Univerza v Ljubljani
Finančna matematika 1. stopnja

# Consecutive square packing

Finančni praktikum

Patrik Gregorič, Petja Murnik

Ljubljana, 2022

# Kazalo



# Slike

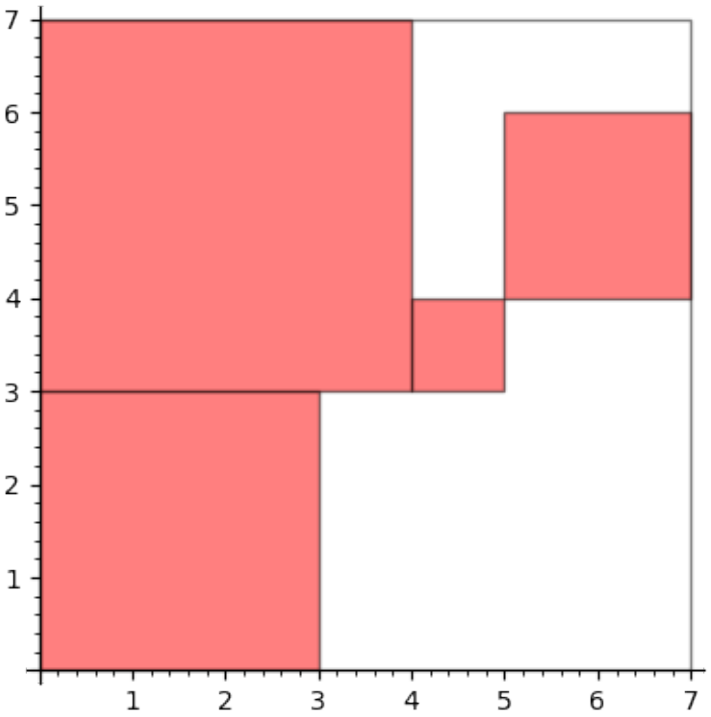
# 1 Opis problema

## 1.1 Osnovni problem

Za vsak celo število $i = 1, \ldots, n$ imamo en kvadrat s stranicami dolžine $i$. Želimo najti kvadrat z najkrajšo stranico, v katerega lahko zložimo vse kvadrate, ne da bi se notranjosti kvadratov prikrivale (robovi se lahko dotikajo). Kvadratov ne moremo obračati lahko jih samo transliramo.

## 1.2 Ideja za reševanje

Problem je enostavno predstavljiv. Imamo zaporedje vse večjih kvadratov, ki jih razporedimo v ravnino, te se ne prekrivajo, se lahko dotikajo. Kvadratov ne smemo rotirati, lahko jih le premikamo levo, desno, gor, dol. Omejila sva problem le na pozitivne $x$ in $y$ zaradi lažje predstave. Za majhen $n$ si je problem lahko predstavljati, saj ga rešimo zelo hitro kar sami.




Slika 1: Lahek primer za $n = 4$

Opazimo, da postavimo največja kvadrata drug zraven drugega in ostale le razporedimo s kar veliko možnostmi na prosta mesta. Problem je enostaven, ker je veliko različnih možnosti.

# 2 Reševanje osnovnega problema

## 2.1 Kratek opis

Za večji $n$ postane problem mnogo težji. Zato sva se odločila reševati s celoštevilskim linearnim programiranjem. To je: uvedla bova spremenljivke, zapisala omejitve in to poslala skozi vgrajene metode za reševanje CLP v programskem jeziku Sage.

## 2.2 Zapis problema kot CLP

Za podatke imamo seznam kvadratov. Vsak kvadrat $i$ v seznamu *kvadrati* je sestavljen iz trojice $(x-koordinata, y-koordinata, dolzina\_daljice)$, kjer imamo spremenljivko $dolzina\_daljice$ poznano, spremenljivki $x-koordinata, y-koordinata$ pa ne.

$$
\begin{array}{lll}
\text{minimize} & z & \\
\text{p.p.:} & i[0] \geq 0 & , \quad \text{za } i \text{ v } kvadrati \\
 & i[1] \geq 0 & , \quad \text{za } i \text{ v } kvadrati \\
 & i[0]i[2] \leq z & , \quad \text{za } i \text{ v } kvadrati \\
 & i[1]i[2] \leq z & , \quad \text{za } i \text{ v } kvadrati \\
 & i[0] + i[2] \leq j[0] + M * a[i,j] & , \quad \text{za } i \neq j \text{ v } kvadrati \\
 & j[0] + j[2] \leq i[0] + M * a[j,i] & , \quad \text{za } i \neq j \text{ v } kvadrati \\
 & i[1] + i[2] \leq j[1] + M * b[i,j] & , \quad \text{za } i \neq j \text{ v } kvadrati \\
 & j[1] + j[2] \leq i[1] + M * b[j,i] & , \quad \text{za } i \neq j \text{ v } kvadrati \\
 & a[i,j] + a[j,i] + b[i,j] + b[j,i] \leq 3, & \quad \text{za } i \neq j \text{ v } kvadrati \\
 & a[i,j] \in \{0,1\} & , \quad \text{za } i \neq j \text{ v } kvadrati \\
 & b[i,j] \in \{0,1\} & , \quad \text{za } i \neq j \text{ v } kvadrati
\end{array}
$$

Pripomnimo nekaj stvari za pojasnilo:

1. $i[0]$ je $x - koordinata$ ,$i[1]$ je $y - koordinata$ in $i[2]$ je $dolzina\_daljice$ od kvadrata $i$
2. $a[i,j]$ je spremenljivka, ki pokaže 1, če kvadrat $i$ NI levo od $j$ in 0 sicer
3. $b[i,j]$ je spremenljivka, ki pokaže 1, če kvadrat $i$ NI spodaj od $j$ in 0 sicer
4. $M$ je zgornja meja, s katero si pomagamo, da je le nekaj pogojev res. Za njo lahko vzamemo $\sum_{i \text{ v } kvadrati} i[2]$

## 2.3 Potek reševanja

Za podatke imava podane kvadrate s stranicami dolžine $i$, kjer $i = 1, \ldots, n$, in $i \in \mathbb{Z}, n \in \mathbb{N}$. Ciljna funkcija najinega problema je iskanje $\min z$, kjer je $z$

dolžina stranice največjega kvadrata, v katerega se zloži vse manjše kvadrate s stranicami $i = 1, \ldots, n$.
Za reševanje problema sva definirala razred *Kvadrat*.

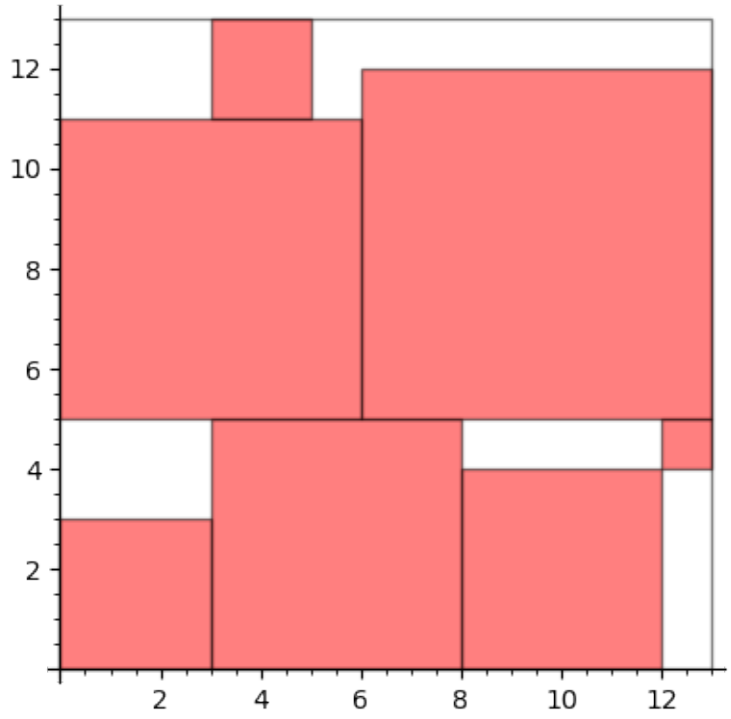
```
from sage.plot.polygon import polygon
class Kvadrat:

    def __init__(self, x, y, dolzina_stranice):
        self.x = x
        self.y = y
        self.dolzina_stranice = dolzina_stranice

    def narisi(self, barva="red"):
        return polygon(
            [(self.x, self.y), (self.x + self.dolzina_stranice, self.y),
             (self.x + self.dolzina_stranice, self.y + self.dolzina_stranice),
             (self.x , self.y + self.dolzina_stranice )],
            color=barva,
            edgecolor='black',
            alpha=0.5,
            zorder=2)

def narisi_rezultat_P(rezultat):
    k = Kvadrat(0,0,rezultat[0])
    rezultat1 = list(rezultat[1])
    K = k.narisi()
    pomoc = []
    for i in rezultat1:
        j = Kvadrat(i[0],i[1],i[2])
        pomoc.append(j)
    RISI = [K]
    for l in pomoc:
        m = l.narisi()
        RISI.append(m)
    show(sum(RISI[i] for i in range(len(RISI))))
```

Tu sva definirala kvadrat kot levo krajišče $(x, y)$ in dolžino stranice. Drugi dve funkciji `narisi` in `narisi_rezultat_P` nam omogočata prikaz rezultatov, ki jih dobiva v obliki seznama.

```
def packing(kvadrati): #kvadrati je seznam cifer
    p = MixedIntegerLinearProgram(maximization=False)
    x = p.new_variable()
    y = p.new_variable()
    zg_meja = sum(kvadrati) + 1
```

```
    a = p.new_variable(binary = True) #kvadrat i ni levo od j
    b = p.new_variable(binary = True) #kvadrat j ni desno od j
    p.set_objective(p['z'])
    p.add_constraint((p['z']) >= ((max(kvadrati))+kvadrati[len(kvadrati)-2]))
    for i, d in enumerate(kvadrati): # i je indeks, d je dolžina stranice
        p.add_constraint(x[i] >= 0)  # želimo nenegativne koordinate
        p.add_constraint(y[i] >= 0)
        p.add_constraint(x[i] + d <= p['z']) # omejimo p['z'] z največjo pokrito koor
        p.add_constraint(y[i] + d <= p['z'])
        for j , g in enumerate(kvadrati):
            p.add_constraint(x[i] + d <= x[j] + zg_meja * a[i,j])
            p.add_constraint(x[j] + g <= x[i] + zg_meja * a[j,i])
            p.add_constraint(y[i] + d <= y[j] + zg_meja * b[i,j])
            p.add_constraint(y[j] + g <= y[i] + zg_meja * b[j,i])
    for i, j in Combinations(range(len(kvadrati)), 2):
        p.add_constraint(a[i, j] + a[j, i] + b[i, j] + b[j, i] <= 3)
    z = p.solve()
    xx = p.get_values(x)
    yy = p.get_values(y)
    return (z, [(xx[i], yy[i], d) for i, d in enumerate(kvadrati)])
```

Ta CLP nam reši problem tudi za večji $n$, do katere rešitve ne pridemo tako intuitivno. Funkcija sprejme seznam cifer v obliki $range(1, n+1)$ in vrne *tuple*, ki vsebuje za prvi element dolžino največjega kvadrata, torej rešitev problema, naslednji element je pa seznam kvadratov, ki so podani od največjega do najmanjšega. Program najprej preveri, da so vse koordinate nenegativne, saj sva tako definirala problem. Nato preverimo pogoje, da morajo biti vsi kvadrati znotraj največjega, ki ga želimo minimizirat in še, da se noben od njih ne sme prekrivat. S pomočjo teh omejitev nam program poda pravilno rešitev, ki jo nato še prikaževa s funkcijo `narisi_rezultat_P`.




Slika 2: Primer za $n = 7$

Opazimo, da problem hitro postane bolj kompleksen za reševanje.

# 3 Nadgradnja problema

## 3.1 Opis nadgradnje

V nadeljevanju bova problem tudi reševala v primeru, če namesto enega kvadrata z dolžino stranice vzamemo dva/tri/štiri ... take kvadrate. Obravnavala bova tudi primer, ko je lahko ena točka v ravnini (velikem kvadratu) pokrita z dvema/tremi/štirimi ... kvadrati.

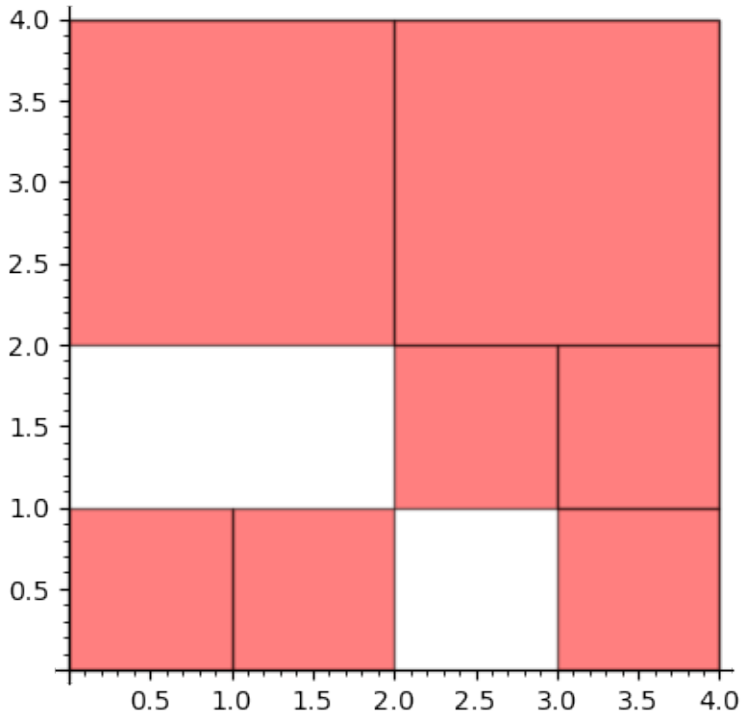
## 3.2 Zapis problema kot CLP, ko imamo kvadrate ne nujno zaporednih velikost

CLP v tem problemu izgleda identičen tistemu v osnovnem primeru 2.2

## 3.3 Potek reševanja, ko imamo kvadrate ne nujno zaporednih velikost

Dani podatki, ki jih sedaj ponudimo je seznam dolžin daljic kvadratov. V tem primeru ni pomembno, koliko je katerih dolžin, važno je le, da je seznam

urejen. To je tako, da so dolžine stranic naraščujoče. V tem primeru, nam funkcija `packing()` kot pri osnovnem primeru v 2.3, na danem seznamu deluje pravilno in nam najde rešitev. Nisva se ubadala s pisanjem funkcije, ki bi generirala seznam, ki bi imel števila od 1 do $n$ in vsako $l$-krat. To lahko bralec naredi sam brez večjih težav in preveri, da stvar deluje.




Slika 3: Primer za pet kvadratov dolžine 1 in dva dolžine 2

### 3.4 Zapis problema kot CLP, ko je vsaka točka v prostoru lahko pokrita $k$-krat

Za podatke imamo seznam kvadratov in število $k$. Vsak kvadrat $i$ v seznamu *kvadrati* je sestavljen iz trojice $(x{-}koordinata, y{-}koordinata, dolzina\_daljice)$, kjer imamo spremenljivko $dolzina\_daljice$ poznano, spremenljivki $x{-}koordinata, y{-}koordinata$ pa ne. Število $k$ nam pove, kolikokrat je lahko ena točka v prostoru pokrita, tj. v koliko kvadratih je lahko.

$$
\begin{array}{lll}
\text{minimize} & z & \\
\text{p.p.:} & i[0] \geq 0 & , \quad \text{za } i \text{ v } kvadrati \\
& i[1] \geq 0 & , \quad \text{za } i \text{ v } kvadrati \\
& i[0] + i[2] \leq z & , \quad \text{za } i \text{ v } kvadrati \\
& i[1] + i[2] \leq z & , \quad \text{za } i \text{ v } kvadrati \\
& i[0] + i[2] \leq j[0] + M * a[i,j] & , \quad \text{za } i \neq j \text{ v } kvadrati \\
& j[0] + j[2] \leq i[0] + M * a[j,i] & , \quad \text{za } i \neq j \text{ v } kvadrati \\
& i[1] + i[2] \leq j[1] + M * b[i,j] & , \quad \text{za } i \neq j \text{ v } kvadrati \\
& j[1] + j[2] \leq i[1] + M * b[j,i] & , \quad \text{za } i \neq j \text{ v } kvadrati \\
& a[i,j] + a[j,i] + b[i,j] + b[j,i] \leq 3 + L[i,j], & \quad \text{za } i \neq j \text{ v } kvadrati \\
& \sum_{\substack{g \in 1,\ldots,k+1 \\ f \geq g, g \in 1,\ldots k+1}} L[i_g, i_f] \leq \binom{k+1}{2} - 1 & , \quad \text{za } i_1 \neq i_2 \neq \ldots \neq i_{k+1} \text{ v } kvadrati \\
& a[i,j] \in \{0,1\} & , \quad \text{za } i \neq j \text{ v } kvadrati \\
& b[i,j] \in \{0,1\} & , \quad \text{za } i \neq j \text{ v } kvadrati \\
& L[i,j] \in \{0,1\} & , \quad \text{za } i \neq j \text{ v } kvadrati
\end{array}
$$

Večino pripomb, ki so tu potrebne, smo že pojasnili pri osnovnem primeru 2.2. Dodati je treba samo še eno:

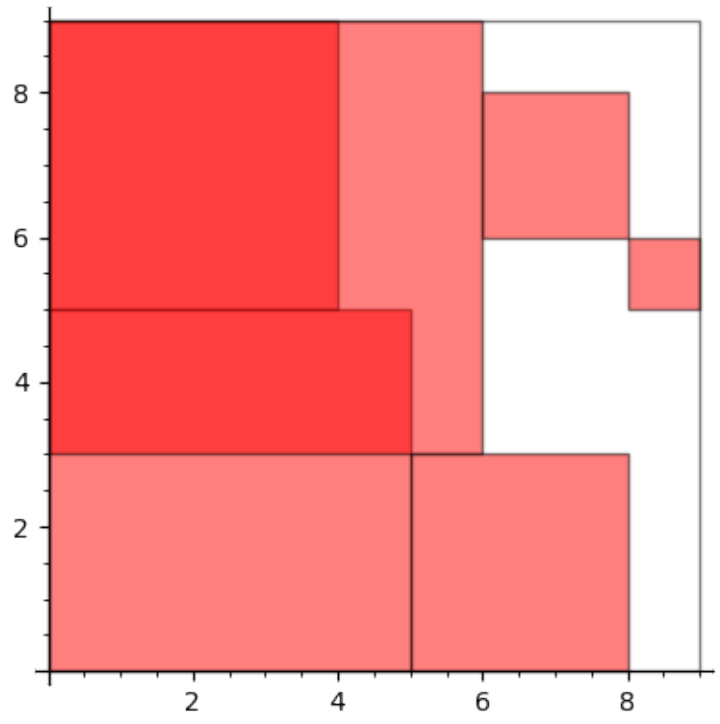
1. Če $L[i,j] = 1$, pomeni da se kvadrata i in j sekata. Pogoj z vsoto pa nam zagotovi da se nam ne seka preveč kvadratov. Če izberemo $k+1$ kvadrat in je vsota enaka $\binom{k+1}{2}$, tedaj se vsaka dva kvadrata iz izbranih $k+1$ kvadratov sekata. Torej obstaja točka ki je v vseh teh kvadratih in torej ne zadošča pogoju.

## 3.5 Potek reševanja, ko je vsaka točka v prostoru lahko pokrita $k$-krat

Podatki, ki jih v tem primeru dobimo je število $k$, ki pove kolikokrat je lahko ena točka v prostoru pokrita in urejen seznam dolžin stranic kvadratov. Za ta primer sva definirala naslednjo funkcijo, ki izračuna željen razultat:

```
def packing_ktimes(kvadrati,k): #kvadrati je seznam cifer, tale dela če so cifre
p = MixedIntegerLinearProgram(maximization=False,solver='GLPK')
x = p.new_variable()
y = p.new_variable()
zg_meja = sum(kvadrati) + 1
a = p.new_variable(binary = True) #kvadrat i ni levo od j
b = p.new_variable(binary = True) #kvadrat j ni desno od j
L = p.new_variable(binary = True)
p.set_objective(p['z'])
p.add_constraint((p['z']) >= (max(kvadrati)))
```

```
for i, d in enumerate(kvadrati): # i je indeks, d je dolžina stranice
    p.add_constraint(x[i] >= 0)  # želimo nenegativne koordinate
    p.add_constraint(y[i] >= 0)
    p.add_constraint(x[i] + d <= p['z']) # omejimo p['z'] z največjo pokrito koor
    p.add_constraint(y[i] + d <= p['z'])
    for j , g in enumerate(kvadrati):
        p.add_constraint(x[i] + d <= x[j] + zg_meja * a[i,j])
        p.add_constraint(x[j] + g <= x[i] + zg_meja * a[j,i])
        p.add_constraint(y[i] + d <= y[j] + zg_meja * b[i,j])
        p.add_constraint(y[j] + g <= y[i] + zg_meja * b[j,i])
for i, j in Combinations(range(len(kvadrati)), 2):
    p.add_constraint(a[i, j] + a[j, i] + b[i, j] + b[j, i] <= 3 + L[i,j])
for C in Combinations(range(len(kvadrati)), (k+1)):
    p.add_constraint(sum(sum(L[C[i],C[j]] for j in range(i+1,k+1)) for i in range
z = p.solve()
xx = p.get_values(x)
yy = p.get_values(y)
return (z, [(xx[i], yy[i], d) for i, d in enumerate(kvadrati)])
```




Slika 4: Primer, ko je vsaka točka lahko pokrita $2-krat$

Tak problem program reši bistveno hitreje, saj ima več možnosti za ureditev

kvadratov. Večkrat, ko je vsaka točka lahko pokrita enostavnejši postane izračun rezultata, saj si lahko predstavljamo problem, da vsak kvadrat razdelimo na svoj podproblem in ga rešujemo kot osnovni problem z malo več svobode pri omejevanju mej.

# 4 Sklep

Problem je težek, saj ne obstaja enolična rešitev. Program ima na voljo veliko možnosti za rešitve, hkrati pa mora biti problem dobro zastavljen, da program dobi pravilno rešitev. Za izračun si moramo postaviti tesne meje, ki omejujejo dopustne rešitve za hitrost delovanja programa.
V nadgradnji problema opazimo veliko hitrejše delovanje programa. Zahteve niso tako omejujoče s prekrivanjem ali pa z velikostjo kvadratov. Pri večkratnem prekrivanju se problem za program zelo poenostavi, kar močno vpliva na hitrost.
Pri zlaganju kvadratov v kvadrat opazimo veliko nepokritih točk, kar nam poda idejo, da bi bilo bolj učinkovito zlagati kvadrate v pravokotnik.