

SRM Technische documentatie

ZU Data
YD Intelligence

door

Oscar Theelen(2204154)

2024

lectoraat Data Intelligence

begeleider

Marc Bertrand

| | |
|---|-----------|
| Inleiding | 3 |
| Beknopte uitleg | 3 |
| Handleiding | 4 |
| Benodigdheden | 4 |
| Programma gebruik | 6 |
| Functies | 7 |
| plan_circular_route_flower() | 7 |
| planner.calculate_start_point_index() | 8 |
| planner.calculate_leaf_nodes() | 8 |
| graph.insert_start_node_and_rearrange() | 9 |
| visualizer.visualize_leaf_points() | 9 |
| visualizer.visualize_best_path() | 10 |
| visualizer.visualize_elevations() | 10 |
| visualizer.visualize_surface_percentage() | 11 |
| analyzer.get_paths_and_path_lengths() | 11 |
| analyzer.min_length_routes_indeces() | 11 |
| analyzer.get_height_diffs() | 12 |
| analyzer.get_surface_diffs() | 12 |
| analyzer.calculate_percentage_hardened_surfaces() | 13 |
| analyzer.calculate_elevation_diff() | 13 |
| analyzer.get_score_only_elevation() | 14 |
| analyzer.get_score_only_surface() | 14 |
| analyzer.get_score_elevation_and_surface() | 15 |
| analyzer.remove_paths_above_stEEPNESS() | 15 |
| graph.export_GPX() | 16 |
| Scoring uitgelegd | 17 |
| Visualisaties | 18 |
| Gegenereerde punten | 18 |
| Beste pad | 18 |
| Hoogteverschillen | 19 |
| Verhard/onverhard | 19 |
| Gehele UI | 20 |
| Hoogteverschillen | 21 |
| Percentage verhard/onverhard | 21 |
| Hellingspercentage | 21 |
| Analyse | 22 |
| Tijdscomplexiteit | 22 |
| Nauwkeurigheid | 24 |
| Export naar GPX | 26 |
| Diagrammen | 27 |
| Usecase diagram | 27 |
| Sequence diagram | 28 |
| Activity diagram | 29 |
| Bronnen | 30 |

Inleiding

Binnen de smart route maker applicatie zitten een aantal functionaliteiten:

- Route maken op basis van start- en eindpunt
- Circulaire route maken op basis van afstand
- Circulaire route maken op basis van afstand en hoogtemeters
- Circulaire route op basis van afstand percentage verhard wegdek
- Circulaire route op basis van afstand percentage verhard wegdek en hoogtemeters
- Het invoeren van een maximaal hellingspercentage in de route

Punten die naar voren zullen komen zijn: functiebeschrijvingen(komen overeen met de doc strings in python), analyse van algoritme, UML diagrammen en keuzes binnen algoritme.

Beknopte uitleg

Het ontworpen programma/algoritme zorgt ervoor dat een gebruiker een circulaire route kan genereren, voor oude functionaliteiten, raadpleeg de oude technische documentatie(S.Vanwersch, S.Tabárá, J.Drenth, M.Meijers, 2022).

De gebruiker kan invoeren: Start coördinaten, afstand, klimhoogte, % verhard wegdek en maximaal hellingspercentage. het is belangrijk om op te merken dat deze variabelen richtlijnen zijn, het zal zeer zelden voorkomen dat een route 100% overeenkomt met de voorkeuren, dit is zelfs vrij onwaarschijnlijk. Start Coördinaat en afstand zijn verplichte velden, afstand, klimhoogte en helling zijn optioneel. Als deze niet worden ingevuld zal er met deze variabelen ook geen rekening worden gehouden.

Als de route generering wordt gestart worden er 64 mogelijke routes gegenereerd in een cirkelvormige structuur, in de *srm/Core/Static/Image* folder is er na uitvoer een grafiek hierover te zien. Elke route wordt gevormd door paden te berekenen tussen 5 punten per route om zo een circulaire route te maken. De hoeveelheid punten per route en het aantal routes is vrij aan te passen en valt mee te experimenteren door de source code aan te passen.

Nadat de routes zijn berekend, worden er een aantal beste gepakt op basis van afstand. Op basis van deze “beste” routes worden scores opgesteld met de andere ingevulde parameters. Hoe hoger de score, hoe hoger de afwijking van de input en dus hoe slechter. Hoe lager de score, hoe beter. Kijkende naar de scores pakt het algoritme de route met de laagste en dus beste score, dit is de uiteindelijke route. Stel de gebruiker voert helling in, worden alle routes die een helling bevatten die boven de ingevoerde waarde ligt niet meegerekend.

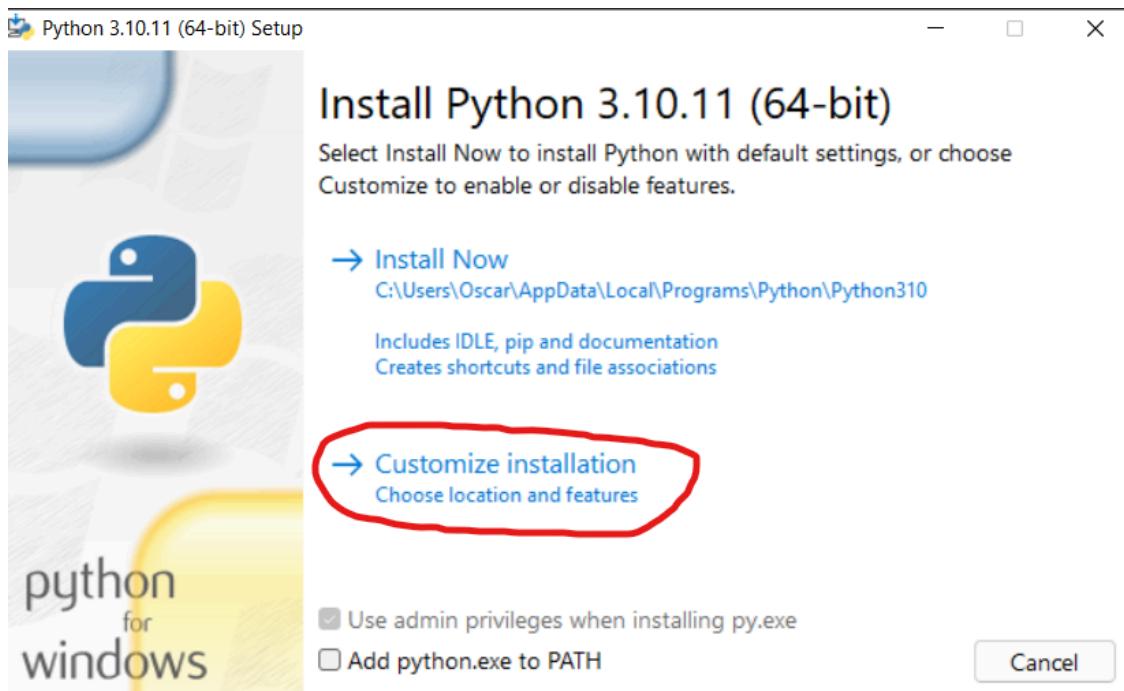
Handleiding

Benodigdheden

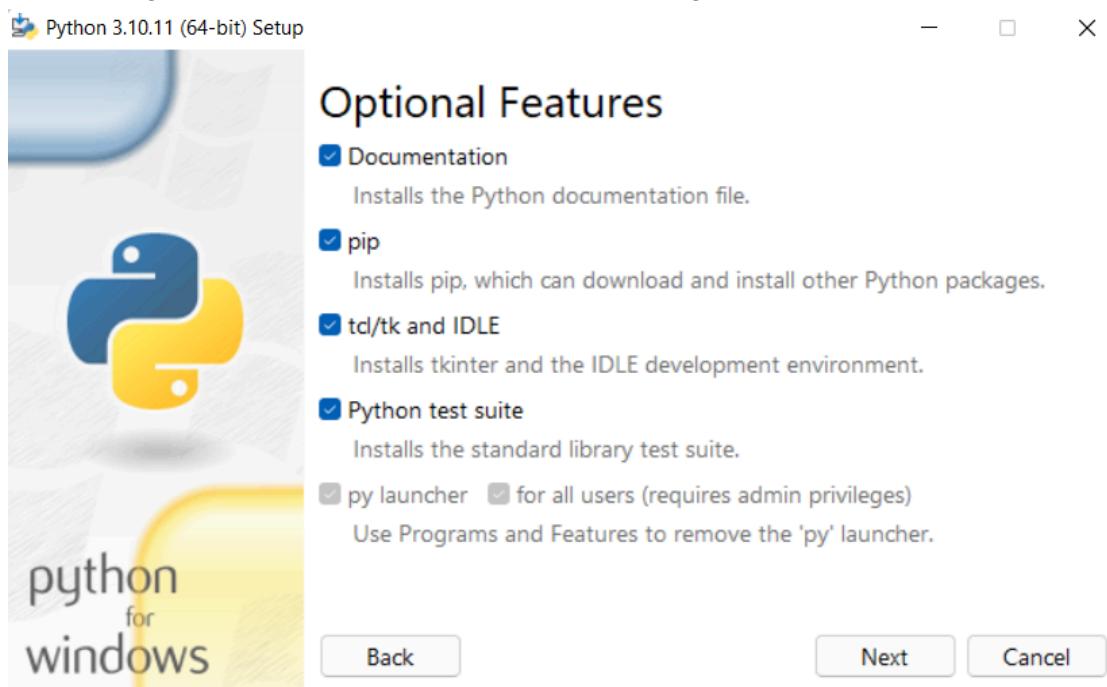
Het gebruik van andere versies kan problemen veroorzaken

- Python 3.10.11 [download](https://www.python.org/ftp/python/3.10.11/python-3.10.11-amd64.exe) correct versie
<https://www.python.org/ftp/python/3.10.11/python-3.10.11-amd64.exe>

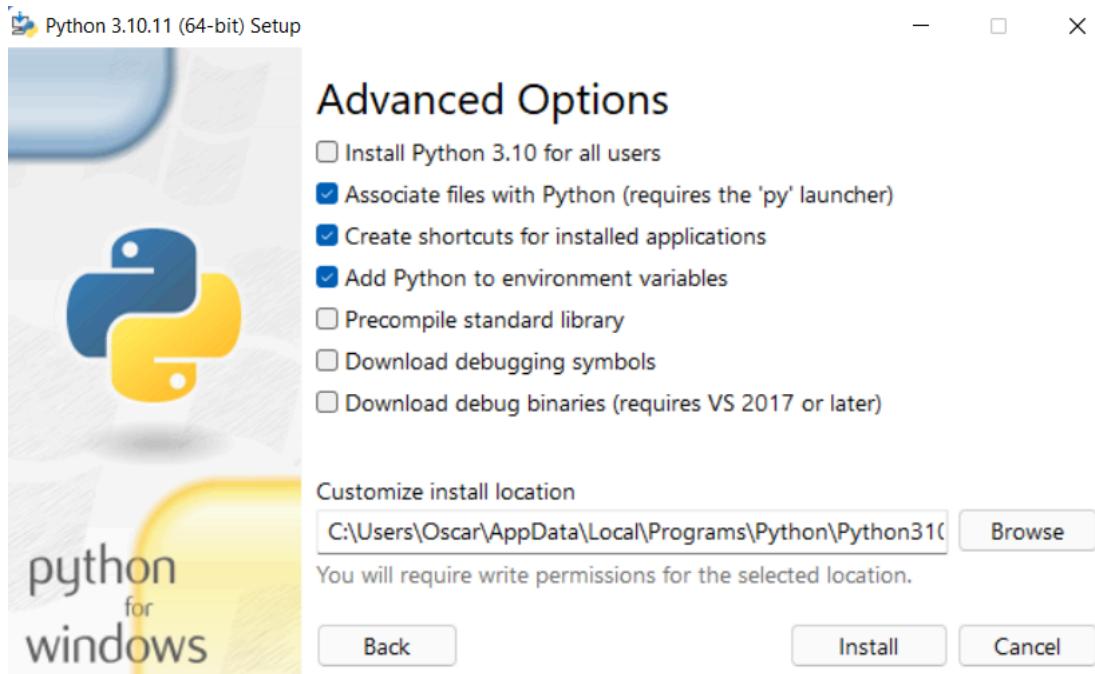
- Open het bestand en klik op “Customize Installation”:



- Zorg dat alles onder Optional Features is aangevinkt:



- Zorg dat Advanced Options als volgt is ingesteld:



- klik op Install

- **Microsoft Visual C++ 14.0 of hoger**
(<https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170#visual-studio-2015-2017-2019-and-2022>) installeer de nieuwste versie.
Volg de instructies.

Programma gebruik

- Pak de zip-folder uit, en open de uitgepakte folder
- Open de folder “srm-main” en draai het bestand “run.py”, laat het even zijn werk doen en volg de instructies. Vervolgens opent de applicatie, *maximaliseer de applicatie voor het beste resultaat*.
- Om te wisselen tussen punt-tot-punt route en circulaire route klik je op de rode knop “Circulaire route”/”Normale route”, er zijn dan een aantal opties te zien.
- Voer de coördinaten van je startpunt in dit kan door ze met de hand in te voeren of door op de kaart te klikken met de rechtermuisknop en het punt te selecteren.
- Voer vervolgens de gewenste afstand in. De rest van de opties zijn optioneel.
 - Totale klim: voer in meters in hoeveel je wilt klimmen tijdens het afleggen van de route
 - Gewenst % verhard: voer in gehele getallen tussen 0 en 100 wat het percentage is dat je op verhard wegdek wilt fietsen.
 - Helling: vul hier het percentage in dat je als maximale hellingspercentage wilt zien in de route, alle routes die een hoger hellingspercentage bevatten worden geschrapt. Indien hierdoor geen routes overblijven zal er geen resultaat volgen.
- Druk op enter of op de blauwe “Bereken route” knop. Geef het programma even de tijd om de routes te berekenen, de tijd die hiervoor nodig is, is afhankelijk van het systeem. Bij een langzamer systeem en/of lange routes kan het weleens 30+ minuten kosten.
- *Voortgang en technische gegevens zijn te zien in de console die in de achtergrond opent.*
- Druk op de “Export GPX” om een GPS Exchange bestand te downloaden die je kan uploaden naar je favoriete fietsroute app om zo de route te kunnen volgen.

Functies

plan_circular_route_flower()

Genereert een bloemachtige route structuur op een gegeven grafiek, waarbij elk blad een cirkelvormig pad vertegenwoordigt dat het startpunt passeert. Het algoritme plaatst punten op elk blad, waardoor een route ontstaat die ze verbindt. Het aantal bladeren, punten per blad en andere parameters kunnen worden aangepast. Het algoritme evalueert de gegenereerde routes en scores van deze routes op basis van afstand, hoogteverschil en percentage verhard wegdek. De route met de laagste score wordt gekozen aangezien dit betekent dat deze route het minste afwijkt van de input.

Parameters

- **start_coordinates** : tuple De coördinaten (breedtegraad, lengtegraad) van het startpunt.
- **max_length** : int De maximale gewenste lengte van de gegenereerde route.
- **elevation_diff_input** : int Het maximale gewenste hoogteverschil van de gegenereerde route.
- **percentage_hard_input** : int Het maximale gewenste percentage van verharde oppervlakken van de gegenereerde route.
- **requested_steeplness** : int Het maximale ingevoerde hellingspercentage. Dit is een hard limiet en het algoritme zal dus geen pad teruggeven met een helling die boven dit getal uitkomt.
- **options** : dict Extra opties voor analyse en visualisatie.

Returns

dict Uitvoer met alle relevante informatie voor verdere analyse en visualisatie.

Notities

- Genereert een cirkelvormig patroon met een configurerbaar aantal bladeren rond het startknooppunt.
- Bereken de route door punten op elk blad te verbinden.
- Evalueert meerdere paden en selecteert degene die het dichtst bij de opgegeven gebruikersinvoer ligt.
- Voert padanalyse uit, oppervlakteverdelingsanalyse en visualiseert optioneel de route.
- Deze functie is ontworpen voor routeplanning op een grafiek, rekening houdend met geografische coördinaten en verschillende padattributen.

Voorbeeld

```
start_coordinates = (breedtegraad, lengtegraad)
max_length = de invoer van de gebruiker in meters
elevation_diff_input = de invoer van de gebruiker in meters
percentage_hard_input = de invoer van de gebruiker in percentage
options = {"analyze": True, "surface_dist": True}
```

planner.calculate_start_point_index()

Deze functie berekent de index van het startpunt op een cirkelvormige structuur op basis van de gegeven bloemhoek en het aantal punten in elk blad. De hoek wordt omgezet in graden, en de index wordt bepaald door de verhouding van de bloemhoek ten opzichte van de totale cirkel. Als de berekende index het totale aantal overschrijdt punten per blad, dan wordt het omgeslagen om ervoor te zorgen dat er een geldige index wordt berekend.

Parameters

- flower_angle: Hoek (in radialen) die de gewenste richting van de cirkelvormige structuur vertegenwoordigt.
- points_per_leaf: Aantal punten of knooppunten in elk blad van de cirkelvormige structuur.
- Geeft terug
- float: Index van het startpunt op de cirkelvormige structuur.

Returns

start_point_index: int, de index waar het startpunt moet worden ingevoegd in de lijst van berekende punten

planner.calculate_leaf_nodes()

Deze methode genereert knooppunten voor elk blad in een bloemachtig patroon. Elke iteratie van deze functie geeft 1 “niet complete” route. In de hoofdfunctie van het programma wordt deze functie meerdere keren aangeroepen om meerdere routes te genereren.

Parameters

- flower_angle (float): De hoek van het bloemachtige patroon in radialen.
- points_per_leaf (int): Het aantal punten (knooppunten) dat moet worden gegenereerd voor elk blad.
- radius (float): De straal van elk blad in het bloemachtige patroon.
- variance (float): De variatie in de straal van elk blad.
- start_node (int): Het knooppunt-ID van het startknooppunt.
- graph (networkx.Graph): De grafiek die het gebied vertegenwoordigt.

Returns

list: Een lijst van knooppunt-ID's die de knooppunten voor elk blad in het bloemachtige patroon vertegenwoordigen.

graph.insert_start_node_and_rearrange()

Voegt een startknooppunt in op een gespecificeerde index in een lijst van bladknooppunten en herordend de lijst om een cirkelvormige structuur te creëren. Waarbij het eerste punt ook echt het startpunt is.

Parameters

- `self`: Instantie van de klasse.
- `leaf_nodes(list)`: Lijst van bladknooppunten die een pad of sequentie vertegenwoordigen.
- `start_node`: Knooppunt dat op de gespecificeerde index moet worden ingevoegd.
- `start_point_index`: Index waar het startknooppunt moet worden ingevoegd.

Returns

lijst: Herordende lijst van bladknooppunten met het startknooppunt op de gespecificeerde index, vormt een cirkelvormige structuur.

visualizer.visualize_leaf_points()

Visualiseert de bladpunten en slaat de plot op als een afbeelding.

Parameters

- `leaf_paths (lijst)`: Lijst van paden die bladknooppunten vertegenwoordigen.
- `graph (MultiDiGraph)`: De grafiek met knooppuntcoördinaten.

Returns

None

visualizer.visualize_best_path()

Visualiseert het beste pad en slaat de plot op als een afbeelding.

Parameters

- path (lijst): Lijst van paden die bladknooppunten vertegenwoordigen.
- graph (MultiDiGraph): De grafiek met knooppuntcoördinaten.

Returns

None

visualizer.visualize_elevations()

Visualiseer de hoogtes van een pad en sla de grafiek op als een afbeelding.

Parameters:

- graph (MutliDiGraph): De grafiek die het gebied vertegenwoordigt.
- path (list): Een lijst met knooppunt-ID's die het pad voorstellen.

Deze methode haalt de hoogtegegevens op voor elk knooppunt in het pad met behulp van de SRTM-bibliotheek. Vervolgens maakt het een grafiek van de hoogtegegevens met behulp van matplotlib, waarbij de knooppuntindex op de x-as staat en de hoogte op de y-as staat.

Opmerking: De methode gebruikt de attributen 'y' en 'x' van de knooppunten in de grafiek om de breedtegraad en lengtegraad voor te stellen.

visualizer.visualize_surface_percentage()

Visualiseer het percentage verhard/onverhard wegdek in een taartdiagram.

Parameters

- Percentage(float): percentage verhard wegdek
-

analyzer.get_paths_and_path_lengths()

Haalt de paden en padlengtes uit een lijst van paden op leafs(bladen). Deze methode berekent eigenlijk tussen elk punt een route en plak deze aan elkaar.

Parameters

- graph (MultiDiGraph): Instantie van een osmnx grafiek.
- leaf_paths (lijst): Lijst van bladpaden.

Returns

- paths (lijst): Lijst van paden.
- path_lengths (lijst): Lijst van padlengtes.

De indices van de twee lijsten komen overeen met elkaar.

analyzer.min_length_routes_indeces()

Identificeert en geeft de indices van de routes met lengtes die het dichtst bij een specifieke maximale lengte liggen terug. Hiermee worden het aantal routes die gescoord moeten worden verminderd

Parameters

- paths(list): Lijst van routes voorgesteld als knooppunten of randen.
- path_lengths(list): Lijst van overeenkomstige lengtes voor elke route.
- max_length(int): Ingevoerde lengte van de route.
- leafs(int): Aantal totaal gegenereerde routes.

Returns

lijst: Indices van de routes met lengtes die het dichtst bij de gespecificeerde maximale lengte liggen, dit is contextueel naar de paden lijst (paths_list).

analyzer.get_height_diffs()

Berekent het procentuele verschil tussen de verhogingen op een pad op ieder punt van het pad.

Parameters

- graph (networkx.Graph): De grafiek die het gebied vertegenwoordigt.
- paths (list): Een lijst met paden, waarbij elk pad een lijst is van knooppunt-ID's.
- path_lengths (list): Een lijst met de lengtes van elk pad.
- min_length_diff_routes_indeces (list): Een lijst met indices van de paden die minimale lengteverschillen hebben.
- elevation_diff_input (float): Het gewenste hoogteverschil.

Returns

- dict: Een woordenboek waarbij de sleutels de indices van de paden zijn en de waarden de procentuele verschillen zijn tussen het hoogteverschil van het pad en het opgegeven hoogteverschil.
-

analyzer.get_surface_diffs()

Bereken het absolute verschil tussen de oppervlakteverdeling van elk pad en het ingevoerde percentage harde oppervlakken.

Parameters

- self: Instantie van de klasse.
- graph(MultiDiGraph): Grafiek met knooppunten en randen.
- paths(list): Lijst van routes weergegeven als knooppunten of randen.
- path_lengths(list): Lijst van overeenkomstige lengtes voor elke route.
- min_length_diff_routes_indeces(list): Lijst van indices van de routes die minimale lengteverschil hebben.
- percentage_hard_input(int): Ingevoerd percentage van harde oppervlakken.

Returns

dict: dictionary waarin het procentuele verschil tussen input en output van oppervlakteverdeling tussen verhard/onverhard staat.

analyzer.calculate_percentage_hardened_surfaces()

Bereken het percentage harde oppervlakken in een pad.

Parameters

- self: Instantie van de klasse.
- graph(MultiDiGraph): Grafiek met knooppunten en randen.
- path(MutliDiGraph): Pad binnen de grafiek voor percentageberekening.

Returns

float: Percentage van harde oppervlakken op het gespecificeerde pad.

analyzer.calculate_elevation_diff()

De methode berekent het hoogteverschil voor elk pad in min_length_diff_routes_indeces en berekent vervolgens het absolute verschil tussen dit hoogteverschil en het opgegeven hoogteverschil. Deze verschillen worden opgeslagen in een woordenboek en vervolgens geretourneerd.

Berekent het totale positieve hoogteverschil langs een gespecificeerd pad binnen een grafiek.

Parameters

- graph: Grafiek met knooppunten en randen.
- path: Pad binnen de grafiek voor de berekening van het hoogteverschil.

Returns

float: Totaal positief hoogteverschil langs het gespecificeerde pad.

Deze functie maakt gebruik van hoogtegegevens, vanuit de SRTM dataset, om hoge waarden op te halen voor knooppunten in het gegeven pad. Het berekent vervolgens de positieve hoogteverschillen tussen aangrenzende knooppunten en geeft de totale som terug.

analyzer.get_score_only_elevation()

Berekent score voor een routes met alleen de lengte en de hoogteverschillen meegerekend

Parameters

- self: Instantie van de klasse.
- graph: Grafiek met knooppunten en randen.
- paths: Lijst met routes.
- path_lengths: Lijst met overeenkomstige lengtes voor elke route.
- min_length_diff_routes_indeces: Lijst met indices van de routes die een minimale lengteverschil hebben met de invoer.
- elevation_diff_input: Ingevoerde hoogteverschil.
- max_length: Ingevoerde lengte van de route.

Returns

- dict: Dictionary waarbij de keys de indices zijn van de paden in de "paths" lijst en de waarden zijn de scores van de overeenkomstige paden.

analyzer.get_score_only_surface()

Berekent score voor routes met alleen de percentage verhard wegdek meegerekend

Parameters

- self: Instantie van de klasse.
- graph: Grafiek met knooppunten en randen.
- paths: Lijst met routes.
- path_lengths: Lijst met overeenkomstige lengtes voor elke route.
- min_length_diff_routes_indeces: Lijst met indices van de routes die een minimale lengteverschil hebben met de invoer.
- percentage_hard_input: De ingevoerde voorkeur van percentage verhard wegdek.
- max_length: Ingevoerde lengte van de route.

Returns

- dict: Dictionary waarbij de keys de indices zijn van de paden in de "paths" lijst en de waarden zijn de scores van de overeenkomstige paden.

analyzer.get_score_elevation_and_surface()

Berekent score voor routes met zowel hoogteverschillen als percentage verhard wegdek meegerekend

Parameters

- self: Instantie van de klasse.
- graph: Grafiek met knooppunten en randen.
- paths: Lijst met routes.
- path_lengths: Lijst met overeenkomstige lengtes voor elke route.
- min_length_diff_routes_indeces: Lijst met indices van de routes die een minimale lengteverschil hebben met de invoer.
- percentage_hard_input: De ingevoerde voorkeur van percentage verhard wegdek.
- elevation_diff_input: Ingevoerde hoogteverschil.
- max_length: Ingevoerde lengte van de route.

Returns

- dict: Dictionary waarbij de keys de indices zijn van de paden in de "paths" lijst en de waarden zijn de scores van de overeenkomstige paden.

analyzer.remove_paths_above_stEEPNESS()

Verwijderd alle paden met een te hoge steilheid uit de dictionary paths_with_scores.

Parameters

- self: Instantie van de klasse.
- graph: Grafiek met nodes en randen.
- paths: Lijst van routes.
- paths_with_scores: Lijst met alle paden en hun scores
- min_length_diff_routes_indeces: Lijst van indices van de routes die minimale lengteverschil hebben met de invoer.
- requested_max_stEEPNESS: Ingevoerde maximale steilheid.

Returns

- dict: paths_with_scores met de paden met een te hoge steilheid verwijderd.

graph.export_GPX()

Exporteert een lijst met knooppunt-ID's naar een GPX-bestand.

Parameters

- node_ids: Lijst met te exporteren knooppunt-ID's.

Returns

- Response: GPX-bestand. Deze functie exporteert een lijst met knooppunt-ID's naar een GPX-bestand. Het haalt eerst de knooppuntgegevens op van de OpenStreetMap API, en maakt vervolgens een GPX-bestand van de gegevens. "
-

Voor volledige werking en volledige context, zie de source-code van het project via de github omgeving(<https://github.com/Ozziehman/srm>)

Scoring uitgelegd

Paden worden opgeslagen in een dictionary path_with_scores. Per pad dat hierin staat is de score als volgt:

verschil klim in/output +

verschil tussen in/output percentage verhard oppervlak +

procentuele afwijking tussen de in/output van de route lengte

in code:

```
paths_with_scores[path_index] =  
height_diffs[path_index] +  
surface_diffs[path_index] +  
abs(path_lengths[path_index] - max_length) / max_length
```

Score van afstand:

De score van de afstand van de route wordt berekend door de procentuele afwijking van de output ten opzichte van de input te berekenen.

Score van hoogte:

De score van de hoogteverschillen wordt berekend door de procentuele afwijking van de output ten opzichte van de input te berekenen.

score van percentage verhard wegdek:

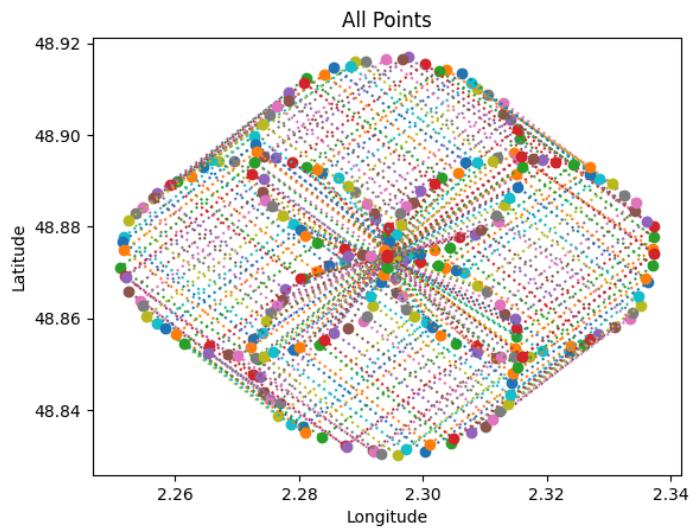
De score van het percentage verhard wegdek wordt berekend door het absolute verschil tussen het ingevoerde percentage en de output percentage te berekenen.

Helling wordt niet meegenomen in de score, alle routes die een te hoge helling bevatten worden geschrapt.

Visualisaties

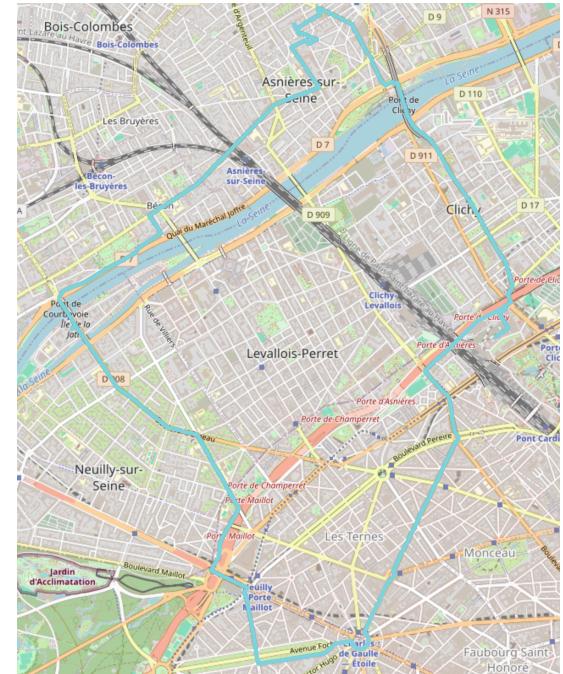
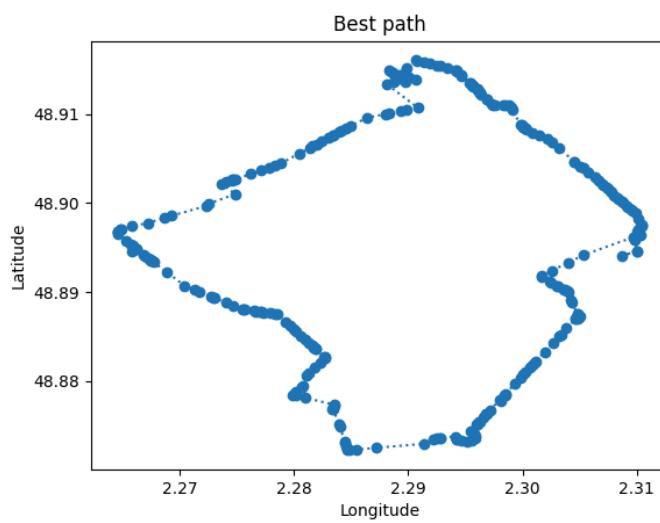
Gegenereerde punten

Dit is een visualisatie van de punten gegenereerd door het algoritme, met behulp van deze abstracte routes worden later routes gemaakt. Alle stippellijnen verbinden met punten binnen dezelfde route, deze zijn op kleur gesorteerd



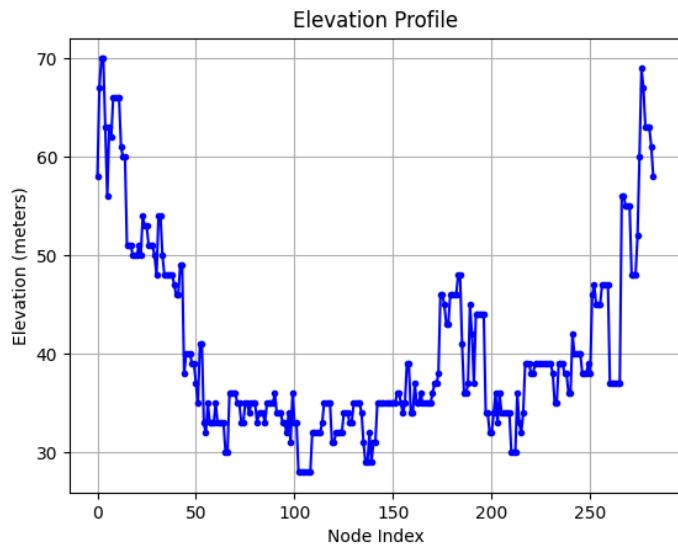
Beste pad

Dit is een visualisatie van het beste pad dat gevonden wordt op basis van scores.



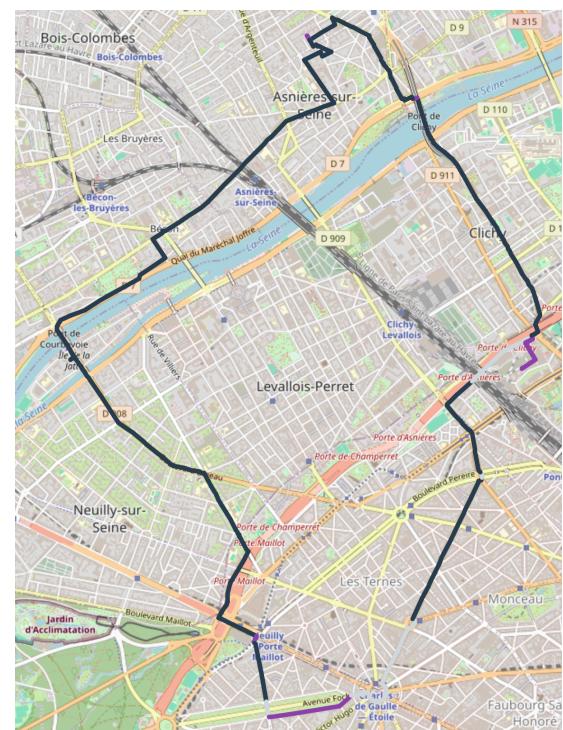
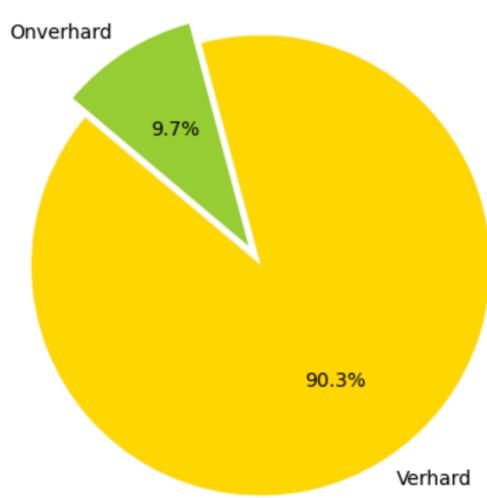
Hoogteverschillen

Dit is een visualisatie van de hoogteverschillen binnen de route. De pieken worden veroorzaakt doordat de afstand tussen de nodes niet altijd hetzelfde is.



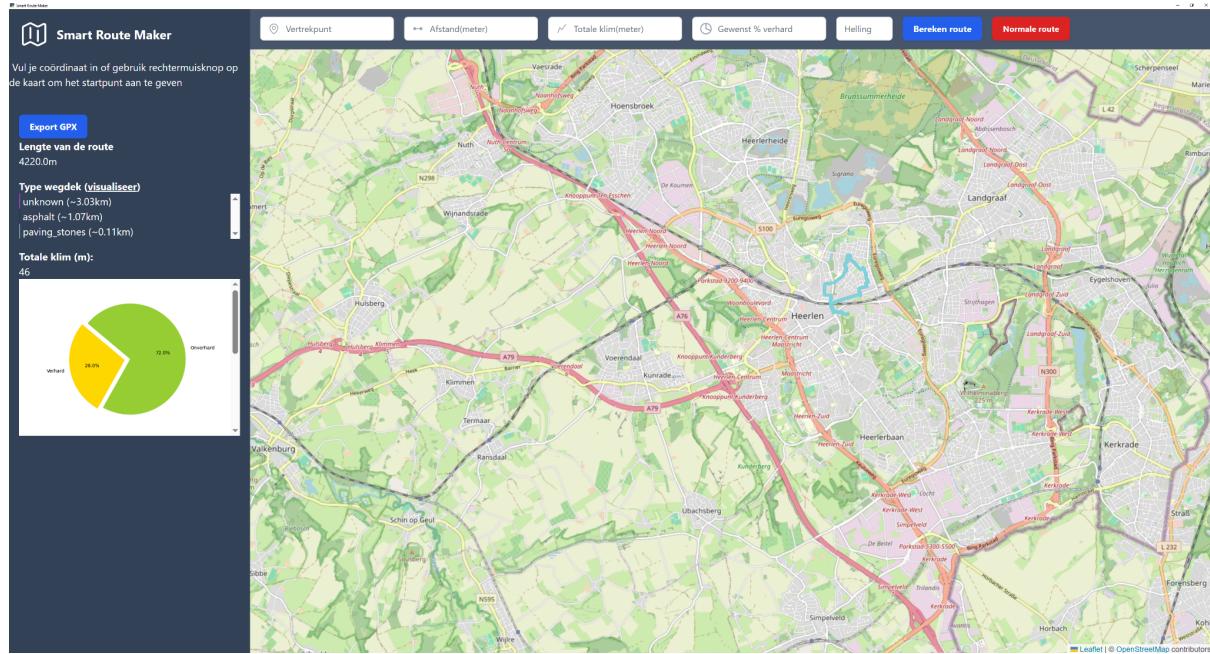
Verhard/onverhard

Dit is een visualisatie van verhard/onverhard wegdek. Diagram en route



Gehele UI

Visualisatie van de applicatie in werking.



Hoogteverschillen

Met betrekking tot het inlezen van hoogteverschillen is een externe tool gebruikt(niet uit OSMnx). Na onderzoek naar mogelijkheden om deze hoogtes in te lezen zijn er een aantal opties naar voren gekomen, API's en datasets. API's maken gebruik van de datasets maar zijn makkelijk te gebruiken. Er waren 2 API's die kandidaat werden: Google elevation API(*Get started*, z.d.) en de open-meteo API. De google API was betaald en niet open source en de Open-Meteo API had problemen met vertraging en overload.

Uiteindelijk is er een manier gevonden om de dataset van de SRTM(Nasa, 2023) in te lezen, d.m.v. de SRTM.py library. Deze optie was veruit het beste met de nagenoeg verwaarloosbare impact die het had op de runtime van het algoritme. Het procentuele verschil tussen de input en de output wordt toegevoegd aan de score van de route.

De route met de laagste score wordt gekozen aangezien de score het verschil tussen de input en de output representeren.

Percentage verhard/onverhard

De gebruiker kan een voorkeur uiten naar het algoritme over het percentage verhard wegdek. Echter is het niet relevant om hier een analyse over precisie uit te voeren aangezien het percentage verhard/onverhard zeer sterk afhangt(nog erger dan hoogtes) van de omgeving waar het pad in gemaakt wordt. Op het gebied van hoogtes kan er nog verschil in zitten in resultaten, maar als het wordt uitgevoerd in bijvoorbeeld Parijs is de kans op onverhard wegdek zo laag dat het vrijwel altijd 100% verhard of deels "unknown" is. "Unknown" wegdek wordt gezien als onverhard. Het algoritme voegt het verschil tussen de input en de output toe aan de score van iedere route.

Om deze percentages te berekenen worden de `analyzer.get_path_attributes()` en `analyzer.get_path_surface_distribution()` functies gebruikt, deze worden ook gebruikt om de route te visualiseren in het eerder aangeleverde product

De route met de laagste score wordt gekozen aangezien de score het verschil tussen de input en de output representeren.

Hellingspercentage

Per pad wordt, nadat de score is toegepast, de maximale helling in dat pad berekend. Als dit boven de ingevulde waarde uitkomt wordt het pas geschrapt. De helling wordt berekend door het hoogteverschil tussen 2 punten te delen door de afstand. Deze parameter wordt NIET meegerekend in de score van een route aangezien er een hard limiet zit aan de maximale helling in een route, dit betekent dat een route met een hogere helling dan aangegeven volledig verwijderd moet worden. De informatie die hiervoor wordt gebruikt is niet altijd even accuraat en zal dus niet altijd een optimale route geven die ook overeenkomt met de werkelijkheid.

Analyse

Om het algoritme te analyseren is er een theoretische analyse uitgevoerd, maar is er ook getest hoe het algoritme zich in de praktijk gedraagt aangezien er gekozen is om te werken met multiprocessing waardoor het te afhankelijk is van de specificaties van het systeem waar het op wordt uitgevoerd om zeker te zeggen wat de tijdscomplexiteit is. De reden waarom er voor multiprocessing is gekozen volgt:

In deze tabel is duidelijk te zien dat multiprocessing de runtime van het algoritme aanzienlijk verlaagt. Bij zeer korte afstanden duurt het wat langer dan normaal aangezien de subprocessen opzetten en beheren meer tijd kost, maar zoals er te zien is wordt multiprocessing al gauw efficiënter bij langere afstanden.

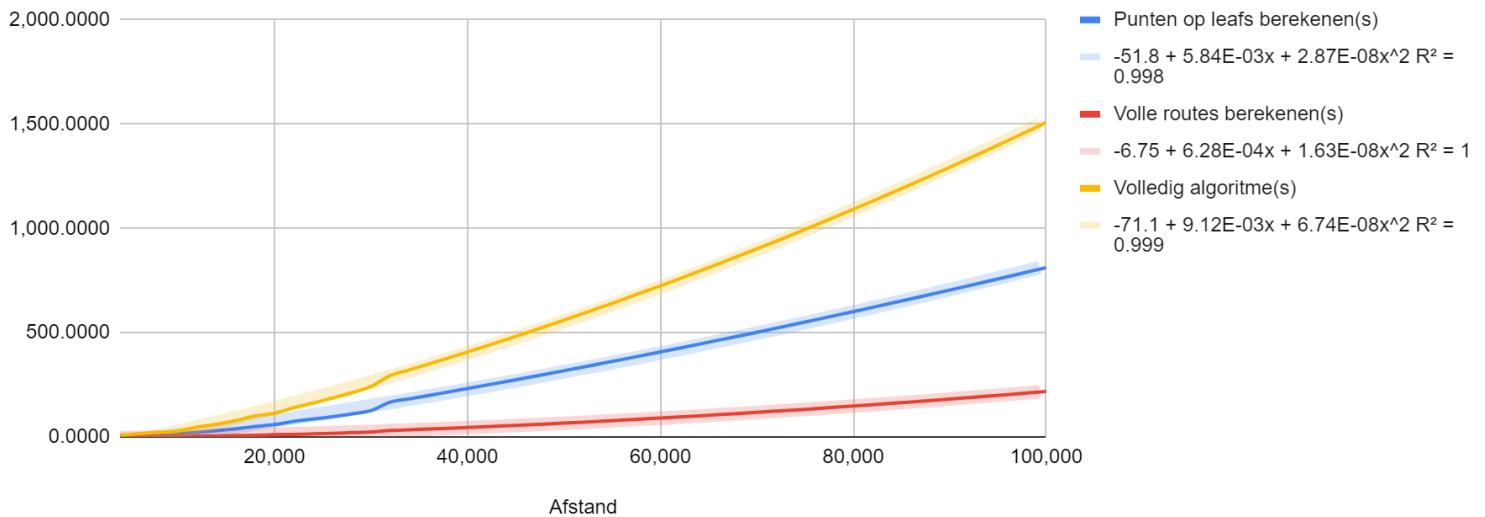
| | Standaard tijd (s) | Multiprocessing (12 workers) tijd (s) |
|------|--------------------|---------------------------------------|
| 2km | 4.283998012542725 | 4.458516836166382 |
| 5km | 10.688997983932495 | 6.682975769042969 |
| 10km | 32.48199987411499 | 16.589996576309204 |
| 15km | 71.85900163650513 | 33.49699902534485 |
| 20km | 122.62920951843262 | 57.96599984169006 |

Voorbij 10km is er te zien dat de tijd meer dan halveert! Het is belangrijk om te onthouden dat deze tijden niet definitief zijn en zullen afhangen van de specificaties van het systeem.

Tijdscomplexiteit

Door de aanwezigheid van nested loops is de worst-case tijdscomplexiteit van het algoritme: $O(n^2)$, maar aangezien er relatief een stuk meer gebruik wordt gemaakt van $O(n)$ en $O(n * \log(n))$ zal het in de praktijk een zeer zwakke $O(n^2)$ zijn. Om deze stelling na te gaan zijn er genoeg metingen gedaan op een systeem met een Ryzen 5 3600 om een polynomiale lijn op te stellen die deze stelling ondersteunt:

Punten op leafs berekenen(s), Volle routes berekenen(s) en Volledig algoritme(s)



Deze tijden zullen vrijwel nooit overeenkomen met een ander systeem. Het algoritme is namelijk zeer CPU-intensief en resultaten zullen dus variëren van processor tot processor van locatie tot locatie en de omstandigheid van de rest van het systeem.

Het zal misschien opvallen dat er een knik zit in de grafiek, maar deze was nodig om de gehele berekening overeen te laten komen met de meeste waarden. De reden hiervoor is dat multiprocessing wordt geregeld door het systeem en niet door het algoritme.

Het is van belang om op te merken dat alle route gerelateerde tests die zijn uitgevoerd uitgevoerd zijn vanaf de Arc de Triomphe in Parijs(48.873756, 2.294946) aangezien hier een zeer complex wegennetwerk omheen zit en dus de duidelijkste resultaten weergeeft.

Hoewel de theoretische worst-case tijdscomplexiteit $O(n^2)$ is, komt de realiteit dichter bij een $O(n * \log(n))$ doordat de $O(n^2)$ maar een klein gedeelte van het algoritme vormt.

Uitgebreide tests op het gebied van efficiëntie zijn te vinden in het testrapport (O.Theelen, 2024, SRM Testrapport)

Nauwkeurigheid

In deze technische documentatie is enkel een omschrijving van de nauwkeurigheid vastgelegd om als voorbeeld te fungeren. Gehele tests op nauwkeurigheid zijn te vinden in het testrapport (O.Theelen, SRM Testrapport, 2024)

Om de nauwkeurigheid van het algoritme te meten zijn er een aantal inputs voorbereid die realistisch te bereiken zijn met het gebruikte landschap.

| 64 mogelijke routes | 6 punten per route | |
|------------------------|--------------------|----------|
| input/uitkomst | afstand(m) | klim(m) |
| 55m klim/2km afstand | 2,320.0000 | 55.0000 |
| 85m klim/4km afstand | 4,030.0000 | 88.0000 |
| 115m klim/6km afstand | 6,160.0000 | 117.0000 |
| 145m klim/8km afstand | 8,100.0000 | 145.0000 |
| 175m klim/10km afstand | 9,890.0000 | 171.0000 |
| 205m klim/12km afstand | 11,780.0000 | 201.0000 |
| 235m klim/14km afstand | 14,160.0000 | 238.0000 |
| 265m klim/16km afstand | 16,160.0000 | 240.0000 |
| 295m klim/18km afstand | 17,680.0000 | 292.0000 |
| 325m klim/20km afstand | 19,740.0000 | 322.0000 |

Er is te zien dat het algoritme met redelijk accurate resultaten komt. De afwijking is niet te verhelpen aangezien straten nou eenmaal niet overeen zullen komen met de gewenste resultaten en dus bijna altijd zullen afwijken.

Nauwkeurigheid met alle mogelijke parameters(lengte, hoogteverschil, verhard percentage), deze tests zijn wel uitgevoerd in Nederland aangezien er bij de Arc de Triomph vrijwel alleen maar verharde wegen zijn. De tests zijn uitgevoerd vanaf Zuyd Hogeschool Heerlen (50.88428, 5.986104):

| 64 mogelijke routes | 5 punten per route | | |
|---|--------------------|---------|--------------------|
| input/uitkomst | afstand(m) | klim(m) | Verhard percentage |
| 10km afstand/ 100m klim/ 50% verhard | 9910 | 105 | 0.5640766902119072 |
| 10km afstand/ 100m klim/ 60% verhard | 9860 | 106 | 0.5851926977687627 |
| 10km afstand/ 100m klim/ 70% | 9620 | 96 | 0.7536382536382535 |
| 10km afstand/ 100m klim/ 80% verhard | 9620 | 96 | 0.7536382536382535 |
| 10km afstand/ 100m klim/ 90% verhard | 9620 | 96 | 0.7536382536382535 |
| 10km afstand/ 100m klim/ 100% verhard | 9620 | 96 | 0.7536382536382535 |

Tot op een bepaald punt heeft het nut om het percentage verhard/onverhard mee te rekenen. Echter raak je op een gegeven moment een limiet met de mogelijkheden in een gebied. Dit is op te lossen door de hoeveelheid punten, de lengte van de route of de hoeveelheid geëvalueerde routes te vergroten, maar dit zou de tijd die nodig is om het algoritme uit te voeren zo erg verhogen dat dit het over het algemeen niet waard zal zijn.

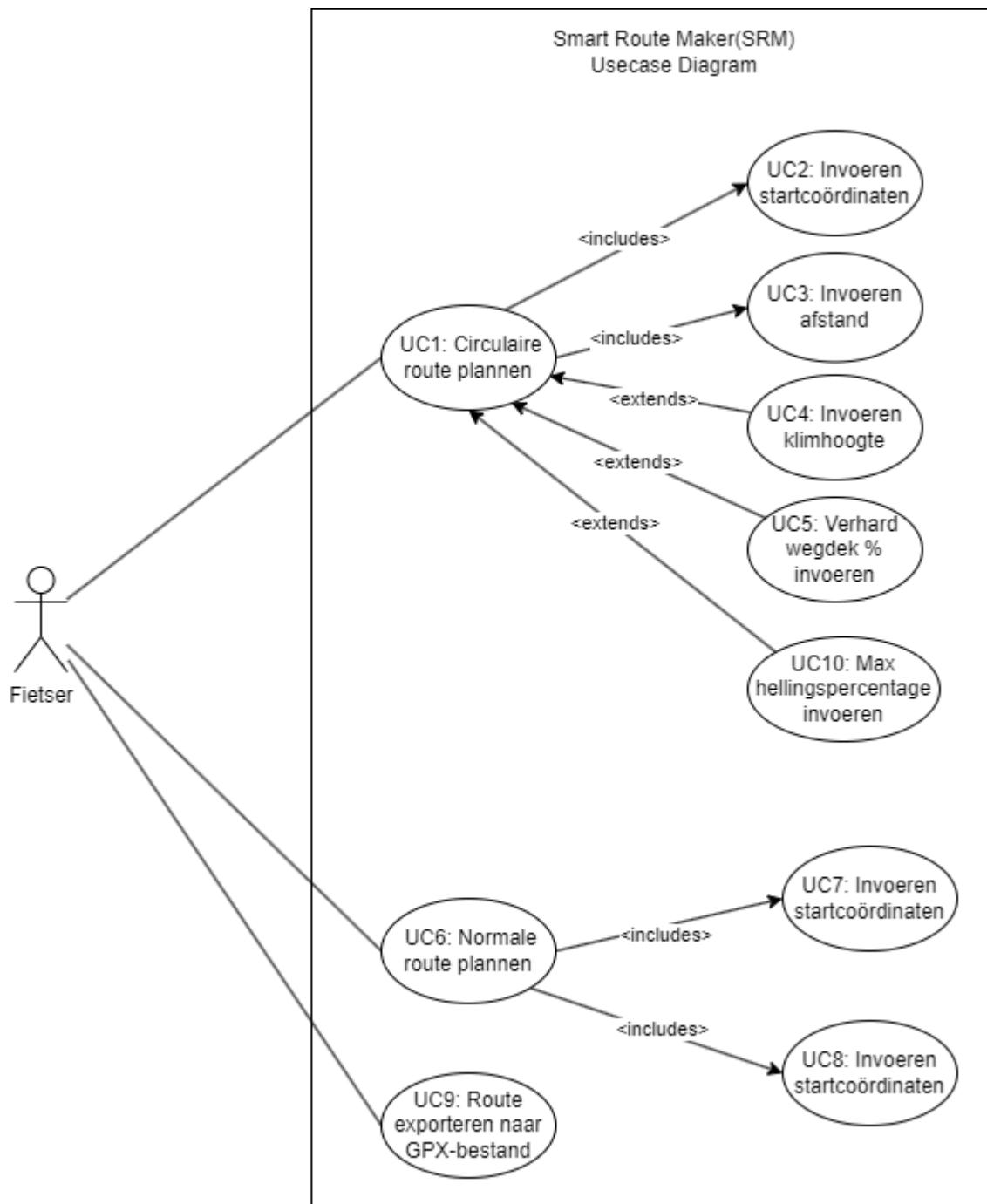
Voor een uitgebreid testrapport met diepgaande tests raadpleeg het SRM testrapport (O.Theelen, SRM Testrapport, 2024)

Export naar GPX

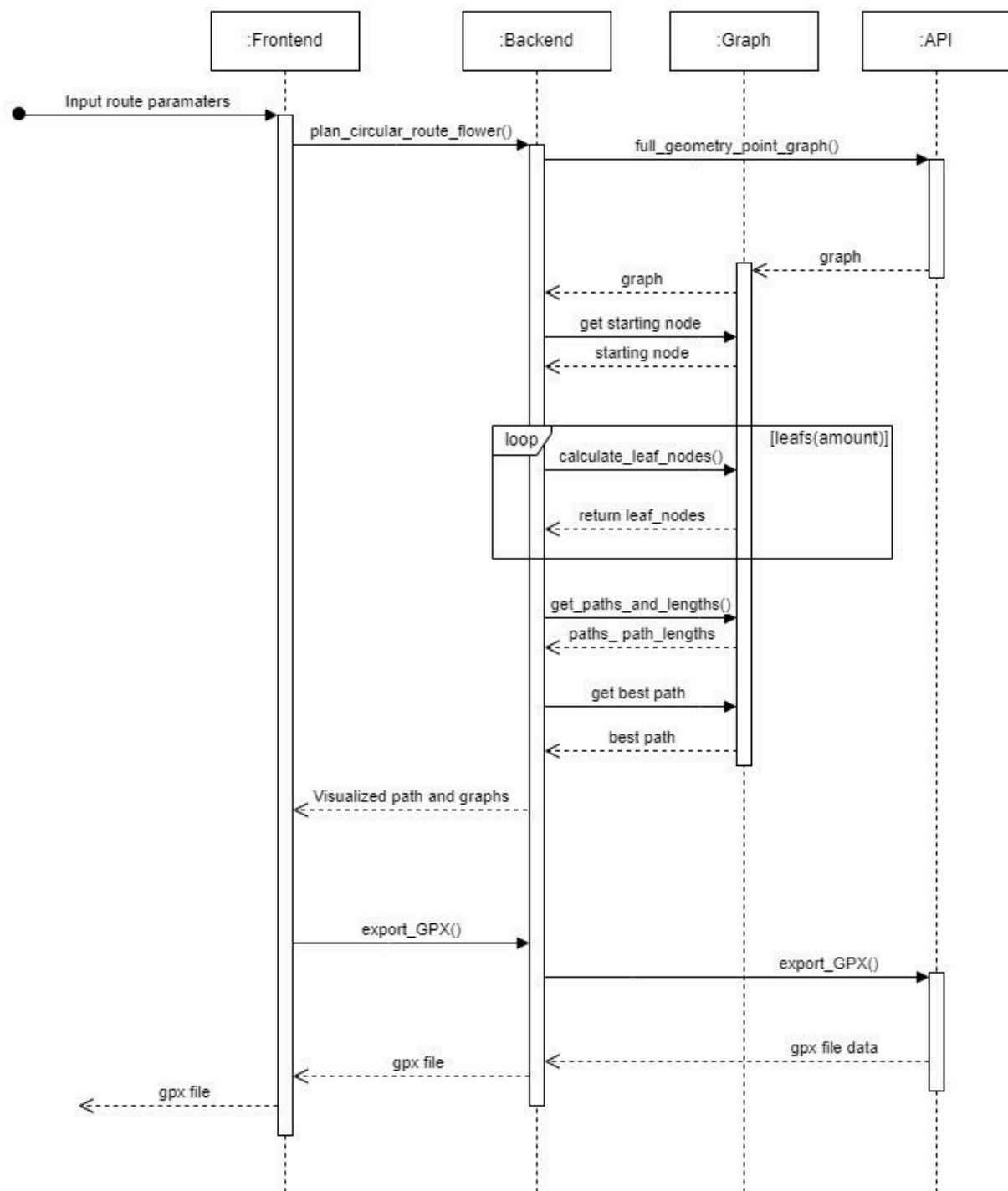
Om een route te exporteren naar een GPX bestand wordt er informatie vanuit de Overpass(*Overpass API - OpenStreetMap Wiki*, z.d.) API gehaald. Hiervoor is gekozen aangezien het niet efficiënt is om de gehele graaf opnieuw in te laden om een route te exporteren, dit kost namelijk een hoop tijd. De API returned benodigde data en wordt in python omgezet naar een geldig GPX formaat. GPX-bestanden zijn te openen in de meeste route-apps, maar het meest toegankelijke hiervoor is Komoot. Met Komoot kan men gratis GPX bestanden omzetten naar een navigeerbare route

Diagrammen

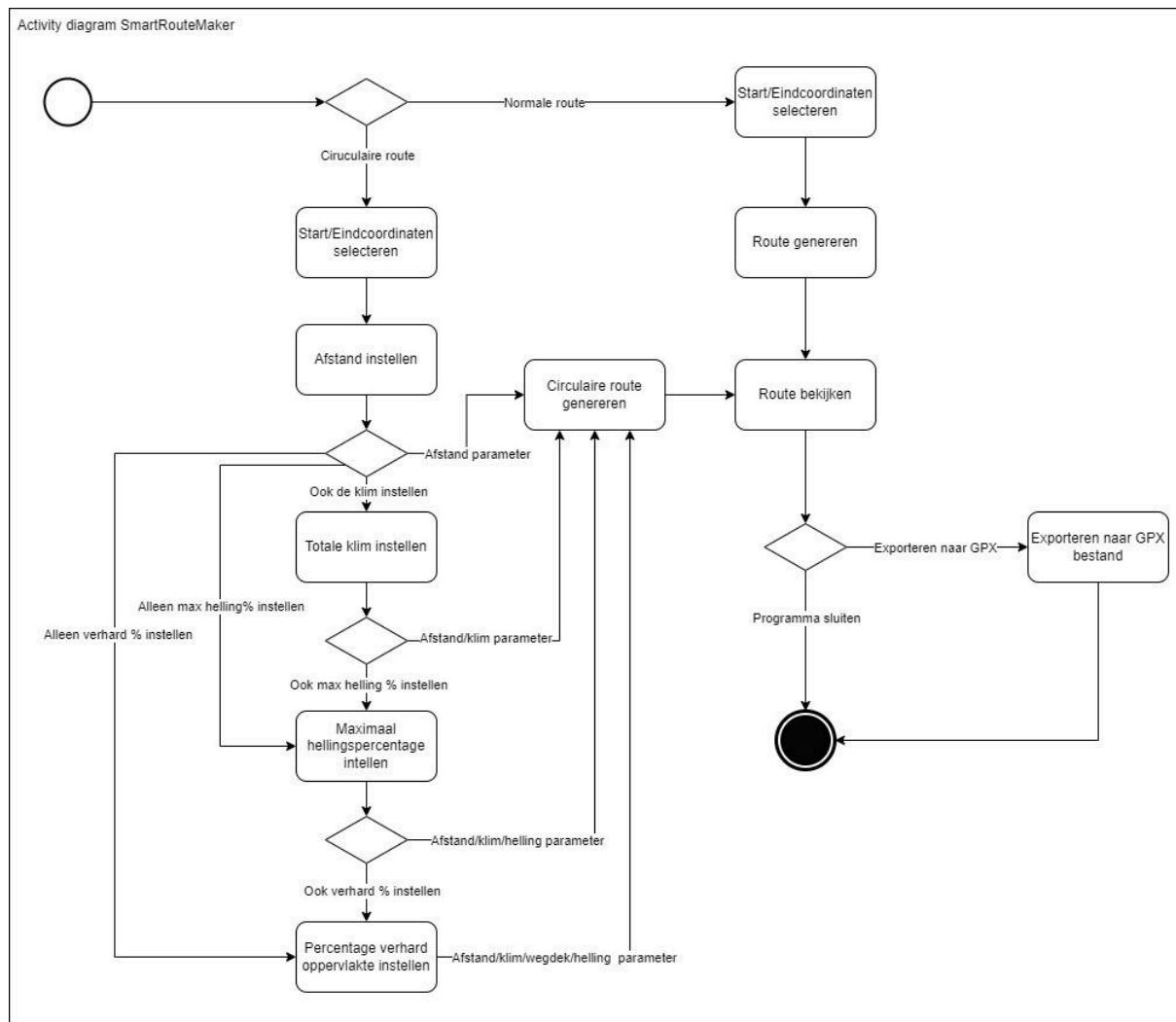
Usecase diagram



Sequence diagram



Activity diagram



Bronnen

S.Vanwersch, S.Tabárá, J.Drenth, M.Meijers (2022)
BD01_TechnischeDocumentatie_SRM_2022

O. Theelen (2024) SRM Analyserapport

O. Theelen (2024) SRM TestRapport

Get started. (z.d.). Google for Developers.

<https://developers.google.com/maps/documentation/elevation/start>

Open-Meteo (z.d.) Elevation API

<https://open-meteo.com/en/docs/elevation-api>

Nasa, E. S. D. S. (2023, 27 juli). *SRTM* | *EarthData*. Earthdata.

<https://www.earthdata.nasa.gov/sensors/srtm>

Overpass API - OpenStreetMap Wiki. (z.d.).

https://wiki.openstreetmap.org/wiki/Overpass_API

Ozziehman (z.d.)

<https://github.com/Ozziehman/srm>