Problem rotacije useva

Nikola Šutić

12.5.2024

Uvod

- Koncept rotacije useva
- Potreba za rotacijom useva
- Problem sa ručnim planiranjem

Principi

- Rotacija useva: principi, benefiti, izazovi
- Genetski algoritam: koncept, postuptak i prilagođenje problemu useva

Postavka problema

- Optimizacija rotacije useva
- Faktori koji utiču: nutrienti zemljišta, upravljanje bolestima i napastima, klima, prinos itd.

Metodologija

- Opis algoritma - inicijalizacija, ukrštanje, odabir roditelja, fitnes jedinke, mutacija
- Prilagođavanje problema algoritmu
- Ograničenja

Implementacija i delovi koda

```
{
    "time_units": 24,
    "start_month": 4,

  "plot_adjacency": [
    [1,2],
    [0,3],
    [0,3],
    [1,2]
  ],

  "plot_areas": [
    0.25,0.25,0.25,0.25
  ],

    "crops": [
  {
    "name": "paradajz",
    "family": "Solanaceae",
    "planting_month": [1, 2, 4, 5],
    "grow_time": 60,
    "plants_per_hectare": 30000,
    "yield": 12.5
  },
  {
    "name": "rani paradajz",
    "family": "Solanaceae",
    "planting_month": [4],
    "grow_time": 60,
    "plants_per_hectare": 30000,
    "yield": 12.5
  },
  {
    "name": "kasni paradajz",
    "family": "Solanaceae",
    "planting_month": [7],
```

Slika: Specifikacija konkretne instance problema CR

Implementacija i delovi koda

```
 9 class CRSPSolution:
10
11     def __init__(self, N, M, K, T, I, B, PPP, mutation_streng=
   th = 0.2):
12
13         self.N = N # Number of crops
14         self.M = M # Number of periods
15         self.K = K # Number of fields
16         self.T = T # Vegetation period of crops (in periods)
17         self.I = I # Crops planting periods
18         self.B = B # Crop families seperated
19         self.PPP = PPP # Plants per plot
20         self.learning_rate = 0.2
21         self.mutation_strength = mutation_strength
22
23         self.plan = np.zeros((
24             self.N,
25             self.M,
26             self.K
27         ))
```

Slika: Implementacija jedinke

Implementacija i delovi koda

```
29      def generate_plot(self, k):
30
31          plot = np.zeros((self.N, self.M))
32          B_order = list(range(len(self.B)))
33          rd.shuffle(B_order)
34          t = 0
35          b = 0
36
37          last_planted_family = None
38
39          while t < self.M:
40
41              time_left = self.M - t - 1
42
43              for b in B_order:
44                  for c in self.B[b]:
45                      if t in self.I[c] \
46                          and time_left >= self.T[c] \
47                          and rd.randint(0,1) > 0 \
48                          and self.PPP[c][k] > 0:
49                          plot[c][t] = 1
50                          t += self.T[c]
51                          time_left -= t
52                          last_planted_family = b
53                          break
54
55              t += 1
56
57          return plot
58
```

Slika: Generisanje mogućeg poretka za jedno parče zemljišta

Implementacija i delovi koda

```
104     def mutation(self):
105
106         for k in range(self.K):
107             if(np.random.random() < self.mutation_strength):
108                 self.plan[:,:,k] = self.generate_plot(k)
109
110         #for i in range(self.N):
111         #    for j in self.I[i]:
112         #        for k in range(self.K):
113         #            if(np.random.random() < self.mutation_st
    rength):
114         #                self.plan[i][j][k] = not self.plan[i
    ][j][k]
115
116         self.mutation_strength *= np.exp(self.learning_rate*
     np.random.randn())
117         return self
```

Slika: Mutacijom uvodimo novi redosled za jedno parče zemlje

Implementacija i delovi koda

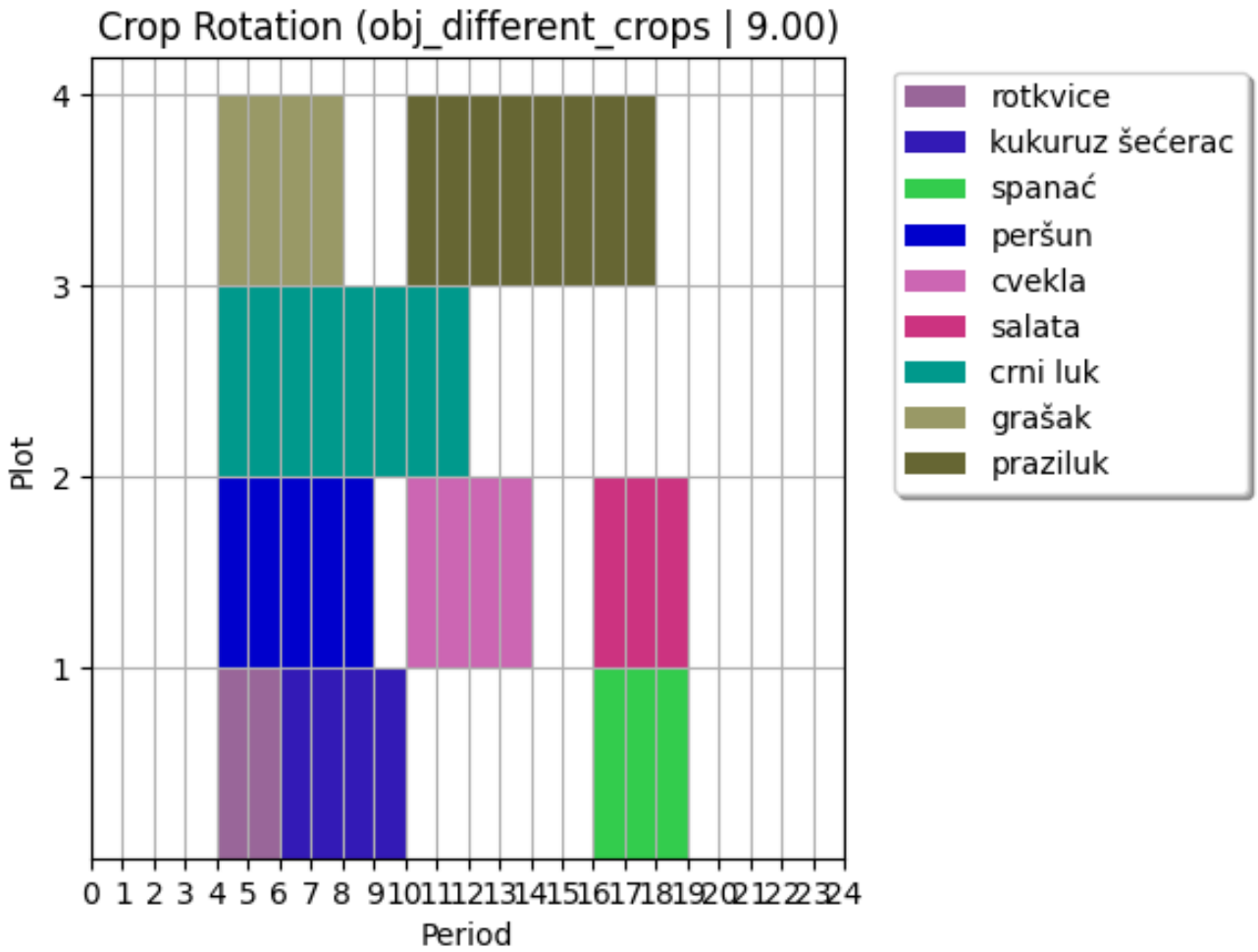
```
def fitness(self, sol):

    _fitness = self.objective(sol) +\
               self.constraint2(sol) +\
               self.constraint3(sol) +\
               self.constraint4(sol) +\
               self.constraint5(sol) +\
               self.constraint6(sol) +\
               self.constraint7(sol) +\
               self.constraint8(sol)
```

Implementacija i delovi koda

```
332     def constraint6(self,sol):
333         # Same botanical families should not be planted in ad
    jacent fields
334
335         penalty = 0
336
337         for k in range(self.K):
338             for f in self.B:
339                 for c1 in f:
340                     q = self.T[c1]
341                     for c1i in self.I[c1]:
342                         _sum = 0
343                         for c2 in f:
344                             for k_adj in self.plot_adjacency[
    k]:
345                                 _sum += sol.plan[c1,c1i:c1i+q
    ,k].sum() + sol.plan[c2,c1i:c1i+q,k_adj].sum()
346
347                         penalty += self.soft_penalty * _sum i
    f _sum > 1 else 0
348
349         return penalty
350
```

Slika: Primer jednog ograničenja (Iste porodice biljaka ne smeju biti jedna pored druge)

Rezultat i diskusija

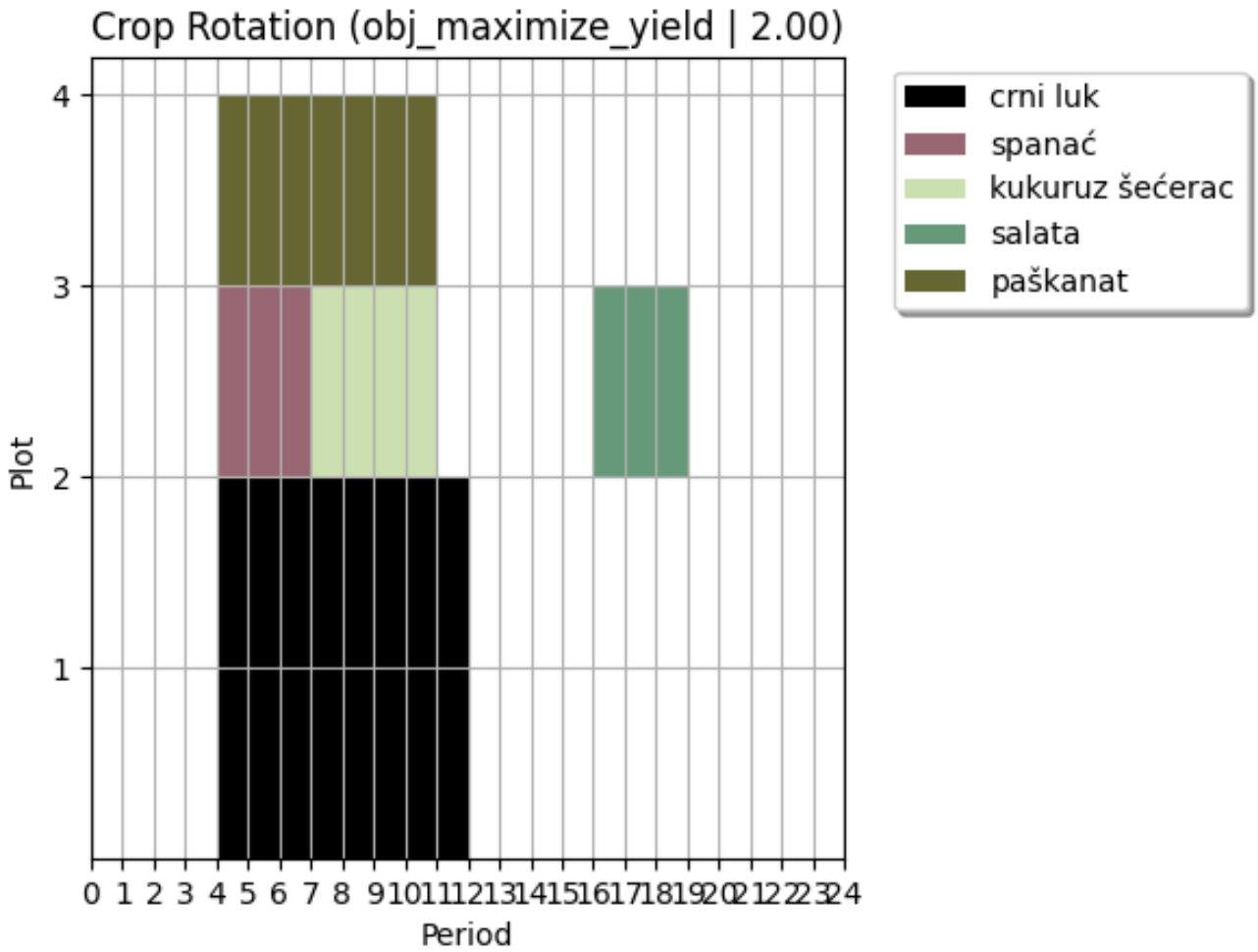


Rezultat i diskusija

Rezultat i diskusija

Rezultat i diskusija

Rezultat i diskusija



Ograničenja i dalji rad

- **Dostupnost podataka u poljoprivredi**
- Veliki broj matematičkih ograničenja
- Učenje potkrepljivanjem

Zaključak

- Problem, rešenje, rezultat
- Primena naprednih optimizacionih algoritama u polju poljoprivrede