stran	N	Minimální rozestup	Dobíjecí stanice	Čerpací stanice	c^{st}	c^{dep}
500 × 500km	50	50km	30%	30%	50	1
500 × 1000km	100	30km	15%	30%	50	1
800 × 800km	150	30km	25%	50%	50	1

Tab. 13: Parametry modelů plug-in hybrid vozidel

	$\underline{P}$	$\overline{P}$	P_s	P_t	ε	$\underline{G}$	$\overline{G}$	G_s	G_t	ρ
Škoda Superb iV	1	13.0	5	7	15.4	5	50	25	25	3.7
Audi A7 55 TFSIe	1	14.1	5	7	10.8	5	52	26	26	4.1
Ford Kuga	1	14.4	5	7	26.0	5	45	22	22	7.1

V tabulce 14 jsou výsledky porovnání algoritmů DH a DHE pro náhodně vytvořený graf s počtem uzlů $N = 50$. Oproti tabulkám s výsledky porovnání elektromobilů v předchozí podkapitole, zde přibyl sloupec C^g, v němž se nachází hodnoty celkové ceny doplnění paliva. Na rozdíl od způsobu tvorby *meta-networku* popsaného v kapitole 3.2 je *meta-network* pro použití s algoritmem DH vytvořený pouze nalezením nejkratších cest mezi příslušnými uzly bez kontroly, zda-li je vozidlo schopno vzdálenosti daných cest ujet. Proto je možné pro všechny modely plug-in hybrid vozidel použít totožný *meta-network*, který stačí vytvořit pouze jednou a čas potřebný pro jeho vygenerování se nachází ve sloupci s označením t_{m-n}. Časové hodnoty pro algoritmy DH uvedené ve sloupci t udávají čas potřebný pro vytvoření DH-grafu a nalezení optimální cesty v něm. Hodnoty sloupce t v řádcích s výsledky získanými algoritmem DHE udávají čas, za který byl matematický model schopen optimalizovat hodnoty nabíjení na trase dříve získané algoritmem DH se stejným stupněm aproximace.

Z hodnot lze na první pohled vidět, že všechny heuristiky volily stejnou cestu pro všechny modely vozidel, což je zapříčiněno malými rozměry grafu. Ze získaných výsledků je také možné vidět, že až na model vozidla *Ford Kuga* se hodnoty celkových cen spočtených algoritmy DH zlepšují s rostoucím stupněm aproximace. Celkové ceny dosažené algoritmy DHE pro modely vozidel *Škoda Superb iV* a *Audi A7 55 TFSIe* se z důvodu použití jedné cesty ničím neliší. Pro vozidlo značky *Ford*

jsou nalezené hodnoty dokonce stejné pro všechny testované algoritmy. V případě tohoto grafu malých rozměrů by pro získání optimálních výsledků bylo dostačující nalezení optimální cesty algoritmem DH1 a následná optimalizace dobíjení baterie a tankování paliva na dané trase pomocí algoritmu DHE. Lze předpokládat, že pro jiné grafy obdobných rozměrů by stačilo totéž.

Tab. 14: Výsledky porovnání různých stupňů aproximace algoritmů DH a DHE pro počet uzlů grafu $N = 50$

	Algoritmus	C^e	C^g	C^{dep}	C^{st}	C	t	t_{m-n}
Škoda	DH1	60.86	722.25	727.41	50	1560.52	0.04s	
	DHE1	60.86	674.51	727.41	50	1512.78	+0.12s	
	DH2	92.54	622.76	727.41	100	1542.71	0.20s	
	DHE2	60.86	674.51	727.41	50	1512.78	+0.15s	
	DH4	92.54	622.08	727.41	100	1542.03	1.76s	
	DHE4	60.86	674.51	727.41	50	1512.78	+0.17s	
	DH8	92.54	621.74	727.41	100	1541.69	17.17s	
	DHE8	60.86	674.51	727.41	50	1512.78	+0.15s	
Audi	DH1	66.44	738.13	727.41	50	1581.99	0.04s	55.4ms
	DHE1	165.10	472.82	727.41	150	1515.33	+0.11s	
	DH2	102.47	610.81	727.41	100	1540.69	0.19s	
	DHE2	165.10	472.82	727.41	150	1515.33	+0.08s	
	DH4	102.47	610.10	727.41	100	1539.98	1.49s	
	DHE4	165.10	472.82	727.41	150	1515.33	+0.09s	
	DH8	165.10	487.01	727.41	150	1529.52	15.55s	
	DHE8	165.10	472.82	727.41	150	1515.33	+0.11s	
Ford	DH1	105.18	1265.88	727.41	100	2198.47	0.04s	
	DHE1	105.18	1265.88	727.41	100	2198.47	+0.10s	
	DH2	105.18	1265.88	727.41	100	2198.47	0.16s	
	DHE2	105.18	1265.88	727.41	100	2198.47	+0.11s	
	DH4	105.18	1265.88	727.41	100	2198.47	1.21s	
	DHE4	105.18	1265.88	727.41	100	2198.47	+0.13s	
	DH8	105.18	1265.88	727.41	100	2198.47	13.37s	
	DHE8	105.18	1265.88	727.41	100	2198.47	+0.07s	

Hodnoty spočtené vybranými heuristikami pro $N = 100$ jsou vypsány v tabulce 15. I z těchto výpočtů lze vidět, že rozdíl celkových cen získaných algoritmy DH8 a DHE8 není tak znatelný oproti ostatním stupňům aproximace. Pro model vozidla značky *Škoda* jsou hodnoty dokonce opět stejné pro heuristiky DHE1 a DHE8.

Rozdíl cen u modelů vozidel *Audi* a *Ford* mezi zmiňovanými algoritmy je způsoben nalezením rozdílné cesty algoritmem DH8, avšak za cenu větší časové náročnosti.

Tab. 15: Výsledky porovnání různých stupňů aproximace algoritmů DH a DHE pro počet uzlů grafu $N = 100$

	Algoritmus	C^e	C^g	C^{dep}	C^{st}	C	t	t_{m-n}
Škoda	DH1	88.12	1220.10	1170.22	150	2628.43	0.12s	
	DHE1	78.89	1134.79	1170.22	100	2483.89	+0.13s	
	DH2	44.06	1282.69	1170.22	100	2596.97	0.40s	
	DHE2	78.89	1134.79	1170.22	100	2483.89	+0.12s	
	DH4	22.03	1264.51	1170.22	100	2556.76	3.12s	
	DHE4	78.89	1134.79	1170.22	100	2483.89	+0.16s	
	DH8	21.88	1227.14	1170.22	100	2519.24	33.60s	
	DHE8	78.89	1134.79	1170.22	100	2483.89	+0.18s	
Audi	DH1	96.20	1289.38	1170.22	150	2705.79	0.07s	0.22s
	DHE1	86.12	1214.42	1170.22	100	2570.75	+0.16s	
	DH2	96.20	1310.40	1170.22	100	2676.82	0.39s	
	DHE2	86.12	1214.42	1170.22	100	2570.75	+0.18s	
	DH4	96.20	1256.91	1170.22	100	2623.33	3.09s	
	DHE4	86.12	1214.42	1170.22	100	2570.75	+0.21s	
	DH8	120.84	1155.51	1169.78	150	2596.12	34.99s	
	DHE8	148.68	1138.14	1169.78	100	2556.59	+0.21s	
Ford	DH1	85.72	2646.52	1324.91	200	4257.15	0.09s	
	DHE1	140.07	2509.84	1324.91	200	4174.82	+0.14s	
	DH2	85.72	2641.84	1324.91	200	4252.47	0.36s	
	DHE2	140.07	2509.84	1324.91	200	4174.82	+0.12s	
	DH4	85.72	2640.34	1324.91	200	4250.97	2.77s	
	DHE4	140.07	2509.84	1324.91	200	4174.82	+0.11s	
	DH8	54.34	2637.96	1301.89	200	4194.19	28.54s	
	DHE8	54.34	2586.95	1301.89	200	4143.18	+0.10s	

Porovnání hodnot jednotlivých modelů a heuristik pro graf s počtem uzlů $N = 150$ je k vidění v tabulce 16. Stejně jako v případě počtu uzlů $N = 50$, i v tomto případě jsou nejlepší výsledky dosaženy pomocí algoritmu DHE1, ačkoliv zde je rozdíl mezi časovou náročností stupňů aproximace 1 a 8 mnohem více viditelný. U modelu vozidla *Škoda Superb iV* je hodnota celkové ceny dosažena algoritmem DHE1 dokonce nejlepší ze všech použitých heuristik pro tento vůz.

Tab. 16: Výsledky porovnání různých stupňů aproximace algoritmů DH a DHE pro počet uzlů grafu $N = 150$

	Algoritmus	C^e	C^g	C^{dep}	C^{st}	C	t	t_{m-n}
Škoda	DH1	50.16	1314.01	1217.05	100	2681.23	0.44s	
	DHE1	50.16	1255.38	1217.05	100	2622.59	+0.13s	
	DH2	50.16	1314.01	1217.05	100	2681.23	2.27s	
	DHE2	50.16	1255.38	1217.05	100	2622.59	+0.13s	
	DH4	50.16	1314.01	1217.05	100	2681.23	18.28s	
	DHE4	50.16	1255.38	1217.05	100	2622.59	+0.10s	
	DH8	50.16	1296.89	1217.37	100	2664.42	5m 15s	
	DHE8	50.16	1291.49	1217.37	100	2659.02	+0.18s	
Audi	DH1	115.64	1300.13	1238.35	100	2754.12	0.47s	1.09s
	DHE1	322.70	791.82	1238.35	250	2602.87	+0.16s	
	DH2	134.12	1242.43	1217.05	150	2743.60	2.26s	
	DHE2	288.06	946.08	1217.05	200	2651.19	+0.17s	
	DH4	103.02	1237.94	1217.05	150	2708.01	17.64s	
	DHE4	288.06	946.08	1217.05	200	2651.19	+0.16s	
	DH8	301.98	878.37	1238.35	250	2668.70	4m 42s	
	DHE8	322.70	791.82	1238.35	250	2602.87	+0.20s	
Ford	DH1	62.27	2521.92	1244.01	150	3978.20	0.41s	
	DHE1	58.17	2469.40	1244.01	150	3921.58	+0.19s	
	DH2	70.73	2468.80	1231.63	200	3971.15	1.83s	
	DHE2	55.94	2444.41	1231.63	200	3931.97	+0.16s	
	DH4	70.73	2464.53	1231.63	200	3966.88	15.28s	
	DHE4	55.94	2444.41	1231.63	200	3931.97	+0.17s	
	DH8	58.17	2473.55	1244.01	150	3925.72	2m 41s	
	DHE8	58.17	2469.40	1244.01	150	3921.58	+0.18s	

Z výsledků obsažených v tabulkách 14, 15 a 16 lze vyvodit, že nejefektivnější volba stupně aproximace je v mnohých případech 1. To ovšem platí pouze pokud bude použit algoritmus DHE, neboli DH v kombinaci s matematickým modelem, neboť pomocí samotného algoritmu DH1 nelze z důvodu diskretizace zaručit optimální hodnoty nabíjení a tankování na nalezené trase. V případě, kdy není možné použít k výpočtu matematický model je tedy vhodnější použít algoritmus DH se stupněm aproximace 2 nebo 4, neboť časová náročnost u stupně aproximace 8 rapidně narůstá pro vyšší počty uzlů v grafu, jak lze vidět v tabulce 16.

5.4 Porovnání optimalizačních metod pro PHEV vozidla

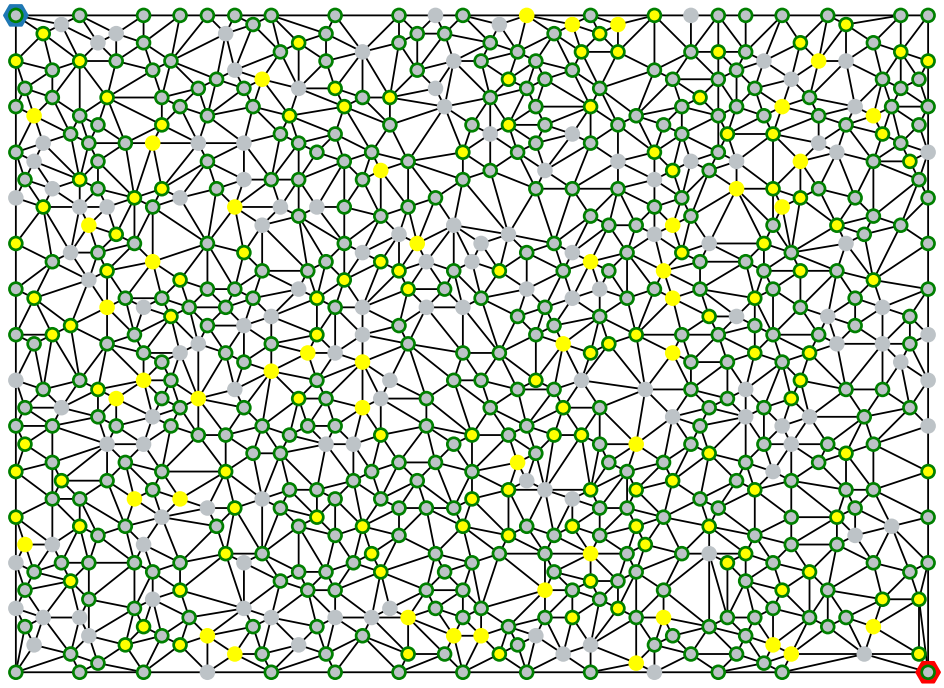
V této podkapitole budou postupně představeny výsledky porovnání jednotlivých optimalizačních metod PHEV vozidel pro tři simulační prostředí s různými parametry. Pro výpočet budou použity modely vozidel uvedené v předchozí podkapitole, jejichž parametry se nachází v tabulce 13. Ve všech prostředích a pro každý model vozidla byly postupně testovány následující heuristiky: matematický model pro PHEV, algoritmy DH a DHE pro stupně aproximace 1, 2 a 4 a algoritmus SP.

Jihomoravský kraj

Prvním z porovnávaných prostředí v této podkapitole je graf svými parametry podobný Jihomoravskému kraji a je k vidění na obr. 17. Jeho parametry lze spatřit v tabulce 17.

Tab. 17: Parametry prostředí *„Jihomoravský kraj“*

Poměr stran	N	Minimální rozestup	Dobíjecí stanice	Čerpací stanice	c^{st}	c^{dep}
72 × 100km	672	2km	25%	70%	50	1



Obr. 17: Graf prostředí *„Jihomoravský kraj“* (otočený o 90°)

Výsledky tohoto experimentu jsou k dispozici v tabulce 18. Stejně jako v předchozím případě bylo i zde možné pro všechny úrovně aproximace a modely vozidel

použít u algoritmů DH jeden stejný *meta-network*. Naproti tomu u algoritmu SP bylo nutné jej generovat pro každý model zvlášť, neboť je použit obdobný způsob tvorby jako tomu bylo u grafových algoritmů výše. Pro matematický model, stejně jako v případě u porovnání elektromobilů, *meta-network* generován nebyl a jeho čas výpočtu byl nastaven na půl hodinu.

Tab. 18: Výsledky metod optimalizace trasy PHEV v simulačním prostředí *„Jihomoravský kraj“*

	Algoritmus	C^e	C^g	C^{dep}	C^{st}	C	t	t_{m-n}
Škoda	MM (55.8%)	42.95	71.05	129.14	50	293.13	30m	–
	DH1	34.17	566.61	132.01	50	782.79	14.30s	70.12s
	DHE1	48.72	72.61	132.01	50	303.35	+0.14s	
	DH2	40.21	76.56	133.42	100	350.19	1m 22s	
	DHE2	43.18	74.84	133.42	50	301.43	+0.19s	
	DH4	není dostupné						
	DHE4							
	SP	48.72	69.03	128.70	50	296.45	0.20s	67.27s
Audi	MM (49.9%)	46.45	32.88	129.10	50	258.43	30m	–
	DH1	32.42	594.94	132.01	50	809.37	14.80s	70.12s
	DHE1	46.96	52.10	132.01	50	281.07	+0.15s	
	DH2	37.21	70.06	133.42	100	340.69	1m 19s	
	DHE2	45.27	42.91	133.42	50	271.60	+0.17s	
	DH4	není dostupné						
	DHE4							
	SP	45.51	52.10	128.70	50	276.31	0.20s	74.37s
Ford	MM (68.4%)	54.41	176.27	128.70	50	409.38	30m	–
	DH1	38.20	481.62	132.01	50	701.83	15.16s	70.12s
	DHE1	54.41	183.14	132.01	50	419.56	+0.16s	
	DH2	47.51	171.06	129.10	100	447.68	1m 12s	
	DHE2	47.51	184.41	129.10	50	411.03	+0.16s	
	DH4	není dostupné						
	DHE4							
	SP	54.41	176.27	128.70	50	409.38	0.18s	66.51s

Už na první pohled si lze z výsledků všimnout, že nejsou dostupné výsledky pro algoritmy DH4 a DHE4, což bylo způsobeno samovolným ukončením výpočtu z důvodu nadměrné paměťové náročnosti. Dále lze u všech tří modelů vozidel vidět nadměrné výchylky cen za tankování paliva u algoritmů DH1 a jejich promítnutí na výsledných cenách oproti ostatním heuristikám. Tyto výchylky nastaly z důvodu

nedostatečně jemné diskretizace a jde vidět, že oproti grafům s menším počtem uzlů uvedených v předchozí kapitole, jsou získané ceny tímto algoritmem zcela jistě neoptimální. Bez použití algoritmu DHE je tak použití úrovně aproximace 1 zcela neefektivní. Mnohem lépe si vedl algoritmus SP, který během nejkratšího času získal pro všechny modely velmi dobré výsledky.

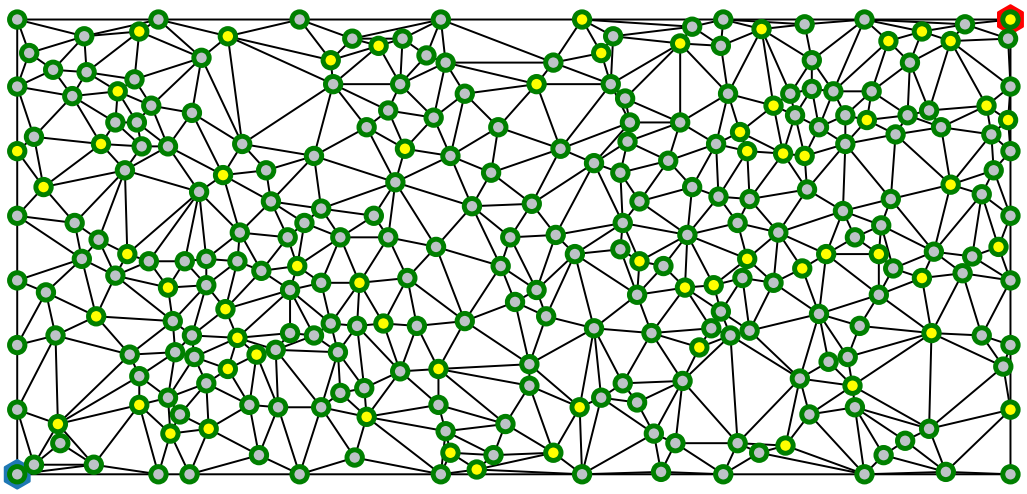
U modelu vozu *Škoda Superb iV* byla nejlepší celková cena dosažena výpočtem matematického modelu, avšak algoritmus SP dosáhl ceny velmi podobné, zvolil nejkratší cestu a zároveň více prioritizoval cestování za využití baterie, než ostatní heuristiky. Algoritmus DHE2 také nedosáhl špatných výsledků a umožnil oproti algoritmu DH2 pouze jedno zastavení, čímž snížil cenu na stejné trase o významnou část.

Model vozidla *Audi A7 55 TFSIe* dosáhl v porovnání v tomto prostředí nejnižší ceny, a to s využitím matematického modelu. Jde také vidět, že docílil většího využití baterie než paliva, což se povedlo také algoritmu DHE2, avšak ne s tak markantním rozdílem.

Výsledky modelu vozidla *Ford Kuga* se od ostatních dvou modelu liší mnohem větší cenou za tankování paliva. To lze vysvětlit velmi velkou spotřebou elektrické energie baterie tohoto vozidla. Oproti ostatním vozidlům zde byla nejkratší cesta nalezena algoritmem SP stejná, jako cesta zvolená matematickým modelem. Cena těchto dvou heuristik je rovněž nejnižší, a tak lze algoritmus SP považovat za nejefektivnější metodu optimalizace pro toto vozidlo v daném prostředí.

Česká republika

Další experiment byl proveden v podobném prostředí, jako bylo uvedené v podkapitole 5.2 s rozdílem rozložení dobíjecích stanic a přítomnosti čerpacích stanic. Verze prostředí pro PHEV je vyobrazena na obr. 18 a jeho parametry mohou být nalezeny v tabulce 19.



Obr. 18: Graf prostředí *„Česká republika“* pro PHEV modely vozidel

Tab. 19: Parametry prostředí *„Česká republika“* pro PHEV modely vozidel

Poměr stran	N	Minimální rozestup	Dobíjecí stanice	Čerpací stanice	c^{st}	c^{dep}
415 × 190km	273	8km	25%	100%	50	1

Hodnoty získané použitými heuristikami pro prostředí *České Republiky* se nachází v tabulce 20. Díky přítomnosti čerpacích stanic ve všech uzlech grafu, nebylo nutné pro algoritmus SP vytvářet nejprve *meta-network*, což jej zároveň učinilo nejrychlejším algoritmem pro toto prostředí.

Tab. 20: Výsledky metod optimalizace trasy PHEV v simulačním prostředí *„Česká republika“*

	Algoritmus	C^e	C^g	C^{dep}	C^{st}	C	t	t_{m-n}
Škoda	MM (51.4%)	44.24	427.72	477.44	50	999.40	30m	–
	DH1	60.42	551.82	477.44	50	1139.69	3.13s	6.56
	DHE1	44.24	427.72	477.44	50	999.40	+0.17s	
	DH2	44.24	442.85	477.44	100	1064.53	17.51s	
	DHE2	44.24	427.72	477.44	50	999.40	+0.11s	
	DH4	44.24	442.85	477.44	100	1064.53	6m 20s	
	DHE4	44.24	427.72	477.44	50	999.40	+0.16s	
	SP	44.24	427.72	477.44	50	999.40	0.17s	–
Audi	MM (50.1%)	105.36	291.11	477.44	100	973.91	30m	–
	DH1	51.59	579.41	477.44	50	1158.44	3.19s	6.56
	DHE1	105.36	291.11	477.44	100	973.91	+0.17s	
	DH2	150.51	194.68	492.89	200	1038.08	17.38s	
	DHE2	154.59	169.50	492.89	150	966.98	+0.14s	
	DH4	150.51	194.68	492.89	200	1038.08	4m 39s	
	DHE4	154.59	169.50	492.89	150	966.98	+0.15s	
	SP	105.36	291.11	477.44	100	973.91	0.20s	–
Ford	MM (66.2%)	101.10	760.79	477.44	100	1439.33	30m	–
	DH1	68.90	869.65	482.02	100	1520.57	3.42s	6.56
	DHE1	120.89	794.48	482.02	100	1497.39	+0.21s	
	DH2	68.90	869.65	482.02	100	1520.57	15.98s	
	DHE2	120.89	794.48	482.02	100	1497.39	+0.24s	
	DH4	179.26	672.54	479.44	150	1481.24	4m 3s	
	DHE4	179.26	670.29	479.44	150	1478.99	+0.2s	
	SP	101.10	760.79	477.44	100	1439.33	0.19s	–

U vozidla značky *Škoda* jsou celkové ceny stejné pro algoritmus SP, všechny algoritmy DHE a také matematický model. Vzhledem k rychlosti algoritmu SP, je jeho použití nejefektivnější. Algoritmus DH4 dosáhl stejných výsledků jako výkonnostně méně náročný algoritmus DH2. Lze si také všimnout velkého rozdílu mezi celkovými cenami za doplnění elektřiny a paliva, ačkoliv z výsledků ostatních modelů vozidel lze odvodit, že na zvolené cestě se možnosti doplnit elektřinu naskytovaly. Je pravděpodobné, že suma ušetřená menším odběrem paliva za cenu většího množství zastávek by se projevila negativně z důvodu nárůstu ceny za zastavení a výsledná cena by tak byla větší, a proto byla zvolena pouze jedna zastávka.

Heuristiky dosáhly u modelu vozidla značky *Audi* podstatně zajímavějších výsledků, než u ostatních vozidel, neboť nejlevnější cena byla nalezena algoritmem DHE2. Stejná cena byla nalezena také algoritmem DHE4, ale jeho časová náročnost je mnohonásobně vyšší. Tento model vozidla má také v tomto případě nejvyváženější poměr spotřeby energie baterie a paliva.

Pro model vozidla *Ford Kuga* opět nejlépe vychází algoritmus SP, jehož cesta dosahuje nejnižší ceny. Poměrně dobrý výsledek přinesl také algoritmus DHE4, který nejlépe zužitkoval baterii vozidla.

Cesta do Chorvatska

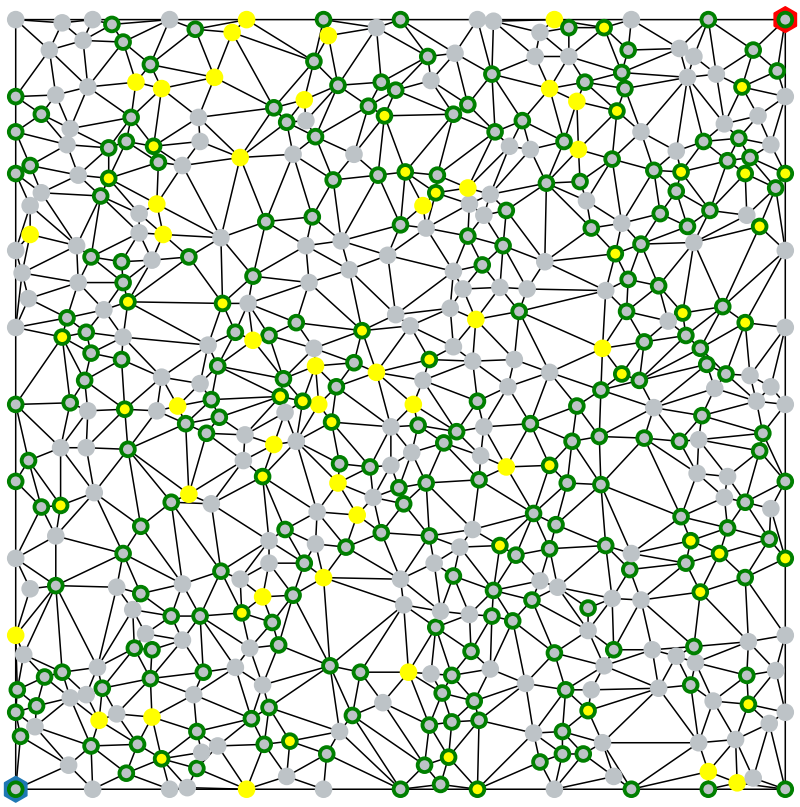
Posledním porovnáním představeným v této práci je experiment probíhající v prostředí, jehož délka úhlopříčky přibližně odpovídá vzdušné vzdálenosti vedoucí z Moravskoslezského kraje do chorvatského města Zadar a je možné jej spatřit na obr. 19. Parametry tohoto simulačního prostředí jsou k nahlédnutí v tabulce 21.

Tab. 21: Parametry prostředí *„cesta do Chorvatska“*

Poměr stran	N	Minimální rozestup	Dobíjecí stanice	Čerpací stanice	c^{st}	c^{dep}
480 × 480km	470	10km	15%	50%	50	1

Výsledné hodnoty cen a časů prostředí jsou dostupné v tabulce 22. Při pohledu na získané hodnoty si lze povšimnout větší frekvence zastávek u vozu značky *Audi*, oproti zbylým dvěma vozidlům. Tyto zastávky byly podniknuty pravděpodobně z důvodu častějšího dobíjení baterie, jelikož nízká spotřeba jak energie baterie, tak paliva modelu vozidla umožňuje dosáhnout lepší ceny i přes vysokou cenu za zastavení.

Heuristiky pro model vozidla *Škoda Superb iV* volily ve většině případů stejnou trasu. Optimální cestu našly algoritmy DHE1, DHE4, SP a matematický model. Algoritmus DHE2 nalezl cestu lišící se pouze o krátkou vzdálenost, avšak byla více



Obr. 19: Graf prostředí *„cesta do Chorvatska“*

zužitkovaná baterie vozidla. Za povšimnutí stojí také cena nalezená algoritmem DH1, která se příliš neliší od hodnoty optimální ceny.

V případě vozidla *Audi A7 55 TFSIe* byla optimální cesta nejrychleji nalezena algoritmem DHE1. V tomto případě také převažuje cena za dobíjení baterie nad cenou za doplnění paliva. Největší rozdíl mezi těmito cenami nastane při výběru a optimalizace trasy nalezené algoritmem DHE2. Algoritmus SP zde našel stejnou cestu, která byla u zbylých dvou modelů vozidel vybrána většinou heuristik jako optimální.

Nejlevnější celkové ceny u modelu vozidla *Ford Kuga* dosáhl matematický model, ne však s velkým rozdílem oproti ostatním heuristikám, které všechny volily stejnou trasu, jež je zároveň nejkratší nalezenou cestou mezi dvěma zadanými body.

Tab. 22: Výsledky metod optimalizace trasy PHEV v simulačním prostředí *„cesta do Chorvatska“*

	Algoritmus	C^e	C^g	C^{dep}	C^{st}	C	t	t_{m-n}
Škoda	MM (52.3%)	52.27	707.47	712.67	50	1522.41	30m	–
	DH1	48.44	692.40	712.67	100	1553.51	3.13s	
	DHE1	52.27	707.47	712.67	50	1522.41	+0.13s	
	DH2	51.95	685.50	715.53	100	1552.98	16.90s	17.53s
	DHE2	94.87	614.81	715.53	100	1525.21	+0.16s	
	DH4	48.44	686.83	712.67	100	1547.94	2m 16s	
	DHE4	52.27	707.47	712.67	50	1522.41	+0.15s	
	SP	52.27	707.47	712.67	50	1522.41	0.17s	16.86s
Audi	MM (49.8%)	279.63	198.24	719.59	250	1447.47	30m	–
	DH1	103.74	624.93	719.59	100	1548.26	3.42s	
	DHE1	279.63	198.24	719.59	250	1447.47	+0.18s	
	DH2	287.22	219.49	731.32	300	1538.02	20.77s	17.53s
	DHE2	291.16	203.08	731.32	250	1475.55	+0.28s	
	DH4	214.98	342.46	719.59	250	1527.04	3m 12s	
	DHE4	279.63	198.24	719.59	250	1447.47	+0.13s	
	SP	222.62	334.06	712.67	200	1469.35	0.19s	17.57s
Ford	MM (67.0%)	177.64	1157.63	723.39	150	2208.66	30m	–
	DH1	112.47	1273.50	712.67	150	2248.64	3.54s	
	DHE1	168.16	1151.99	712.67	200	2232.82	+0.18s	
	DH2	112.47	1273.50	712.67	150	2248.64	15.91s	17.53s
	DHE2	168.16	1151.99	712.67	200	2232.82	+0.16s	
	DH4	112.47	1262.98	712.67	150	2238.12	1m 56s	
	DHE4	168.16	1151.99	712.67	200	2232.82	+0.17s	
	SP	168.16	1151.99	712.67	200	2232.82	0.18s	17.00s

6 ZÁVĚR

Cíle této závěrečné práce byly popsat problematiku optimalizace plánování trasy pro elektromobily a následně pro tuto problematiku vytvořit praktickou implementaci.

V první části práce byla představena teorie grafů a algoritmy sloužící k jejich prohledávání společně s optimalizačními metodami, konkrétně celočíselným a dynamickým programováním. Následně byly popsány způsoby aplikace vybraných metod na řešenou problematiku.

Praktická část práce spočívala ve vytvoření grafického rozhraní, které uživateli umožní dle jeho zadaných parametrů vytvořit prostředí, v němž bude následně možné plánovat a posléze optimalizovat trasu pomocí několika metod. Do praktické části byly vybrány následující heuristické metody: Dijkstrův algoritmus, algoritmus A*, algoritmus dynamického programování a dále metody uvedené v článku [3], přesněji matematický model celočíselného programování, algoritmus DH, algoritmus DHE a algoritmus SP. Vytvořená aplikace uživateli umožňuje náhodně vytvořit graf se specifickými parametry, následně jej uložit do souboru a v případě potřeby graf ze souboru načíst. Dále je možné v programu upravovat nastavení modelů vozidel pro uvedené výpočetní metody. Zpětnou vazbu uživatel od aplikace získá jak pomocí vizualizace grafu společně s jeho parametry, tak výpisem událostí a podrobných výsledků v textové podobě přímo v uživatelském prostředí.

Na závěr proběhlo porovnání implementovaných metod v několika experimentech. Celkem byly prováděny čtyři typy experimentů. Nejprve byla srovnána efektivita využití grafových algoritmů při tvorbě *meta-networku*, kde se jako nejrychlejší způsob osvědčilo použití Dijkstrova algoritmu.

Další experiment se týkal srovnání různých metod pro optimalizaci trasy elektromobilů, kde ve většině případů obstály všechny testované algoritmy. V závislosti na zvyšujícím se počtu uzlů v grafu se také zvyšuje riziko, že nejkratší nalezená cesta grafovými algoritmy nebude optimální z hlediska cen dobíjení baterie vozidla. V takovém případě lze dosáhnout lepších výsledků použitím matematického modelu, který prozkoumává větší množství přípustných tras mezi zadanými body.

V následujícím experimentu se porovnávaly různé stupně aproximací pro algoritmy DH a DHE. Ze získaných hodnot vyplynulo, že velmi dobré výsledky lze získat algoritmem DHE se stupněm aproximace pouze 1 nebo 2. V případech, kdy je možné použít pouze algoritmus DH, lze lepších výsledků získat nastavením úrovně aproximace na hodnotu 4. Z důvodu velké časové náročnosti, která není vykoupena výrazně lepšími výsledky, nelze doporučit použití vyšších hodnot stupňů aproximace.

Poslední experiment porovnával metody optimalizace trasy plug-in hybrid vozidel představené v kapitole 3.4. Zde až na jednu situaci docílil nejlepších výsledků matematický model, avšak ostatní algoritmy dosáhly ve většině případů stejných,

nebo jen mírně odlišných výsledků. Po srovnání jednotlivých hodnot lze říci, že nejefektivnější použití matematického modelu je jako součást algoritmů DHE a SP, jelikož jejich aplikace dokáže významně snížit časové nároky za současného poskytnutí velmi dobrých výsledků.

Do vytvořené aplikace byla přidána také volba nahrání grafové struktury ze souboru ve formátu `.gpickle`, čímž je umožněno prohledávání i takových grafů, které byly vytvořeny mimo aplikaci. Tímto způsobem lze dosáhnout prakticky využitelnějších výsledků skrz import předem připraveného grafu skutečné dopravní sítě a následné optimalizace reálné cesty.

7 SEZNAM POUŽITÉ LITERATURY

[1] 24net s.r.o.: *fDrive*. [online], 2021, [cit. 2021-05-17].
URL `https://fdrive.cz/`

[2] Ahuja, R. K.; Magnanti, T. L.; Orlin, J. B.: *Network Flows: Theory, Algorithms, and Applications*. Spojené státy americké: Prentice-Hall, Inc, první vydání, 1993, ISBN 0-13-617S49-X.

[3] Arslan, O.; Yıldız, B.; Karaşan, O. E.: *Minimum cost path problem for Plug-in Hybrid Electric Vehicles. Transportation Research Part E: Logistics and Transportation Review*, ročník 80, 2015: s. 123–141, doi:10.1016/j.tre.2015.05.011.

[4] Bellman, R. E.: *Dynamic Programming*. Princeton University Press, 6 vydání, 1972, ISBN 0-691-07951-X.

[5] Bertsekas, D. P.: *Dynamic Programming and Optimal Control Volume I*. Athena Scientific, třetí vydání, 2005, ISBN 1-886529-26-4.

[6] Březina, T.; Dvořák, J.: *Algoritmy umělé inteligence*. Modernizace výukových materiálů a didaktických metod - CZ.1.07/2.2.00/15.0463.

[7] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press, třetí vydání, 2009, ISBN 978-0-262-03384-8.

[8] Dijkstra, E. W.: *A note on two problems in connexion with graphs. Numerische Mathematik*, ročník 1, č. 1, 1959: str. 269–271, doi:10.1007/bf01386390.

[9] Ding, D.; Li, J.; Tu, P.; aj.: *Electric Vehicle Charging Warning and Path Planning Method Based on Spark. IEEE Access*, ročník 8, 2020: s. 8543–8553, doi:10.1109/ACCESS.2020.2964307.

[10] Dvořák, J.: *Operační a systémová analýza*. Výukový materiál ve formě prezentací, 2018/2019.

[11] Floyd, R. W.: *Algorithm 97: Shortest path. Communications of the ACM*, ročník 5, č. 6, 1962: str. 345, doi:10.1145/367766.368168.

[12] Gurobi Optimization, LLC. All Rights Reserved.: *Gurobi Optimization*. [online], [cit. 2021-05-05].
URL `https://www.gurobi.com/`

[13] Hart, P.; Nilsson, N.; Raphael, B.: *A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics*, ročník 4, č. 2, 1968: str. 100–107, doi:10.1109/tssc.1968.300136.

[14] Khuller, S.; Malekian, A.; Mestre, J.: *To Fill or Not to Fill: The Gas Station Problem. ACM Transactions on Algorithms*, ročník 7, č. 3, 2011, doi:10.1145/1978782.1978791.

[15] Klapka, J.; Dvořák, J.: *Úvod do operační a systémové analysy.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2002.

[16] NetworkX developers.: *NetworkX: Network Analysis in Python.* [online], 2020, [cit. 2021-05-05].
URL `https://networkx.org/`

[17] Python Software Foundation.: *python.* [online], 2021, [cit. 2021-05-05].
URL `https://www.python.org/`

[18] Python Software Foundation.: *Python 3.9.5 documentation.* [online], 2021, [cit. 2021-05-05].
URL `https://docs.python.org/3/`

[19] Sekáč, O.: *Plánování cesty pro více robotů.* Diplomová práce, Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2020, 72 s., vedoucí práce: RNDr. Jiří Dvořák, CSc.

[20] Sioshansi, R.: *Modeling the Impacts of Electricity Tariffs on Plug-In Hybrid Electric Vehicle Charging, Costs, and Emissions. Operations Research*, ročník 60, č. 3, 2012: s. 506–516, doi:10.1287/opre.1120.1038.

[21] Sweda, T. M.; Dolinskaya, I. S.; Klabjan, D.: *Adaptive Routing and Recharging Policies for Electric Vehicles. Transportation Science*, ročník 51, č. 4, 2017: s. 1326–1348, doi:10.1287/trsc.2016.0724.

[22] The Matplotlib development team.: *matplotlib.* [online], 2021, doi:10.5281/zenodo.4649959, [cit. 2021-05-05].
URL `https://matplotlib.org/stable/index.html`

[23] Šeda, M.: *Teorie grafů.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, listopad 2003.

[24] Warshall, S.: *A Theorem on Boolean Matrices. Journal of the ACM*, ročník 9, č. 1, 1962: s. 11–12, doi:10.1145/321105.321107.

[25] Werner, T.: *Optimalizace*. [online], 2020, [cit. 2021-05-05].
URL `https://cw.fel.cvut.cz/wiki/_media/courses/b0b33opt/opt.pdf`

[26] Wilson, R. J.: *Introduction to Graph Theory*. Anglie: Longman Group Ltd, Čtvrté vydání, 1996, ISBN 0-582-24993-7.