Improving Search applied to Running Dinner Problem

Justification of research and implemention

Josine Verkoeijen & Teun Voesenek, 9 oktober 2023

Inhoud

- Probleem definitie
- Onderzoek aanpak
- Ontwerp van heuristiek obv Improving Search
  - Gekozen Buurtruimte
  - De verbondenheid van de buurtruimte
- Discrete Improving Search 2-opt: Logica
- Python Implementation: Data
- Python Implementation: Functies eisen
- Python Implementation: Functies wensen
- Ordering van de code
- Testen en experimenten
- Discrete Improving Search: Start eerste oplossing 2023 VS Start tweede oplossing 2023 voor T = 3
- Discrete Improving Search: Oplossingswaarde per iteratie voor T = 3
- Conclusie van de 2-opt oplossing
- Gerealiseerde diepte in OR

Probleem Definitie

- Willemen, R. (2023). Runningdinerprobleem Vughterpoort 2023 – conceptueel model. Fontys Engineering. Geraadpleegd op, van Canvas

Onderzoek aanpak

- Literature
  - R. Rardin (2014), *Optimization in Operations Research*, Pearson New International Edition
  - E.H.L. Aarts and J.K. Lenstra (Editors) (2003), *Local Search in Combinatorial Optimization*, Princeton University Press

Ontwerp van heuristiek obv Improving Search

- Gekozen Buurtruimte
  - Move: 2-opt cf Aarts and Lenstra (2003:
    - Verwijder twee personen uit de planning binnen een van de drie gangen
    - Verwissel deze twee personen en voeg ze terug in de planning in de gang
- De verbondenheid van de buurtruimte
  - Binnen iedere gang volg op een volgende moves toe, zo kan iedere planning gecreëerd worden.

  Hiervoor is een vorm van Improving search gebruikt:
  - **Discrete Improving Search**
    - Start op een gegeven toegestane planning
    - 2-opt
    - Wanneer een beter planning gevonden wordt, gebruik deze als de nieuwe start oplossing
    - Gegarandeerd een lokaal optimum

Discrete Improving Search 2-opt: Logica

T = Hoeveel iteraties in een gang tot de volgende gang.

k = gangen {‘*voor’, ‘hoofd’, ’na’*}

i = j = alle adressen (*i* of *j)* waar een bewoner die op (*i* of *j*) een gerecht *k* kan eten.

Pseudo code:

Start Oplossing

Herhaal

Voor *k* in gangen

Herhaal voor *T* iteraties

Voor adres *i* van gang *k* van bewoner die op *i* eet

Voor adres *j* van gang *k* van bewoner die op *j* eet

Als *i* gelijk is aan *j*

Volgende *j*

Anders huidige oplossing is adres *i* verwisselt met adres *j* voor gang *k*

Als eisen huidige oplossing **niet meer** dan 0 zijn

Als wensen huidige oplossing **minder zijn dan** wensen start oplossing

Start oplossing is huidige oplossing

Tot iteratie gelijk is aan *T* dan volgende *k* (wanneer *k* laatste in gangen volgende is eerste in gangen)

Tot alle verwisselingen geprobeerd zijn en er geen beter oplossing gevonden is

#Huidige oplossing is oplosbaar en een lokaal optimum door gebruik van de 2-opt methode is gevonden.

Python Implementation: Data

- Data:
  - df = Startoplossing.
  - df_bewoners = Set van alle bewoners en of ze moeten koken.
  - df_adressen = Set van alle huisadressen, hoeveel bewoners op dit huisadres kunnen komen eten en de voorkeursgang.
  - df_kookte_2022 = Set van alle huisadressen die vorig jaar een gang kookt en de gang die ze kookte.
  - df_tagelgenoot _2022 = Set met koppels van bewoners die vorig jaar tafelgenoten waren.
  - df_buren = Set met koppels van bewoner en buur van deze bewoner.
  - df_paar_blijft_bij_elkaar = Set met koppels van bewoners die verplicht bij elkaar moeten blijven.
  - gt = Set met koppels van adres en de bijbehorende gang die dit adres moeten koken.
  - Ts = Set met koppels van bewoner + gang en adres.

Python Implementation: Functies eisen

- Functies eisen gedefinieerd:
  - elkeganganderadres(ts) = returns hoe vaak in de planning niet door iedere persoon 3 gangen op verschillende adressen gegeten wordt.
  - moetkoken(df,) = return hoe vaak in de planning mensen die moeten koken niet koken.
  - kookadresishuisadres(df) = return hoe vaak het kook adres niet gelijk is aan het huisadres.
  - countaanteletersvoldoed(df, df_adressen) = return hoe vaak het nummer of eters buiten de minimale en de maximale waarde valt.
  - paarbijelkaar(df, df_paar_blijft_bij_elkaar) = return hoe vaak een paar dat verplicht bij elkaar moet zitten niet bij elkaar zit.

  - eisen(ts, df, df_adressen, df_paar_blijft_bij_elkaar) = return de som van hoe vaak de eisen voorkomen.

Python Implementation: Functies wensen

- Functies wensen gedefinieerd:
  - meerdermalentafelgenoot(df) = return hoe vaak in de planning bewoners meerder malen elkaar tafelgenoten zijn.
  - hoofdgerecht2022(gt, df_kookte_2021) = return hoe vaak in de planning adressen die vorig jaar het hoofdgerecht gekookt hebben het hoofdgerecht weer moeten koken.
  - voorkeursgang(df, df_adressen) = return hoe vaak in de planning de bewoners die een voorkeur voor gang hebben opgegeven ook deze gang moeten koken.
  - zelfdetafelpartners2022(df, df_tafelgenoot_2022) = return hoe vaak in de planning de bewoners de zelfde tafelpartners als in 2022 hebben.
  - metdeburenaantafel(df, df_buren) = return hoe vaak bewoners met hun directe buren aan tafel zaten.
  - zelfdetafelpartners2021(fd, df_tafelgenoot_2021) = return hoe vaak in de planning de bewoners de zelfde tafelpartners als in 2021 hebben.

  - wensen(df, gt, df_kookte_2022, df_adressen, df_tafelgenoot_2022, df_buren, df_tafelgenoot_2021) = return de som van de wensen die per keer met een factor vermenigvuldigd zijn.

  - factor per wens: meerdermalentafelgenoot * 0,3 + hoofdgerecht2022 * 0,2 + voorkeursgang * 0,175 + zelfdetafelpartners2022 * 0,15 + metdeburenaantafel * 0,125 + zelfdetafelpartners2021 * 0,05

Ordering van de code

- Er is een file met de ingeladen data, wensen, eisen en de uitvoering van improving search in deze volgorde:
  - Functies van eisen:

```
#Functies eisen:
def elkeganganderadres(ts):#Functie die telt hoe vaak er niet door een persoon een voor, hoofd en nagerecht gegeten wordt en dat dit op een ander adres is.…
def moetkoken(df):#Functie die telt hoe vaak er een persoon niet kookt die wel moet koken.    """Functie die telt hoe vaak het kook adres niet gelijk is aan het huisadres."""…
def kookadresishuisadres(df):#Functie die telt hoe vaak het kook adres niet gelijk is aan het thuisaders.…
def countaantaletersvoldoed(df, df_adressen):#Functie die telt hoe vaak het gasten aantal waarvoor ze moeten koken buit het gasten aantal waarvoor ze kunnen koken ligt.…
def paarbijelkaar(df, df_paar_blijft_bij_elkaar):#Functie die telt hoe vaak een paar dat bij elkaar moet blijven niet bij elkaar is.…

def eisen(ts, df, df_adressen, df_paar_blijft_bij_elkaar):#Een Functie die alle eisen controleert.…
```

  - Functies van wensen:

```
#Functies wensen:
def meerdermalentafelgenoot(df):#Functie die telt hoe vaak er twee personen meer dat twee keer aan de zelfde tafel zitten.…
def hoofdgerecht2022(gt, df_kookte_2021): #Functie die telt hoe vaak een huis houden het hoofdgerecht vorige jaar en dit jaar moet koken.…
def voorkeursgang(df, df_adressen):#Functie die telt hoevaak een voorkeur gang juist is.…
def zelfdetafelpartners2022(df, df_tafelgenoot_2022):#Functie die telt hoevaak twee tafelgenoten in 2022 ook de tafelgenoot van 2023 waren.…
def metdeburenaantafel(df, df_buren):#Functie die telt hoevaak er met de driecte buren aan tafel gezeten wordt.…
def zelfdetafelpartners2021(df, df_tafelgenoot_2021):#Functie die telt hoevaak twee tafelgenoten in 2021 ook de tafelgenoot van 2023 waren.…

def wensen(df, gt, df_kookte_2022, df_adressen, df_tafelgenoot_2022, df_buren, df_tafelgenoot_2021):#Een Functie die alle wensen controleert.…
```

  - Data:

```
#Data inladen
df = pd.read_excel('Running Dinner tweede oplossing 2023 v2.xlsx')
df_bewoners = pd.read_excel("Running Dinner dataset 2023 v2.xlsx",sheet_name="Bewoners" )
df_adressen = pd.read_excel("Running Dinner dataset 2023 v2.xlsx",sheet_name="Adressen" )
df_kookte_2022 = pd.read_excel("Running Dinner dataset 2023 v2.xlsx",sheet_name="Kookte vorig jaar")
df_tafelgenoot_2022 = pd.read_excel("Running Dinner dataset 2023 v2.xlsx",sheet_name="Tafelgenoot vorig jaar")
df_buren = pd.read_excel("Running Dinner dataset 2023 v2.xlsx",sheet_name="Buren" )
df_paar_blijft_bij_elkaar = pd.read_excel("Running Dinner dataset 2023 v2.xlsx",sheet_name="Paar blijft bij elkaar" )
df_tafelgenoot_2021 = pd.read_excel("Running Dinner dataset 2022.xlsx",sheet_name="Tafelgenoot vorig jaar")
```

  - 2-opt:

```
#2 opt toegepast bij running dinner
moment0 = wensen(df, gt, df_kookte_2022, df_adressen, df_tafelgenoot_2022, df_buren, df_tafelgenoot_2021)
logger.debug(msg=f'Start:{moment0}')
class itteratiepergang(Exception): pass

def plot(X, Y):…

verwisselde_personen = []
itteratie = 0
improved = True
gangen = ['','','Voor','Hoofd','Na']
X = []
Y = []
X.append(0)
Y.append(moment0)
while improved:
    improved = False
    for k in range(2,5):
        loc_itteratie = 0
        try:
            for i in range(len(df)):
                for j in range(len(df)):
                    if i == j:
                        continue
                    else:
                        df_new = df.copy()
                        changes = []
                        change1 = df.iloc[i,k]
                        change2 = df.iloc[j,k]      #De verwisseling van de twee cellen
                        df_new.iloc[j,k] = change1
                        df_new.iloc[i,k] = change2

                        koppel = []
                        persoon1 = df.iloc[i,1]
                        persoon2 = df.iloc[j,1]   #Het maken van een tuple met het verwisselde koppel en de gang.
                        koppel.append(persoon1)
                        koppel.append(persoon2)
                        koppel.append(gangen[k])
                        tup = tuple(sorted(koppel))
                        if tup not in verwisselde_personen:  #Het zorgen dat er alleen gekeken wordt naar unique oplossing door de verwisselde bewoners en de gang in een lijste te stopp
                            verwisselde_personen.append(tup)

                            if eisen(ts, df_new, df_adressen, df_paar_blijft_bij_elkaar) > 0: #De controlle dat er geen eisen overschreden worden
                                continue

                            else:
                                sol = wensen(df_new, gt, df_kookte_2022, df_adressen, df_tafelgenoot_2022, df_buren, df_tafelgenoot_2021)
                                start = wensen(df, gt, df_kookte_2022, df_adressen, df_tafelgenoot_2022, df_buren, df_tafelgenoot_2021)
                                itteratie += 1
                                loc_itteratie += 1
                                logger.debug(msg=f'Itteratie:{itteratie}, local:{loc_itteratie}')

                                X.append(itteratie)
                                Y.append(sol)

                                if itteratie % 1000 == 0:
                                    df.to_excel('Oplossing 2 Running dinner 2023.xlsx', index = False)
                                    plot(X,Y)


                                if sol < start: #Wanneer de oplossing kleiner is dan de start wordt de oplossing de nieuwe start.
                                    df = df_new
                                    improved = True
                                    logger.debug(msg=f'Oplossing:{sol}')

                                if loc_itteratie == 3:
                                    raise itteratiepergang()


        except itteratiepergang:
            pass
```

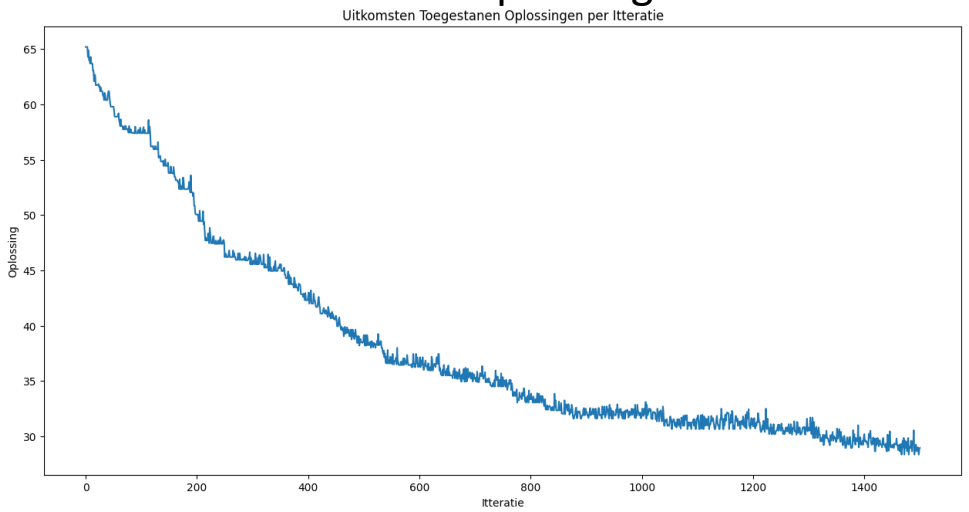
Testen en experimenten

- Eerst werd de methode 2-opt geïmplementeerd en kleinschalig getest.
- Deze code heeft op alle eisen en wensen een mogelijkheid op fouten:
  - Dubbel gecontroleerde de implementatie van alle eisen: **OK**
  - Dubbel gecontroleerde de implementatie van alle wensen: **OK**
  - Dubbel gecontroleerde de implementatie van 2-Opt: **NOK**
    - Ten eerste de gang *voor* geprobeerd maar na 7700 iteraties nog steeds geen lokaal optimum waardoor er nooit aan andere gangen begonnen zal worden, dus er zat wat fout.
    - Om dit probleem te verhelpen worden bij elke gang 3 iteraties uitgevoerd waardoor de optimalisatie over de gangen wordt verdeeld.
- Code uitbreidingen:
  - Methode voor visualisaties geïmplementeerd om de iteratie tegen de oplossing te visualiseren.
  - Logboek toegevoegd om bij te houden welke iteraties er zijn gedaan en welke oplossing er op dat moment de beste is.
- Tests:
  - 1500 iteraties 2-opt uitgevoerd op start oplossing een en twee om de beste startoplossing te vinden (zie slide 12). Er is gekozen om door te rekenen op oplossing één.
  - 2-Opt toegepast op startoplossing één, voor 8000 iteraties om een optimum te vinden. Na 6950 iteraties is het lokaal optimum gevonden (zie slide 13).

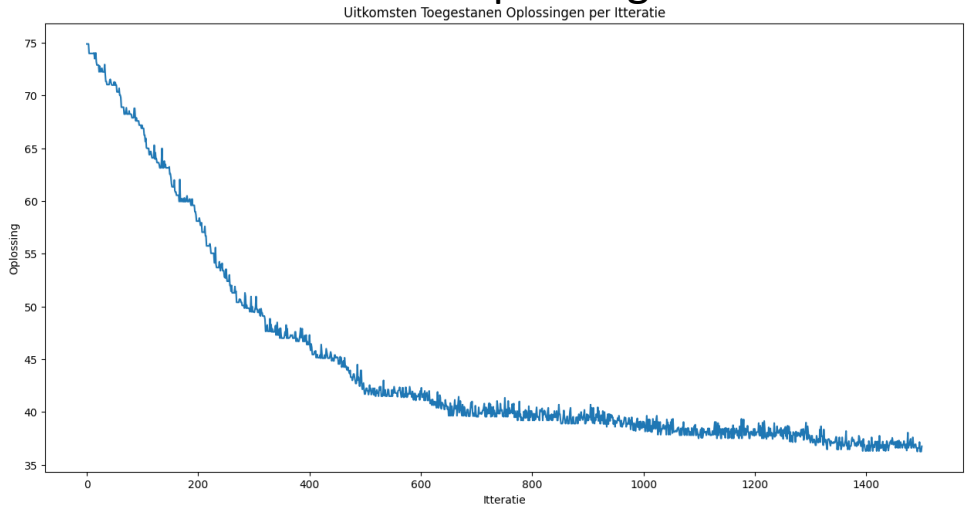
Discrete Improving Search: Start eerste oplossing 2023 VS Start tweede oplossing 2023 voor T = 3

Na 1500 iteraties is de eerste start oplossing van 2023 op een betere oplossing gekomen dan de tweede startoplossing hierdoor is er gekozen om verder te testen op de eerste start oplossing 2023,

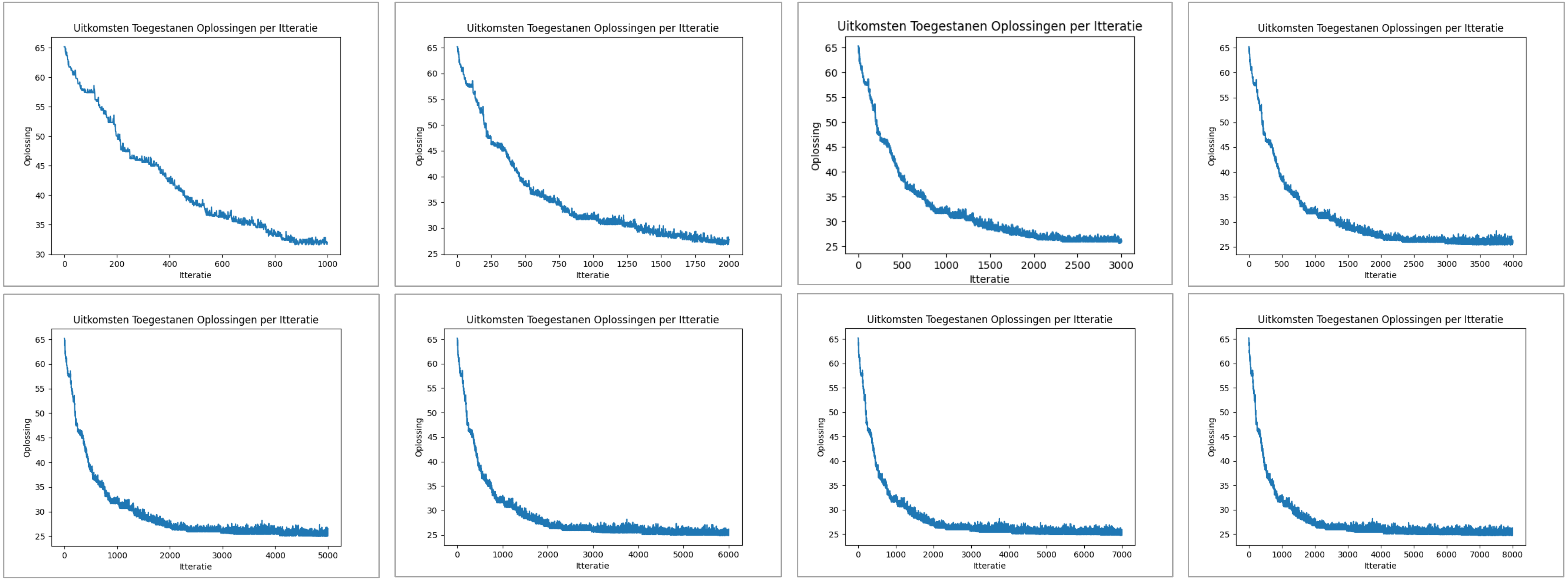
Start eerste oplossing 2023



Start tweede oplossing 2023



Discrete Improving Search: Oplossingswaarde per iteratie voor T = 3



Iteratie	1000	2000	3000	4000	5000	6000	7000	8000
Beste Oplossing	31.60	26.75	25.60	25.40	25.00	24.90	24.75	24.75
Tot rekentijd	1:11:36	2:09:12	3:21:06	4:24:36	5:31:21	6:42:55	8:03:25	9:42:44

Conclusie van de 2-opt oplossing

Planning	Waarde Oplossing	meerdermalentafelgenoot	hoofdgerecht2022	voorkeursgang	zelfdetafelpartners2022	metdeburenaantafel	zelfdetafelpartners2021
Start	65.2	82	0	0	234	0	110
Finish	24.75	14	0	0	118	0	57

Zo als te zien in de tabel is er vooruitgang geboekt op de punten meerdermalentafelgenoot, zelfdetafelpartners2022 en zelfdetafelpartners2021.

Gerealiseerde diepte in OR

- Optimalisatie doormiddel van Python.
- Vergelijken start oplossing één en twee.