Stikstofpuzzel

Release 1.0

Witteveen+Bos

23 augustus 2022

CONTENTS:

CHAPTER

ONE

STIKSTOFPUZZEL

1.1 Aanleiding

Onder het voormalige Programma Aanpak Stikstof (PAS) bestond voor activiteiten met een stikstofdepositie onder een bepaalde grenswaarden een meldingsplicht. In totaal zijn onder dit programma meer dan 3500 meldingen gedaan. Een groot deel van deze meldingen betreft landbouwbedrijven, de overige meldingen komen uit de bouw-, industrie- en energiesector.

Door de uitspraak van de Raad van State op 29 mei 2019 (r.o. 33.2 van de uitspraak) zijn deze meldingen met terugwerkende kracht niet meer vrijgesteld van de vergunningplicht in het kader van de Wet natuurbescherming - gebiedsbescherming. Het is nodig om mogelijke (significante) gevolgen voor beschermde Natura 2000-gebieden van de betreffende activiteiten alsnog te beoordelen en zo nodig en mogelijk te vergunnen (legalisatie). Circa 2300 melders hebben aangegeven de gemaakte PAS-melding te willen legaliseren. Voor de legalisering van deze voormalige PAS-meldingen is door het ministerie van LNV stikstofruimte gereserveerd welke afkomstig is uit bronmaatregelen.

1.2 Probleemstelling

Het vergunnen van de circa 2300 PAS-meldingen wordt uitgevoerd door de beschikbare stikstofruimte uit de bronmaatregelen te matchen met de benodigde stikstofruimte van de PAS-meldingen. Daarnaast is het uitgangspunt dat er zo weinig mogelijk sprake is van restruimte (d.w.z. niet gebruikte maar wel beschikbare stikstofruimte) na het matchen. De systematiek en uitgangspunten voor het matchen van beschikbare stikstofruimte uit bronmaatregelen en benodigde stikstofruimte van PAS-meldingen en de systematiek voor het onderscheid tussen PAS-meldingen met en zonder prioriteit is vastgelegd in de notitie Visualisatie van prioritering geverifieerde meldingen.

Het bevoegd gezag geeft bij het reserveren van depositieruimte voor gemelde PAS-projecten voorrang aan projecten ten aanzien waarvan het vóór 13 januari 2022 een verzoek heeft ontvangen tot handhaving van de Natura 2000-vergunningplicht. Als blijkt dat niet alle PAS-meldingen gelegaliseerd kunnen worden in de beschikbare stikstofruimte, is het noodzakelijk om een keuze te maken tussen PAS-meldingen. Het bevoegd gezag kiest een combinatie aan PAS-meldingen op basis en op volgorde van de volgende criteria:

1. De combinatie aan PAS-meldingen waarbij de resterende depositieruimte minimaal is.
2. Als de totale resterende depositieruimte voor twee of meer combinaties van PAS-meldingen gelijk is, dan wordt de situatie gekozen waarbij het grootste aantal PAS-meldingen gelegaliseerd kan worden.
3. Als én de totale resterende depositieruimte én het aantal PAS-meldingen dat kan worden gelegaliseerd gelijk is voor meerdere combinaties van PAS-meldingen, wordt de situatie gekozen met de eerst ontvangen PAS-meldingen bij het RVO om gegevens aan te leveren.

Bovenstaand proces wordt tweemaal doorlopen. Eerst voor PAS-meldingen die een handhavingsverzoek hebben ontvangen vóór 13 januari 2022; daarna wordt de resterende ruimte gebruikt om een keuze te maken uit de andere PAS-meldingen.

1.3 Deterministische versus stochastische optimalisatietechnieken

Bovenstaand omschreven stikstofpuzzel is een typisch voorbeeld van een optimalisatieprobleem; men is geïnteresseerd in een optimale manier van verdelen van beschikbare stikstofruimte. Om dergelijke optimalisatieproblemen op te lossen kan gebruik gemaakt worden van verschillende technieken. Hierbij is er onder andere een onderscheid in *deterministische* technieken en *stochastische* technieken.

Wordt een deterministische techniek gebruikt om een oplossing te vinden voor het optimalisatieprobleem, dan zal de optimale oplossing altijd hetzelfde zijn; namelijk het meest optimale scenario voor de gehele oplossingsruimte. In het geval van de stikstofpuzzel betekent dit dat altijd de oplossing gevonden wordt waarbij de resterende depositieruimte minimaal is. Een deterministische techniek is daarmee transparant en geeft altijd een reproduceerbare oplossing.

Wordt een stochastische techniek gebruikt, dan is de uitkomst afhankelijk van bijvoorbeeld het startpunt vanaf waar gezocht wordt naar een optimum. Het kan zijn dat hierdoor een oplossing voor het optimalisatieprobleem gevonden wordt, die mogelijk niet het absolute optimum is. Het eerste noemen we dan een *lokaal optimum* (optimum gezien vanuit het lokale startpunt). Het echte optimum is het *globale optimum* (het optimum voor de gehele oplossingsruimte). Met het gebruik van een stochastische techniek kan wel gezocht worden naar een globaal optimum (het beste scenario), maar bestaat de kans dat er alleen een lokaal optimum gevonden wordt. Er is dan niet zeker of het gevonden optimum ook daadwerkelijk het globale optimum is. Bij deterministische technieken bestaat wel de garantie dat het globale optimum gevonden wordt.

Om te kunnen garanderen dat de optimale combinatie van PAS-meldingen gelegaliseerd wordt, waarbij de resterende depositieruimte minimaal is, is dus een deterministische techniek nodig voor het oplossen van het optimalisatieprobleem. Een voorbeeld van een deterministische techniek is Linear Programming; de techniek die ook op dit optimalisatieprobleem toegepast is.

1.4 Linear Programming

1.4.1 Definitie

Een Linear Program is een optimalisatiemodel dat opgebouwd is uit een aantal componenten:

1. Constanten, parameters en variabelen: Deze drie componenten vormen samen de invoer voor het model. Constanten en parameters zijn vaststaande waarden, de variabelen zijn degenen die geoptimaliseerd kunnen worden tijdens het oplossen van het optimalisatiemodel.
2. Voorwaarden waaraan de oplossing moet voldoen: De voorwaarden vertellen het model welke oplossingen geldig zijn en welke niet. Deze voorwaarden worden beschreven met behulp van de constanten, de parameters en de variabelen.
3. De doelfunctie: elke Linear Program bevat één doelfunctie. Deze functie beschrijft welke oplossing gezien wordt als optimaal.

De voorwaarden en de doelfunctie moeten beschreven worden met lineaire functies, vandaar de naam Linear Programming. Wanneer de variabelen alleen gehele getallen mogen aannemen, spreken we van een Integer Linear Program (ILP). Mogen de variabelen alleen 0 of 1 zijn (dit heet binair) dan spreekt men van Binary Integer Programming (BIP). Zowel ILP als BIP is dus een vorm van Linear Programming, waarbij een extra voorwaarde gesteld wordt aan de variabelen.

1.4.2 Oplossingstechnieken

Er zijn verschillende technieken beschikbaar waarmee een Linear Program opgelost kan worden. Deze technieken kunnen opzichzelfstaand, maar ook in combinatie met elkaar gebruikt worden. Het doel van alle oplossingstechnieken is het vinden van de optimale oplossing. Ongeacht welke oplossingstechniek gebruikt wordt, zal dezelfde optimale oplossing gevonden worden. Wanneer gebruik gemaakt wordt van een solver om de optimale oplossing van een Linear Program te berekenen, zal deze solver zelf bepalen welke oplossingstechniek het beste toepast kan worden. Welke oplossingstechnieken gebruikt worden, is dus afhankelijk van het probleem dat opgelost moet worden. Hieronder worden een aantal technieken beschreven die ontwikkeld zijn voor het oplossen van een Linear Program.

1.4.2.1 Simplexmethode

In het geval dat een Linear Program een optimale oplossing heeft, zijn er altijd één of meer combinaties beslissingsvariabelen die voldoen aan de voorwaarden gesteld in het Linear Program. Gebaseerd op de waarde van de doelfunctie, wordt één van deze combinaties bestempeld als de optimale oplossing. In het geval dat er n beslissingsvariabelen zijn, kunnen alle geldige combinaties beslissingsvariabelen in een n-dimensionale ruimte geplot worden. Er ontstaat dan een figuur die de oplossingsruimte wordt genoemd. Binnen deze ruimte bevinden zich alle potentiële oplossingen van het Linear Program. De optimale oplossing bevindt zich altijd in één van de hoekpunten van de oplossingsruimte.

De Simplexmethode maakt slim gebruik van deze eigenschap van een Linear Program, door over de randen van de oplossingsruimte van hoekpunt naar hoekpunt te lopen, waarbij elk volgende hoekpunt steeds een betere waarde voor de doelfunctie moet opleveren. Uiteindelijk zal het niet meer mogelijk zijn om door te lopen naar een volgend hoekpunt, omdat het optimale hoekpunt bereikt is. In dat geval is dus de optimale oplossing voor het Linear Program gevonden.

1.4.2.2 Branch-en-Bound

De Branch-en-Bound-techniek gaat uit van de gedachte dat de oplossingsruimte gezien kan worden als een boom. Alle takken van deze boom staan voor (delen van) mogelijke oplossing van het Linear Program. Voor elke tak van de boom is het mogelijk om een onder- en een bovengrens vast te stellen voor de oplossingswaarde die bereikt kan worden. Met behulp van deze onder- en bovengrenzen is het mogelijk om de boom te 'snoeien'. Voordat alle oplossing van een tak worden doorgerekend, worden de onder- en bovengrens van de tak vergeleken met de huidige beste oplossing. Als blijkt dat de tak geen betere oplossing kan bevatten, hoeft heel de tak niet meer doorgerekend te worden. Op deze manier wordt de oplossingsruimte dus sterk verkleind. Op het moment dat het niet meer mogelijk is om de boom verder te 'snoeien', worden alle overige oplossingen met elkaar vergeleken, of wordt een andere oplossingstechniek toegepast.

1.4.2.3 Cutting Planes

De Cutting-Planes-techniek is specifiek ontwikkeld voor het oplossen van een Integer Linear Program. Omdat een Binary Integer Program een speciale vorm van een Integer Linear Program is, kan deze techniek dus ook toegepast worden bij het oplossen van een Binary Integer Program.

Bij het oplossen van een Integer Linear Program is de kans groot dat er een 'optimale' oplossing gevonden wordt die voldoet aan alle voorwaarden van het model, behalve de voorwaarde dat de beslissingsvariabelen gehele getallen moeten zijn. Is dit het geval, dan kan een zogenaamd 'cutting plane' gevonden worden. Dit 'cutting plane' verdeelt de oplossingsruimte in tweeën, waarbij bekend is in welk van de twee delen de optimale oplossing zich bevindt. Alle potentiële oplossingen die zich in het andere deel van de oplossingsruimte bevinden, vallen af en worden niet meer meegenomen in de zoektocht naar de optimale oplossing. Ook voor deze oplossingsmethode geldt dat op het moment dat het niet meer mogelijk is om een 'cutting plane' te creëren, alle overige oplossingen met elkaar vergeleken worden, of gebruik gemaakt wordt van een andere oplossingstechniek.

1.5 Het ontwikkelde computermodel

Vanwege de aard van het probleem en de wens van reproduceerbaarheid van de oplossing, is gekozen voor het gebruik van Binary Integer Programming om tot een optimale oplossing te komen.

1.5.1 Invoer

De invoer van het Binary Integer Programming-model bestaat uit twee onderdelen:

1. Informatie over de benodigde stikstofruimte: deze informatie wordt geleverd in de vorm van GML-bestanden. Elke GML bevat voor één PAS-melding informatie over de benodigde stikstofruimte in verschillende gebieden.
2. Informatie over de beschikbare stikstofruimte: deze informatie wordt eveneens geleverd in een GML-bestand. De GML bevat voor elk gebied informatie over de beschikbare stikstofruimte.

1.5.2 Binary Integer Programming

Het Binary Integer Programming-model dat is ontwikkeld, bestaat zoals hierboven uitgelegd uit drie onderdelen:

1. *Constanten, parameters en variabelen*: constanten en parameters hebben betrekking op het aantal hexagonen en het aantal PAS-meldingen dat gelegaliseerd kan worden. Voor elke PAS-melding is één variabele toegevoegd aan het model. Omdat gebruik gemaakt is van Binary Integer Programming, kan elke variabele een waarde aannemen van 1 of van 0 in de oplossing van het optimalisatiemodel. Aan de hand van de waarde van de variabelen kan voor elke PAS-melding bepaald worden of deze valt binnen de optimale combinatie van PAS-meldingen. Bij een waarde van 1, zorgt legaliseren van de betreffende PAS-melding voor een minimale resterende depositieruimte. Bij een waarde van 0 moet de PAS-melding niet gelegaliseerd worden.
2. *Voorwaarden waaraan de oplossing moet voldoen*: Voor elk gebied geldt dat de beschikbare stikstofruimte niet overschreden mag worden. Voor elk gebied is daarom een voorwaarde toegevoegd aan het model die hier zorg voor draagt.
3. *Doelfunctie*: de doelfunctie zorgt ervoor dat de combinatie aan PAS-meldingen waarbij legaliseren zorgt voor de minste resterende depositieruimte aangemerkt wordt als de optimale oplossing van het optimalisatieprobleem.

De implementatie van het model is gedaan in Python, waarin toolboxen beschikbaar zijn waarmee Binary Integer Programming modellen opgelost kunnen worden.

Doordat gekozen is voor Binary Integer Programming, kan gegarandeerd worden dat de oplossing van het optimalisatiemodel optimaal gebruik maakt van de beschikbare depositieruimte en dus de resterende depositieruimte minimaal maakt. Daarnaast is deze oplossing reproduceerbaar.

1.5.3 Optimale oplossing selecteren

Het algoritme levert standaard de optimale oplossing van het probleem, maar voor test- en analysedoeleinden is het mogelijk om ook sub-optimale oplossingen uit te voeren waarbij de resterende depositieruimte in percentage minder dan een bepaalde drempelwaarde afwijkt van de optimale minimale hoeveelheid resterende depositieruimte. In het geval dat er meerdere oplossingen mogelijk zijn voor het optimalisatieprobleem - in dit geval zijn er meerdere combinaties van PAS-meldingen waarbij de resterende depositieruimte minimaal is, of is ook het vinden van sub-optimale oplossingen toegestaan - worden de regels toegepast die genoemd zijn in paragraaf 1.2. Door het gebruik van de volgorde van ontvangen van de PAS-melding wordt het verkrijgen van een unieke oplossing gegarandeerd.

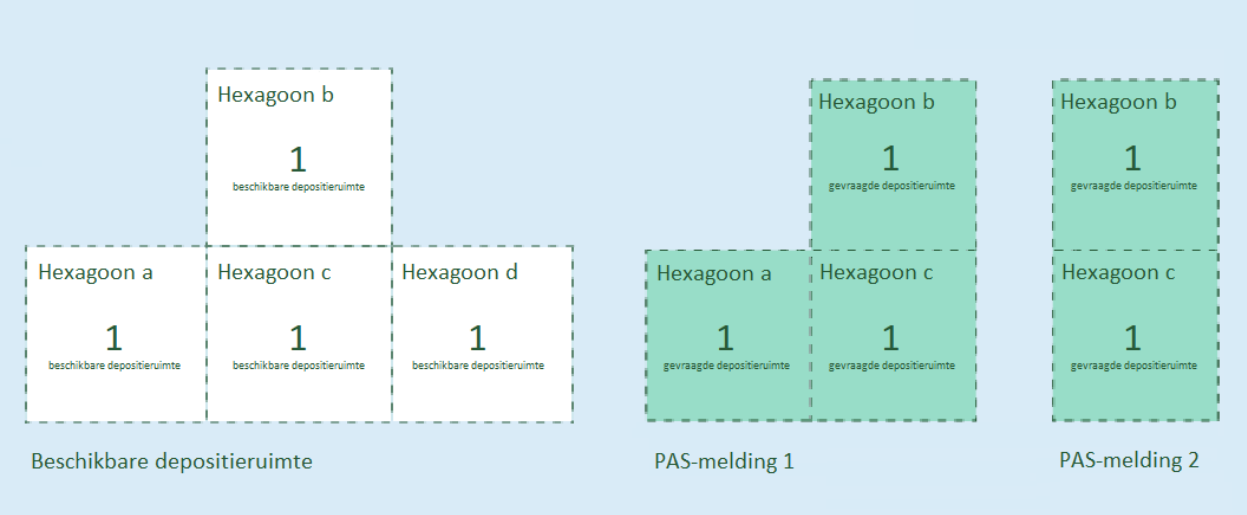
1.5.4 Uitvoer

Het computermodel levert drie soorten uitvoer in tekstbestanden:

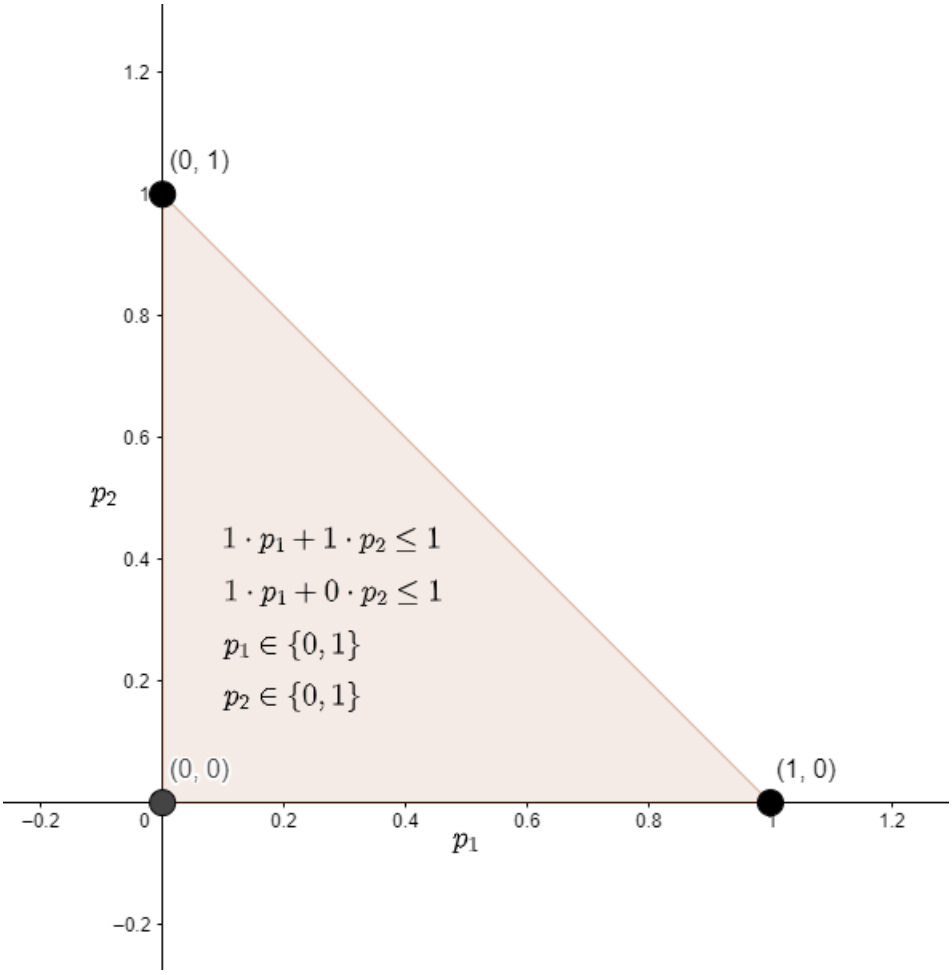
- `Oplossing_van_de_stikstofpuzzel.txt`: dit bestand bevat de (enige) optimale oplossing van de stikstofpuzzel, in de vorm van de ID's van de PAS-meldingen die samen zorgen voor een optimaal gebruik van de beschikbare stikstofruimte en daarmee een minimale resterende depositieruimte opleveren.
- `Afgevallen_oplossingen.txt`: in het geval dat het Binary-Integer-Program-model meerdere (sub-optimale) oplossingen levert, bevat dit bestand voor alle niet-optimale oplossingen de reden op basis waarvan de oplossing is afgekeurd als optimale oplossing.
- `Stikstofpuzzel<datum>-<tijd>.txt`: De logging die gegenereerd is tijdens het uitvoeren van het model.

1.6 Voorbeeld

Hieronder is een voorbeeld opgenomen waarin duidelijk gemaakt wordt hoe met behulp van Binary Linear Programming een oplossing gevonden kan worden voor de stikstofpuzzel. Voor het voorbeeld wordt uitgegaan van 4 hexagonen waarbinnen per hexagoon 1 stikstofruimte beschikbaar is. Verder zijn er 2 PAS-meldingen passend binnen de beschikbare stikstofruimte. De eerste PAS-melding bevat 3 hexagonen, de tweede PAS-melding bevat 2 hexagonen. Beide meldingen vragen in 1 stikstofruimte per hexagoon.



De oplossingsruimte van dit probleem ziet er als volgt uit, waarbij p_1 en p_2 staan voor PAS-melding 1 en PAS-melding 2, respectievelijk:



De vergelijkingen die weergegeven zijn in bovenstaande afbeelding vormen de voorwaarden van het Binary-Integer-Program-model. In het geval dat PAS-meldingen gedeeltelijk gelegaliseerd zouden kunnen worden - en een niet-geheeltallige oplossing dus toegestaan zou zijn - bevat het oranje vlak alle mogelijke oplossingen voor de stikstofpuzzel. Omdat bovenstaande niet het geval is, bestaat de oplossingsruimte slechts uit de drie zwarte stippen, waarbij beslissingsvariabelen p_1 en p_2 alleen 0 of 1 mogen zijn.

De optimale oplossing ligt in één van de hoekpunten van de oplossingsruimte.

Wordt gebruik gemaakt van de Simplexmethode om bovenstaand probleem op te lossen, dan zouden één voor één de hoekpunten afgelopen worden, totdat geen van de aanliggende hoekpunten een hogere waarde voor de doelfunctie oplevert. In dat geval is de optimale oplossing gevonden.

In het geval van het voorbeeld heeft de oplossingsruimte drie hoekpunten en dus zijn er drie kandidaten voor de optimale oplossing, namelijk:

1. $(p_1 = 0, p_2 = 0)$: beide PAS-meldingen worden niet gelegaliseerd. Dit levert een resterende depositieruimte van 4 op;
2. $(p_1 = 1, p_2 = 0)$: alleen PAS-melding 1 wordt gelegaliseerd. In dit geval blijft er een resterende depositieruimte van 1 over;
3. $(p_1 = 0, p_2 = 1)$: alleen PAS-melding 2 wordt gelegaliseerd. Dit resulteert in een resterende depositieruimte van 2.

Op basis van bovenstaande opsomming kan geconcludeerd worden dat optie 2 de minste resterende depositieruimte oplevert. Het is in dit geval dus optimaal om alleen PAS-melding 1 te legaliseren.

1.7 Runnen van de tool

De tool kan op twee manieren worden gedraaid, namelijk vanuit de broncode of vanuit een Docker-container. Voor beide manieren van draaien is een configuratiebestand nodig. Het bestand `stikstof_puzzel/config.env.example` dient als voorbeeld voor het configuratiebestand, en moet worden aangepast overeenkomend met de lokale situatie.

1.7.1 Vanuit de broncode

Eerst moet de broncode naar de lokale machine worden gehaald door middel van een git clone:

```
git clone https://gitlab.com/witteveenbos/stikstof-puzzel.git
```

Geadviseerd wordt om vervolgens de Python-omgeving door middel van conda te beheren.

```
conda create --name pas python=3.9 --yes
conda activate pas
pip install -r requirements.txt
```

Het configuratie bestand `stikstof_puzzel/config.env` moet worden aangemaakt zodat het correspondeert met de situatie van de lokale machine. Zie voor een voorbeeld het bestand `stikstof_puzzel/config.env.example`. De tool kan worden gedraaid door de `main.py` aan te roepen vanuit de map stikstof_puzzel:

```
cd stikstof_puzzel
python main.py
```

1.7.2 Vanuit een Docker-container

De Docker-image moet worden gepulled van de repository.

```
docker login --username <username> --password <password> registry.gitlab.com
docker pull registry.gitlab.com/witteveenbos/stikstof-puzzel/tool:latest
```

De image kan worden gedraaid waarbij configuratie en volume-mount wordt meegegeven:

```
docker run -it --env-file config_docker.env \
  --volume /my/local/data/path:/data \
  registry.gitlab.com/witteveenbos/stikstof-puzzel/tool:latest
```

Het bestand `config_docker.env` moet worden aangemaakt zodat het correspondeert met de situatie van de lokale machine. Zie voor een voorbeeld het bestand `stikstof_puzzel/config.env.example`. De string `/my/local/data/path` moet corresponderen met de map waarin de data van de PAS-meldingen en de Srv staan.

1.8 Disclaimer

De tool is ontwikkeld door Witteveen+Bos in opdracht van het RIVM. De opdracht is uitgevoerd onder de voorwaarden zoals deze beschreven staan in de betreffende aanbieding van 7 april 2022, getiteld: *'Ontwikkeling optimalisatietool voor afboekingen PAS-meldingen'* [ref: 131389/22-005.227].

CHAPTER

TWO

TECHNICAL DOCUMENTATION: STIKSTOF_PUZZEL

Om de stikstofpuzzel op te lossen is gebruik gemaakt van Binary Integer Programming. Het rekenmodel is geprogrammeerd in Python. Onderstaande paragrafen bevatten een uitgebreide beschrijving van de functies de gebruikt zijn om de combinatie PAS-meldingen te vinden waarbij legaliseren zorgt voor een minimale stikstofrestruimte.

2.1 main.py

Dit script koppelt al de functies uit de andere scripts aan elkaar, waardoor de stikstofpuzzel opgelost wordt. Kortgezegd werkt dit script als volgt:

1. De data (PAS-meldingen en Srv) wordt ingelezen (m.b.v. read_gml.py);
2. De data wordt getransformeerd naar invoer die geschikt is om ingevoerd te worden in het Binary-Integer-Program-model;
3. Het Binary-Integer-Program-model wordt opgelost;
4. De oplossingen uit het Binary-Integer-Program-model worden verwerkt, om zo tot een optimale oplossing te komen;
5. De oplossing en bijbehorende argumentatie worden weggeschreven naar een bestand in de uitvoer-map.

2.2 read_gml.py

De invoerdata voor de stikstofpuzzel wordt geleverd in GML's, een specifiek soort dataformat. Dit script bevat functies die nodig zijn voor het inlezen van deze GML-bestanden en om de benodigde invoer voor het BIP-model eruit te halen. Het script bevat vier functies en één class:

- obtain_filepath_and_threshold_from_env: deze functie leest de benodigde config-parameters, zoals paden waar data opgeslagen staat, in uit het configbestand;
- read_data: deze functie leest GML's in vanuit het opgegeven datapad. Deze GML's bevatten zowel de stikstofvraag vanuit de PAS-meldingen als de stikstofruimte die gecreëerd is binnen de Srv;
- check_input: deze functie checkt de data die ingelezen is uit een GML;
- extract_features_from_gml: deze functie leest alle data uit één GML in;
- class ImaerRead: deze class wordt gebruikt om door de attributen van een GML heen te lopen.

class stikstof_puzzel.read_gml.**ImaerRead**(*gml_file*)

Deze class wordt gebruikt om door alle features in een GML-bestand heen te kunnen lopen. Om een volgend feature te bekijken, wordt de 'nextFeature'-methode gebruikt.

next_feature()

Met deze functie is het mogelijk om door de ReceptorPoints in de GML heen te lopen en zo de data te bekijken per ReceptorPoint.

stikstof_puzzel.read_gml.**check_input**(*notification_meta*, *features*, *notification=True*, *final_check=False*)

Deze functie checkt het format van de Srv- en PAS-meldingendata. Bij het checken wordt gelet op de volgende punten:

- De PAS-meldingen-id's moeten allemaal van het type 'str' zijn;
- De hexagoon-id's uit de PAS-meldingen moeten allemaal uniek en van het type 'str' zijn;
- De databaseposities van de PAS-meldingen moeten uniek zijn, en van het type integer;
- De gevraagde stikstofruimte is van het type 'float' of 'int' en is niet negatief;
- De Srv-hexagonen moeten allemaal uniek en van het type 'str' zijn;
- De beschikbare stikstofruimte is van het type 'int' of 'float' en is niet negatief.

Parameters

notification_meta
[tuple of list] Informatie over de PAS-melding, in de vorm van een tuple (PAS-melding-id, databasepositie). Indien het gaat om de laatste check, bevat deze parameter een lijst van dergelijke tuples.

features
Een lijst met tuples (hexagoon-id, stikstofruimte) horend bij de PAS-melding die gecheckt wordt.

notification
[bool, default True] Geeft aan of de data die gecheckt moet worden afkomstig is uit een PAS-melding-GML of uit de Srv-GML. In beide gevallen worden andere checks uitgevoerd.

final_check
[bool, default False] Geeft aan of het gaat om de laatste check van alle ingelezen data, of om een check van één ingelezen GML-bestand. In beide gevallen worden andere checks uitgevoerd.

Raises

TypeError
Als een PAS-melding-id of een hexagoon-id niet van het type 'str' is

ValueError
Als een PAS-melding negatieve stikstofruimte vraag of als een hexagoon negatieve beschikbare stikstofruimte heeft.

KeyError
Als de databaseposities van de PAS-meldingen niet uniek zijn.

stikstof_puzzel.read_gml.**extract_features_from_gml**(*gml_filepath: str*, *notifications=True*)

Met deze functie kunnen gegevens uit een GML gelezen worden. Twee soorten GML's kunnen gelezen worden:

- Een PAS-meldingen-GML: deze bevat informatie over de benodigde stikstofruimte in verschillende hexagonen;
- De Srv-GML: deze bevat informatie over de beschikbare stikstofruimte in verschillende hexagonen.

Parameters

gml_filepath
[str] De locatie van het GML-bestand.

notifications: bool
Geeft aan of het opgegeven GML-bestand PAS-meldingen of informatie over de Srv bevat. True geeft aan dat het gaat om een PAS-melding, False geeft aan dat het gaat om informatie over de Srv.

Returns

(notification_id, database_position)
[tuple] PAS-melding-id en databasepositie van de PAS-melding.

hexagon_feature_list
[list] Lijst met tuples (hexagoon_id, stikstofruimte).

Raises

ValueError
Als de databasepositie geen geheel getal is.

stikstof_puzzel.read_gml.**obtain_filepath_and_threshold_from_env**(*env_name*)

Alle 'veranderlijke' parameters staan opgeslagen in een .env bestand. Dit bestand moet een gebruiker zelf aanmaken. Om te weten hoe het bestand eruit moet komen te zien, is het bestand 'config.env.example' toegevoegd aan de repository.

Het .env-bestand bevat drie parameters, namelijk:

- DATA_FILEPATH: Het pad waar de map met invoerdata zich bevindt;
- EXPORT_DIR: Het pad waar de uitvoer van het model naartoe geschreven moet worden;
- THRESHOLD: De drempelwaarde die gebruikt wordt bij het selecteren van (sub)optimale oplossingen van het BIP-model. Als deze waarde niet gegeven wordt, wordt de threshold op 0.0 gezet.

Deze functie wordt gebruikt om de parameters uit het .env-bestand binnen te halen, zodat alle functies gebruik kunnen maken van deze parameters. Als het .env-bestand niet bestaat, dan wordt aangenomen dat de veranderlijke parameters al zijn aangemaakt in de omgeving waar het algoritme draait.

Parameters

env_name
[str] De naam van het .env-bestand.

Returns

filepath_data
[str] Het pad naar de map die de invoerdata bevat.

export_dir
[str] Het pad naar de map waar de uitvoer in weggeschreven moet worden.

threshold
[float] De drempelwaarde die gebruikt wordt bij het selecteren van oplossingen van het BIP-model.

Raises

KeyError
Als één van de parameters DATA_FILEPATH of EXPORT_DIR niet gedefinieerd is.

TypeError
Als de drempelwaarde niet als getal gegeven is.

NotADirectoryError
Als de map die de invoerdata bevat niet gevonden kan worden.

`stikstof_puzzel.read_gml.read_data(`*filepath_data*`)`

Deze functie kan gebruikt worden om data in te lezen vanuit de PAS-meldingen-GML's en de Srv-GML. Deze GML's moeten zich bevinden in de map die opgegeven kan worden als argument in deze functie. De opgegeven map moet twee mappen bevatten, namelijk:

- de map 'Meldingen': deze map bevat alle PAS-meldingen-GML's;
- de map 'SRV': deze map bevat de Srv-GML.

Parameters

filepath_data
[str] Het pad waar de map met invoerdata zich bevindt.

Returns

notifications
[dict] Python-dictionary met informatie over de PAS-meldingen. De keys van de dictionary zijn tuples (melding-id, databasepositie). De values van de dictionary zijn lijsten met tuples (hexagoon-id, benodigde stikstofruimte).

srv
[list] Lijst met per hexagoon aangegeven hoeveel stikstofruimte er beschikbaar is. Alle elementen uit deze lijst zijn tuples (hexagoon-id, beschikbare stikstofruimte).

Raises

ValueError
Als een PAS-melding-GML geen informatie bevat over de databasepositie van de PAS-melding.

KeyError
Als een PAS-melding-id vaker voorkomt in de lijst met PAS-meldingen-GML's.

2.3 preprocess_input.py

Om de invoerdata te kunnen gebruiken om een Binary-Integer-Program-model te definiëren, moet de invoer het geschikte format hebben. Dit script bevat de functies die hiervoor gebruikt kunnen worden.

Het gaat om de volgende functies:

- transform_input_data: deze functie transformeert de GML-data naar invoer die geschikt is om in te voeren in het BIP-model;
- df_to_input: deze functie wordt aangeroepen binnen de functie 'transform_input_data' en transformeert een dataframe naar het format dat geschikt is om in te voeren in het BIP-model.

`stikstof_puzzel.preprocess_input.df_to_input(`*input_data*, *get_total_need=False*`)`

Deze functie wordt gebruikt om invoer te creëren voor het BIP-model vanuit een dataframe dat informatie over de benodigde en de beschikbare stikstofruimte bevat. Het dataframe dat in deze functie ingevoerd wordt, moet aan een bepaald format voldoen, namelijk:

- rijen: de rijlabels zijn de id's van de hexagonen. Elk hexagoon heeft dus zijn eigen rij. Alle hexagonen die in de Srv voorkomen zijn onderdeel van het dataframe.

- kolommen: het dataframe moet een kolom bevatten met de naam 'nitrogen_available'. Deze kolom bevat voor alle hexagonen de beschikbare stikstofruimte. De overige kolommen van het dataframe bevatten per PAS-melding de benodigde stikstofruimte per hexagoon. De naam van deze kolommen is telkens een tuple: (PAS-melding-id, databasepositie). De databasepositie wordt bepaald door het moment waarop de PAS-melding is ingediend bij RVO.

Onder het kopje *examples* is een voorbeeld van een dataframe opgenomen dat in deze functie ingevoerd kan worden.

Parameters

input_data
[pd.DataFrame] Dit dataframe bevat alle informatie waarmee de stikstofpuzzel opgelost kan worden, namelijk de beschikbare stikstofruimte en de benodigde stikstofruimte voor de PAS-meldingen.

get_total_need
[bool, default False] Bepaalt de uitvoer van deze functie. Indien get_total_need True is, wordt een lijst met totaal benodigde stikstofruimte per PAS-melding gegenereerd. Indien get_total_need False is, wordt de benodigde stikstofruimte per PAS-melding opgesplitst in benodigde stikstofruimte per hexagoon.

Returns

total_need
[list] Lijst met per PAS-melding de totaal benodigde stikstofruimte. De lijst staat op volgorde van de kolommen in het dataframe.

nitrogen_available
[list] De beschikbare stikstofruimte per hexagoon. De lijst staat op volgorde van de rijen in het dataframe.

nitrogen_needed
[list] Voor elke PAS-melding een lijst met per hexagoon aangegeven hoeveel stikstofruimte er nodig is. De hexagonen moeten in dezelfde volgorde staan als in de lijst 'nitrogen_available'. De PAS-meldingen moeten in dezelfde volgorde staan als in de lijst 'total_need'.

notifications
[int] Aantal PAS-meldingen die opzichzelfstaand past binnen de beschikbare stikstofruimte.

hexagons
[int] Aantal relevante hexagonen waarin stikstofruimte gevraagd wordt door een PAS-melding. Met andere woorden: hexagonen die geen PAS-melding bevatten die gelegaliseerd kan worden, worden niet meegenomen bij het opstellen van het model. Ook hexagonen waarvoor de beschikbare stikstofruimte niet overschreden wordt als alle PAS-meldingen gelegaliseerd zouden worden, worden niet meegenomen in het model.

notifications_list
[list] Lijst met alle relevante PAS-meldingen-id's en de bijbehorende databaseposities. De PAS-meldingen-id's zijn afgeleid uit de bestandsnamen van de PAS-meldingen-GML's, de databasepositie is gebaseerd op het moment waarop de PAS-melding is ingediend bij RVO.

Examples

```
>>> input_data
            nitrogen_available  ('PAS-melding 1', 20) ... ('PAS-melding n', 4)
hexagoon-id
'hex 1'                  1.0                 0.8          ...          0.12
'hex 2'                  2.0                 1.2          ...          1.3
'hex 3'                  4.297               3.0          ...          2.1
   :                      :                   :                         :
'hex m'                  0.34                0.3          ...          0.23
```

stikstof_puzzel.preprocess_input.**transform_input_data**(*notifications: dict*, *srv: list*)

Deze functie transformeert de data die ingelezen is vanuit de GML's naar invoer die ingevoerd kan worden in het BIP-model. Een dataframe wordt gecreëerd om alle invoer aan elkaar te koppelen. Vervolgens wordt dit dataframe met behulp van functie 'df_to_input' omgezet in invoer voor het BIP-model.

Parameters

notifications
[dict] Python-dictionary met informatie over de PAS-meldingen. De keys van de dictionary zijn tuples (melding-id, databasepositie). De values van de dictionary zijn lijsten met tuples (hexagoon-id, benodigde stikstofruimte).

srv
[list] Lijst met per hexagoon aangegeven hoeveel stikstofruimte er beschikbaar is. Alle elementen uit deze lijst zijn tuples (hexagoon-id, beschikbare stikstofruimte).

Returns

nitrogen_available
[list] De beschikbare stikstofruimte per hexagoon.

total_need
[list] Lijst met per PAS-melding de totale stikstofvraag.

nitrogen_needed
[list] Voor elke PAS-melding een lijst met per hexagoon aangegeven hoeveel stikstofruimte er nodig is. De hexagonen moeten in dezelfde volgorde staan als in de lijst 'nitrogen_available'. De PAS-meldingen moeten in dezelfde volgorde staan als in de lijst 'total_need'.

no_notifications
[int] Aantal PAS-meldingen dat opzichzelfstaand past binnen de beschikbare stikstofruimte.

no_hexagons
[int] Aantal relevante hexagonen waarin stikstofruimte gevraagd wordt door een PAS-melding. Met andere woorden: hexagonen die geen PAS-melding bevatten die gelegaliseerd kan worden, worden niet meegenomen bij het opstellen van het model. Ook hexagonen waarvoor de beschikbare stikstofruimte niet overschreden wordt als alle PAS-meldingen gelegaliseerd zouden worden, worden niet meegenomen in het model.

notifications_list
[list] Lijst met alle relevante PAS-meldingen-id's en de bijbehorende databaseposities. De PAS-meldingen-id's zijn afgeleid uit de bestandsnamen van de PAS-meldingen-GML's, de databasepositie is gebaseerd op het moment waarop de PAS-melding is ingediend bij RVO.

Raises

TypeError
Als het format van de PAS-meldingen (parameter notifications) of van de Srv-data (parameter

srv) niet klopt.

ValueError
Als een PAS-melding is opgegeven die negatieve stikstofruimte vraagt of wanneer een hexagoon is opgegeven waarin negatieve stikstofruimte beschikbaar is. Negatieve stikstofruimte zorgt potentieel voor een onoplosbare puzzel en/of een verkeerde uitkomst.

2.4 binary_integer_program.py

Het oplossen van de stikstofpuzzel wordt gedaan met behulp van een zogenaamd Binary-Integer-Program-model. Dit script bevat de functies die nodig zijn voor het definiëren en oplossen van het model.

Het gaat om de volgende functies:

- binary_integer_program: deze functie definieert het Binary-Integer-Program-model waarmee de stikstofpuzzel kan worden opgelost;
- get_all_solutions: deze functie wordt gebruikt om de *n* beste oplossingen te vinden voor de stikstofpuzzel;
- get_solution: deze functie wordt gebruikt om na het oplossen van het Binary-Integer-Program-model de oplossing te filteren uit de modeluitvoer.

`stikstof_puzzel.binary_integer_program.`**`binary_integer_program`**(*notifications*, *hexagons*, *nitrogen_available*, *total_need*, *nitrogen_needed*)

Met deze functie wordt het Binary-Integer-Program-model gedefinieerd aan de hand van gegeven invoer.

Het model ziet er als volgt uit:

constanten

m	aantal PAS-meldingen
h	aantal hexagonen

variabelen

p_i	binair, waarbij $i \in \{1, \ldots, m\}$. $p_i = 1$ geeft aan dat PAS-melding i gelegaliseerd kan worden

parameters

r_j	stikstofruimte in hexagoon j, waarbij $j \in \{1, \ldots, h\}$
t_i	totale stikstofvraag van PAS-melding i, waarbij $i \in \{1, \ldots, m\}$
v_{ij}	stikstofvraag van PAS-melding i in hexagoon j, waarbij $i \in \{1, \ldots, m\}$ en $j \in \{1, \ldots, h\}$

binary integer program

$$\begin{aligned} &\textbf{maximaliseer} && \sum_{i=1}^{m} p_i \cdot t_i \\ &\textbf{onder voorwaarden} && \sum_{i=1}^{m} p_i \cdot v_{ij} \leq r_j, \quad \forall j \in \{1, \ldots, h\} \\ & && p_i \in \{0, 1\}, \quad \forall i \in \{1, \ldots, m\} \end{aligned}$$

De invoer voor het BIP-model bestaat uit drie onderdelen:

- Constanten: het aantal relevante PAS-meldingen en hexagonen. Niet alle PAS-meldingen en niet alle hexagonen worden in het BIP-model ingevoerd. Alleen PAS-meldingen die binnen de beschikbare stikstofruimte passen worden toegevoegd. PAS-meldingen die niet binnen de beschikbare stikstofruimte passen, worden dus niet meegenomen in het model. Voor de hexagonen geldt dat alleen hexagonen waarbinnen passende PAS-meldingen vallen worden toegevoegd, mits er een conflict zou ontstaan binnen dit hexagoon als alle beschikbare PAS-meldingen gelegaliseerd zouden worden. Hexagonen waarbinnen alle passende PAS-meldingen gelegaliseerd kunnen worden zonder de beschikbare ruimte te overschrijden, worden dus niet toegevoegd aan het BIP-model, om het model niet onnodig groot te maken;

- Parameters: voor elke hexagoon de beschikbare stikstofruimte, en voor elke PAS-melding zowel de totale stikstofvraag als per hexagoon de gevraagde stikstofruimte.
- Variabelen: voor elke PAS-melding één. Elke variabele kan de waarde 0 of 1 krijgen, vandaar de naam *binary* integer programming. Als een variabele in de oplossing een waarde van 1 heeft, betekent dit dat de PAS-melding onderdeel is van de combinatie aan PAS-meldingen waarbij optimaal gebruik gemaakt wordt van de stikstofruimte. Een waarde van 0 betekent dat dit de PAS-melding hier geen onderdeel van is;

Het BIP-model bestaat uit een doelfunctie en uit voorwaarden:

- De doelfunctie zorgt ervoor dat de combinatie aan PAS-meldingen die zoveel mogelijk stikstofruimte gebruikt, gekozen wordt als optimale oplossing. Zoveel mogelijk stikstofruimte gebruiken staat gelijk aan zo weinig mogelijk stikstofrestruimte overlaten;
- Voor elk hexagoon is een voorwaarde toegevoegd die ervoor zorgt dat de totale hoeveelheid beschikbare stikstofruimte niet overschreden wordt door de PAS-meldingen in de oplossing.

Als het BIP-model gedefinieerd is, kan het gebruikt worden om de optimale oplossing voor de stikstofpuzzel te vinden. De voorwaarden bepalen welke combinaties van PAS-meldingen mogelijk zijn binnen de beschikbare stikstofruimte. Vervolgens zorgt de doelfunctie ervoor dat dat uit al deze mogelijke combinaties de combinatie gekozen wordt die de meeste stikstofruimte gebruikt. De oplossing van het BIP-model zal dus altijd, binnen de gestelde voorwaarden, de meeste stikstofruimte gebruiken, en dus ook een zo minimaal mogelijke stikstofrestruimte overhouden.

Parameters

notifications
[int] Aantal PAS-meldingen passend binnen de gecreëerde stikstofruimte.

hexagons
[int] Aantal hexagonen binnen de stikstofruimte waarin ruimte gevraagd wordt door een PAS-melding. Met andere woorden: hexagonen die geen PAS-melding bevatten die gelegaliseerd kan worden, worden niet meegenomen bij het opstellen van het model. Ook hexagonen waarbinnen alle beschikbare PAS-meldingen gelegaliseerd kunnen worden, zonder dat de beschikbare stikstofruimte overschreden wordt, worden niet meegenomen in het model.

nitrogen_available
[list] De beschikbare stikstofruimte per hexagoon.

total_need
[list] De totale stikstofvraag per PAS-melding.

nitrogen_needed
[list of lists] Voor elke PAS-melding een lijst waarin per hexagoon aangegeven is hoeveel stikstofruimte er nodig is. De volgorde van PAS-meldingen en hexagonen moet overeenkomen met de volgorde die gebruikt is bij het samenstellen van de lijsten 'nitrogen_available' en 'total_need'.

Returns

model
[pulp.pulp.LpProblem] Het Binary-Integer-Program-model. Dit model bevat de doelfunctie en de bijbehorende voorwaarden, zoals hierboven gedefinieerd.

dec_vars
[dict] De beslissingsvariabelen die horen bij het Binary-Integer-Program-model. Met behulp van de beslissingsvariabelen kan combinatie aan PAS-meldingen worden bepaald waarmee optimaal gebruik gemaakt wordt van de beschikbare stikstofruimte.

stikstof_puzzel.binary_integer_program.**get_all_solutions**(*notifications*, *hexagons*, *nitrogen_available*, *total_need*, *nitrogen_needed*, *threshold=0.0*)

Deze functie wordt gebruikt als wrapper rondom het Binary-Integer-Program-model. Aan de hand van de opgegeven drempelwaarde (parameter *threshold*) wordt bepaald of een passende combinaties van PAS-meldingen geldt als mogelijk optimale oplossing van de stikstofpuzzel. Deze functie vindt door middel van het oplossen van het Binary-Integer-Program-model alle geldige oplossingen voor de stikstofpuzzel.

Wanneer blijkt dat er meerdere geldige combinaties van PAS-meldingen zijn, kan de optimale combinatie van te legaliseren PAS-meldingen bepaald worden met behulp van functie 'postprocess_solution' uit het script 'post-process_solution.py'

Extra uitleg over de drempelwaarde:

De drempelwaarde is toegevoegd voor test- en analysedoeleinden, en wordt gebruikt om te bepalen of een passende combinatie van PAS-meldingen geldt als oplossing van de stikstofpuzzel. De drempelwaarde wordt altijd gebruikt in combinatie met de optimale oplossing van het Binary-Integer-Program-model. Deze optimale oplossing bepaalt namelijk hoeveel stikstofruimte er maximaal gebruikt gaat worden bij het legaliseren van de PAS-meldingen. Bij een minder optimale oplossing (die eveneens bestaat uit een combinatie van PAS-meldingen die past binnen de gecreëerde stikstofruimte) zal er minder stikstofruimte gebruikt worden. Is de drempelwaarde 0.1, dan worden alle combinaties van PAS-meldingen geaccepteerd waarbij de hoeveelheid gebruikte stikstofruimte maximaal 10% afwijkt van de maximaal te gebruiken stikstofruimte.

Voorbeelden:

Worden er bij het legaliseren van de optimale combinatie aan PAS-meldingen 100 eenheden stikstofruimte gebruikt, dan gelden alle combinaties PAS-meldingen die minimaal 90 eenheden stikstofruimte gebruiken ook als oplossing van de stikstofpuzzel. Bij een drempelwaarde van 0 zullen enkel de optimale combinaties van PAS-meldingen gelden als oplossing van de stikstofpuzzel. Bij een drempelwaarde van 1 worden alle passende combinaties van PAS-meldingen geaccepteerd als oplossing van de stikstofpuzzel.

Parameters

notifications
[int] Aantal PAS-meldingen passend binnen de gecreëerde stikstofruimte.

hexagons
[int] Aantal hexagonen binnen de stikstofruimte waarin ruimte gevraagd wordt door een PAS-melding. Met andere woorden: hexagonen die geen PAS-melding bevatten die gelegaliseerd kan worden, worden niet meegenomen bij het opstellen van het model. Ook hexagonen waarbinnen alle beschikbare PAS-meldingen gelegaliseerd kunnen worden, zonder dat de beschikbare stikstofruimte overschreden wordt, worden niet meegenomen in het model.

nitrogen_available
[list] De beschikbare stikstofruimte per hexagoon.

total_need
[list] De totale stikstofvraag per PAS-melding.

nitrogen_needed
[list of lists] Voor elke PAS-melding een lijst waarin per hexagoon aangegeven is hoeveel stikstofruimte er nodig is. De volgorde van PAS-meldingen en hexagonen moet overeenkomen met de volgorde die gebruikt is bij het samenstellen van de lijsten 'nitrogen_available' en 'total_need'.

threshold
[float, default 0.0] Drempelwaarde die aangeeft hoeveel procent de oplossingswaarde (=gebruikte stikstofruimte) mag afwijken van de optimale oplossingswaarde, om te gelden als

geaccepteerde oplossing van de stikstofpuzzel. Bijvoorbeeld: is de threshold 0.01, dan worden alle oplossingen waarvan de oplossingswaarde maximaal 1% afwijkt van de optimale oplossingswaarde ook geaccepteerd als oplossing van de stikstofpuzzel.

Returns

solutions
[list] Lijst met alle combinaties aan PAS-meldingen die passen in de beschikbare stikstofruimte en tegelijkertijd voldoen aan het door middel van de drempelwaarde gestelde criteria.

stikstof_puzzel.binary_integer_program.**get_solution**(*dec_vars*)

Deze functie zet de waarden van de beslissingsvariabelen van het Binary-Integer-Program-model om in een combinatie aan PAS-meldingen die volgens het BIP-model past binnen de beschikbare stikstofruimte. Elke beslissingsvariabele is gekoppeld aan één PAS-melding, en heeft een waarde van 1 of 0:

- Een waarde van 1 betekent dat de PAS-melding volgens de oplossing WEL gelegaliseerd kan worden;
- Een waarde van 0 betekent dat de PAS-melding volgens de oplossing NIET gelegaliseerd kan worden.

De beslissingsvariabelen zijn bepaald door het oplossen van het Binary-Integer-Program-model. Na het oplossen van het model bevatten de beslissingsvariabelen de optimale combinatie PAS-meldingen die voldoet aan de in het model gestelde voorwaarden. Optimaal betekent in dit geval dat er zo weinig mogelijk stikstofrestruimte overblijft na het legaliseren van de geselecteerde PAS-meldingen.

Parameters

dec_vars
[dict] Lijst met de beslissingsvariabelen. De beslissingsvariabelen hebben altijd een waarde van 0 of 1, en worden berekend met behulp van het BIP-model.

Returns

picked
[list] Lijst met de PAS-meldingen die passen binnen de beschikbare stikstofruimte.

2.5 postprocess_solution.py

Nadat het BIP-model is uitgevoerd zijn de mogelijke oplossingen van de stikstofpuzzel bekend. Dit script bevat vier functies waarmee uit deze mogelijke oplossingen een optimale oplossing geselecteerd wordt aan de hand van twee regels:

- Zijn er meerdere oplossingen mogelijk, dan wordt gekozen voor de oplossing waarbij het grootste aantal PAS-meldingen gelegaliseerd wordt.
- Zijn er meerdere oplossingen die een maximaal aantal PAS-meldingen legaliseren, dan wordt gekeken naar de positie van alle PAS-meldingen in de database, in oplopende volgorde. Zodra een oplossing een PAS-melding niet bevat die wel in een andere oplossing voorkomt, valt deze oplossing af. Omdat alle oplossingen uniek zijn, levert dit uiteindelijk een unieke optimale oplossing.

Het gaat om de volgende functies:

- postprocess_solutions: deze functie past de hierboven gedefinieerde regels toe op de oplossingen van het BIP-model en selecteert zo de optimale oplossing van de stikstofpuzzel;
- check_solutions: deze functie checkt of er één unieke oplossing is gevonden, of dat een volgende regel moet worden toegepast;
- add_solution_argumentations: deze functie houdt voor alle oplossingen van het BIP-model bij waarom ze wel of niet geselecteerd worden als de optimale oplossing van de stikstofpuzzel;

- write_solution: deze functie schrijft de oplossing en de onderbouwing weg naar een zelf te kiezen uitvoermap.

stikstof_puzzel.postprocess_solution.**add_solution_argumentations**(*solutions*, *notifications_list*, *reason*, *no_solutions_reasons=None*)

Wanneer één van de oplossingen van het BIP-model afvalt bij het toepassen van één van de regels, zorgt deze functie ervoor dat de argumentatie voor het afvallen wordt bijgehouden. Met behulp van functie 'write_solution' kan de argumentatie - na het vinden van de optimale oplossing - weggeschreven worden naar een .txt-bestand.

Parameters

solutions
[list] De lijst met afgevallen oplossingen. Elke oplossing bevat een lijst met PAS-meldingen die gelegaliseerd kunnen worden binnen de beschikbare stikstofruimte.

notifications_list
[list] Lijst met alle relevante PAS-meldingen-id's en de bijbehorende databaseposities. De PAS-meldingen-id's zijn afgeleid uit de bestandsnamen van de PAS-meldingen-GML's, de databasepositie is gebaseerd op het moment waarop de PAS-melding is ingediend bij RVO.

reason
[str] De reden waarom de oplossingen zijn afgevallen

no_solutions_reasons
[pd.DataFrame, default None] Het dataframe waarin de afgevallen oplossingen met bijbehorende reden worden opgeslagen. Indien geen dataframe wordt ingevoerd, wordt het dataframe aangemaakt.

Returns

——-

no_solutions_reasons
Het dataframe waarin de afgevallen oplossingen met bijbehorende reden zijn opgeslagen.

Raises

TypeError
Als solution_args niet van het type None of pd.DataFrame is

stikstof_puzzel.postprocess_solution.**check_solutions**(*solution*)

Deze functie onderzoekt of er één unieke optimale oplossing gevonden is.

Parameters

solution
[list] De lijst met oplossingen van het BIP-model. Elke oplossing bevat een lijst met PAS-meldingen die gelegaliseerd kunnen worden binnen de beschikbare stikstofruimte.

Returns

solution_found
[bool] True of False. True als er één unieke oplossing voor de stikstofpuzzel gevonden is, False als de lijst meerdere oplossingen bevat.

Raises

TypeError
Als het format van de oplossingenlijst niet gelijk is aan één van de volgende:

- een lege lijst
- een lijst met gehele getallen
- een lijst met hierin lijsten met gehele getallen

stikstof_puzzel.postprocess_solution.**postprocess_solutions**(*solution*, *notifications_list*)

Het aanroepen van het BIP-model levert één of meerdere oplossingen op. Deze functie wordt gebruikt om uit de oplossingen van het BIP-model de optimale oplossing te selecteren, door gebruik te maken van een aantal aanvullende criteria, naast de voorwaarden die in het BIP-model gesteld worden:

- Zijn er meerdere oplossingen mogelijk, dan wordt gekozen voor de oplossing waarbij het grootste aantal PAS-meldingen gelegaliseerd wordt.
- Zijn er meerdere oplossingen die een maximaal aantal PAS-meldingen legaliseren, dan wordt gekeken naar de positie van alle PAS-meldingen in de database, in oplopende volgorde. Zodra een oplossing een PAS-melding niet bevat die wel in een andere oplossing voorkomt, valt deze oplossing af. Omdat alle oplossingen uniek zijn, levert dit uiteindelijk een unieke optimale oplossing.

Parameters

solution
[list] De oplossing(en) die berekend zijn met behulp van het Binary-Integer-Program-model.

notifications_list
[list] Lijst met alle relevante PAS-meldingen-id's en de bijbehorende databaseposities. De PAS-meldingen-id's zijn afgeleid uit de bestandsnamen van de PAS-meldingen-GML's, de databasepositie is gebaseerd op het moment waarop de PAS-melding is ingediend bij RVO.

Returns

selected_notifications
[list] Lijst met de PAS-meldingen-id's die samen de optimale oplossing van de stikstofpuzzel vormen. Als alle PAS-meldingen uit deze lijst gelegaliseerd worden, dan wordt de beschikbare stikstofruimte optimaal benut en zal er zo weinig mogelijk stikstofrestruimte overblijven.

no_solutions_reasons
[pd.DataFrame] Bevat alle oplossingen die berekend zijn met het script binary_integer_program.py, maar die niet de optimale oplossing bleken te zijn. Bij elke oplossing is een reden toegevoegd waarom dit niet de optimale oplossing van de stikstofpuzzel is.

Raises

TypeError
Als het format van de lijst met PAS-meldingen (= parameter notifications_list) niet gelijk is aan een lijst met tuples (PAS-melding-id, databasepositie).

stikstof_puzzel.postprocess_solution.**write_solution**(*solution*, *no_solutions_reasons*, *export_dir*)

Deze functie wordt gebruikt om de oplossing en de reden voor het afvallen van eventuele overige oplossingen van het BIP-model weg te schrijven naar een tekstbestand.

Parameters

solution
[list] De optimale oplossing van de stikstofpuzzel. Elk element uit deze lijst verwijst naar een PAS-melding.

no_solutions_reasons
[pd.DataFrame or str] Bevat alle oplossingen die berekend zijn met het script binary_integer_program.py, maar die niet de optimale oplossing bleken te zijn. Bij elke oplossing is een reden toegevoegd waarom dit niet de optimale oplossing van de stikstofpuzzel is.

export_dir
[str] Het pad naar de map waar de uitvoer in weggeschreven moet worden.

Raises

TypeError
Als de oplossingenlijst elementen bevat die geen string zijn.

TypeError
Als de argumentatie van de afgevallen oplossingen niet het goede format heeft.

CHAPTER

THREE

TECHNICAL DOCUMENTATION: TESTS

3.1 conftest.py

In dit script wordt het pad gedefinieerd naar de map die de testdata bevat. Deze testdata is gebruikt om de code te testen die gebruikt wordt om de stikstofpuzzel op te lossen.

tests.conftest.**datadir**()

Het pad naar de map waar alle testdata opgeslagen staat

3.2 test_read_gml.py

Deze module bevat de toetsen die geschreven zijn om het functioneren van de inleesroutine te controleren. De inleesroutine moet GML's correct inlezen, en bij foutieve GML's of foutieve bestandsnamen van GML's indicatieve foutmeldingen geven.

De routines die getoetst worden, zijn opgenomen in de module [read_gml.py]

tests.test_read_gml.**test_no_config_but_correct_vars**(*monkeypatch*, *datadir*)

testen van routine als de env-file niet gegeven is, maar variabelen wel gezet zijn.

tests.test_read_gml.**test_no_config_no_vars**(*monkeypatch*)

testen of routine faalt als de env-vars niet gezet zijn.

tests.test_read_gml.**test_no_config_non_existing_data_dir**(*monkeypatch*)

testen of routine faalt als de env-vars foutief gezet zijn.

tests.test_read_gml.**test_no_config_wrong_threshold**(*monkeypatch*, *datadir*)

testen of routine faalt als de env-vars foutief gezet zijn.

tests.test_read_gml.**test_read_config**(*monkeypatch*, *datadir*)

testen of config goed gelezen wordt.

tests.test_read_gml.**test_read_double_ids**(*datadir*)

Test het inlezen van de bestanden vanuit de mappen 'meldingen' en 'SRV', met dubbele databaseposities in de bestandsnamen.

tests.test_read_gml.**test_read_double_notification_ids**(*datadir*)

Test het inlezen van de bestanden vanuit de mappen 'meldingen' en 'SRV', met dubbele PAS-meldingen-id's in de bestandsnamen.

tests.test_read_gml.**test_read_files**(*datadir*)

Test het inlezen van de bestanden vanuit de mappen 'meldingen' en 'SRV'.

tests.test_read_gml.**test_read_notification_data**(*datadir*)

Test de inhoud van de ingelezen PAS-melding-bestanden.

tests.test_read_gml.**test_read_old_files**(*datadir*)

Test het inlezen van de bestanden vanuit de mappen 'meldingen' en 'SRV', met oude bestandsnaam formatting.

tests.test_read_gml.**test_read_srv_data**(*datadir*)

Test de inhoud van het ingelezen Srv-bestand.

tests.test_read_gml.**test_read_weird_files**(*datadir*)

Test het inlezen van de bestanden vanuit de mappen 'meldingen' en 'SRV', met onjuiste bestandsnamen.

3.3 test_preprocess_input.py

Deze module bevat de toetsen die geschreven zijn om het functioneren van het voorbewerken van de invoerdata te controleren. De voorbewerkroutine moet de data die ingelezen is uit de GML's transformeren naar invoerdata die geschikt is om ingelezen te worden in het BIP-model. Bij foutieve invoer moeten indicatieve errormeldingen gegeven worden.

De routines die getoetst worden, zijn opgenomen in de module [preprocess_input.py]

tests.test_preprocess_input.**test_preprocess_input_1**()

Test of GML-data goed ingelezen wordt.

tests.test_preprocess_input.**test_preprocess_input_2**()

Test of GML-data goed ingelezen wordt.

tests.test_preprocess_input.**test_preprocess_input_empty_notification**()

Test wat de voorbewerkroutine doet als een PAS-melding geen hexagonen bevat waarin stikstofruimte gevraagd wordt.

tests.test_preprocess_input.**test_preprocess_input_empty_srv**()

Test wat de voorbewerkroutine doet als de Srv geen hexagonen bevat waarin stikstofruimte beschikbaar is.

tests.test_preprocess_input.**test_preprocess_input_hex_in_notification_not_in_srv**()

Test wat de voorbewerkroutine doet als een PAS-melding een hexagoon bevat die niet voorkomt in de Srv.

tests.test_preprocess_input.**test_preprocess_input_nan_values_notifications**()

Test de voorbewerkroutine als de gevraagde stikstofruimte een NaN-waarde bevat.

tests.test_preprocess_input.**test_preprocess_input_nan_values_srv**()

Test de voorbewerkroutine als de beschikbare stikstofruimte een NaN-waarde bevat.

tests.test_preprocess_input.**test_preprocess_input_negative_value_for_notification**()

Test de voorbewerkroutine als de gevraagde stikstofruimte negatief is.

tests.test_preprocess_input.**test_preprocess_input_negative_value_for_srv**()

Test de voorbewerkroutine als de beschikbare stikstofruimte negatief is.

tests.test_preprocess_input.**test_preprocess_input_unfeasible_notification**()

Test de voorbewerkroutine als er een PAS-melding is die niet binnen de beschikbare stikstofruimte past.

tests.test_preprocess_input.**test_preprocess_input_unused_hexagon**()

Test de voorbewerkroutine als de PAS-meldingen stikstofruimte vragen in een hexagoon waar geen stikstofruimte beschikbaar is, en er een hexagoon is waar wel stikstofruimte beschikbaar is, maar niet wordt gevraagd.

tests.test_preprocess_input.**test_preprocess_input_wrong_input_notification_list**()

Test de voorbewerkroutine als de PAS-meldingen gegeven zijn in een verkeerd format: de gevraagde stikstofruimte is gegeven als tekst.

tests.test_preprocess_input.**test_preprocess_input_wrong_input_notifications**()

Test de voorbewerkroutine als de PAS-meldingen gegeven zijn in een verkeerd format: de informatie is gegeven als tekst.

tests.test_preprocess_input.**test_preprocess_input_wrong_input_srv**()

Test de voorbewerkroutine als de beschikbare stikstofruimte gegeven is als tekst.

3.4 test_bip.py

Deze module test de functionaliteit van het BIP-model. Er zijn tests geschreven waarbij foutieve invoer wordt gegeven, en foutmeldingen worden verwacht. Ook zijn er tests geschreven die de uitkomst van het model toetst, wanneer correcte invoer gebruikt wordt.

De routines die getoetst worden, zijn opgenomen in de module [binary_integer_program.py].

tests.test_bip.**test_bip_no_solution**()

Test het BIP-model wanneer er geen oplossingen mogelijk zijn, doordat er niet genoeg stikstofruimte beschikbaar is in de hexagonen.

tests.test_bip.**test_bip_non_existing_hexagon_in_notification**()

Test het BIP-model wanneer niet alle hexagonen aangeboden worden.

tests.test_bip.**test_bip_notification_zero_space**()

Test de output van het model als er een PAS-melding wordt meegegeven die geen ruimte vraagt.

tests.test_bip.**test_bip_single_solution**()

Test aan de hand van PAS-meldingen en Srv-data of het BIP-model de twee verwachte optimale oplossingen teruggeeft.

tests.test_bip.**test_bip_with_high_threshold**()

Test de functionaliteit van het BIP-model waarbij alle mogelijke oplossingen (ongeacht stikstofrestruimte) worden teruggegeven.

tests.test_bip.**test_bip_with_some_threshold**()

Test aan de hand van PAS-meldingen en Srv-data of het BIP-model de twee verwachte oplossingen teruggeeft wanneer een andere threshold dan 0 gebruikt wordt.

tests.test_bip.**test_bip_wrong_input**()

Test de functie wanneer er verkeerde input ingevoerd wordt. Er moeten nette errormeldingen teruggegeven worden.

tests.test_bip.**test_goal_function_value**()

Test of de waarde van de doelfunctie overeenkomt met de verwachte waarde.

3.5 test_postprocess_solution.py

Deze module bevat de toetsen die geschreven zijn om het functioneren van de nabewerkingsroutine te controleren. De nabewerkingsroutine moet de optimale oplossing kiezen in het geval er door het BIP-model meerdere oplossingen berekend zijn. Bij foutieve invoer moeten indicatieve errormeldingen gegeven worden.

De routines die getoetst worden, zijn opgenomen in de module [postprocess_solution.py]

tests.test_postprocess_solution.**test_postprocess_solution_double_ids**()

Test wat er gebeurt als meerdere PAS-meldingen dezelfde databasepositie hebben

tests.test_postprocess_solution.**test_postprocess_solution_double_solution**()

Test of de functie goed omgaat met een dubbele oplossing in de lijst met oplossingen. Het model zal één van beide oplossingen terug moeten geven.

tests.test_postprocess_solution.**test_postprocess_solution_multi_diff_length**(*notifications_list*)

Test of de oplossing met de meeste PAS-meldingen geselecteerd wordt als optimaal als er meerder oplossingen zijn waarbij een verschillend aantal PAS-meldingen gelegaliseerd wordt.

tests.test_postprocess_solution.**test_postprocess_solution_no_solution**(*notifications_list*)

Test de functie in het geval er geen oplossing is

tests.test_postprocess_solution.**test_postprocess_solution_same_length**(*notifications_list*)

Test of de oplossing met de PAS-meldingen die als eerste in de database voorkomen geselecteerd wordt als er meerder oplossingen zijn waarbij een gelijk aantal PAS-meldingen gelegaliseerd wordt.

tests.test_postprocess_solution.**test_postprocess_solution_single**(*notifications_list*)

Test of de functie de gewenste antwoorden geeft als er één oplossing gevonden is door het BIP-model.

tests.test_postprocess_solution.**test_postprocess_solution_wrong_list**(*notifications_list*)

Test de functie in het geval er verkeerde invoer gebruikt wordt: de oplossing uit het BIP-model bevat een string in plaats van een integer

tests.test_postprocess_solution.**test_postprocess_solution_wrong_nested_list**(*notifications_list*)

Test de functie in het geval er verkeerde invoer gebruikt wordt: de oplossing uit het BIP-model bevat een driedubbel geneste lijst

tests.test_postprocess_solution.**test_postprocess_solution_wrong_no_databasepositions**(*notifications_list*)

Test de functie in het geval er verkeerde invoer gebruikt wordt: de lijst met PAS-meldingen bevat geen databaseposities

tests.test_postprocess_solution.**test_postprocess_solution_wrong_type**(*notifications_list*)

Test de functie in het geval er verkeerde invoer gebruikt wordt: de oplossing uit het BIP-model is een string in plaats van een lijst

tests.test_postprocess_solution.**test_postprocess_solution_wrong_type_databasepositions**(*notifications_list*)

Test de functie in het geval er verkeerde invoer gebruikt wordt: de databaseposities zijn geen gehele getallen

PYTHON MODULE INDEX

s

t