Slijedna shema za generiranje rasporeda i prioritetna pravila za rješavanje problema raspoređivanja s ograničenim sredstvima

Hančić, Anita

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, School of Applied Mathematics and Informatics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet primijenjene matematike i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:126:390509

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Slijedna shema za generiranje rasporeda i prioritetna pravila za rješavanje problema raspoređivanja s ograničenim sredstvima

Završni rad

Mentor:

doc. dr. sc. Mateja Đumić

Kandidat:

Anita Hančić

Komentor:

dr. sc. Rebeka Čorić

Osijek, 2024

Sadržaj

1 | Uvod

Problem raspoređivanja s ograničenim sredstvima (engl. *Resource constrained project scheduling problem*, skraćeno RCPSP) je problem koji se sastoji od sredstava s ograničenom dostupnosti i aktivnosti s poznatim trajanjem i zahtjevima na dana sredstva. Uveo ga je James E. Kelley Jr. 1963. godine, no problem je i danas vrlo popularan i prisutan u stvarnom svijetu te se smatra standardnim problemom kada govorimo o projektima s ograničenim sredstvima.

Jedna od najčešćih metoda rješavanja ovog problema su prioritetna pravila koja se u kombinaciji sa shemom za generiranje rasporeda koriste za izradu kompletnog rasporeda. U ovom radu obradit ćemo slijednu shemu za generiranje rasporeda.

Cilj rješavanja problema raspoređivanja s ograničenim sredstvima je rasporediti sve aktivnosti u projektu tako da zadovolje dana ograničenja te da je ukupno vrijeme trajanja projekta minimizirano.

Drugo poglavlje ovog rada uvodi i kratko opisuje problem raspoređivanja s ograničenim sredstvima. U trećem poglavlju su dane metode rješavanja danog problema s naglaskom na prioritetna pravila te je prikazana implementacija prioritetnih pravila. Četvrto poglavlje upoznaje nas s dvije sheme za generiranje rasporeda koje se koriste pri rješavanju problema te daje detaljniji opis, algoritam i implementaciju za slijednu shemu koja se obrađuje u ovom radu. U petom poglavlju na malom primjeru problema raspoređivanja s ograničenim sredstvima prikazano je kako korištenjem prioritetnih pravila i slijedne sheme dolazimo do rasporeda. Šesto poglavlje pokazuje koji su podaci korišteni u praktičnom dijelu projekta i kako iz njih iščitavamo one koji su potrebni za rješavanje problema. U sedmom poglavlju daje se kratak opis biblioteke korištene za razvoj korisničkog sučelja, kako u korisničkom sučelju generiramo raspored, prikaz primjera generiranih rasporeda i usporedba završnih vremena za svako prioritetno pravilo.

2 | Problem raspoređivanja s ograničenim sredstvima

Problem raspoređivanja s ograničenim sredstvima sastoji se od skupa aktivnosti $A = \{A_0, \ldots, A_{n+1}\}$ gdje A_0 i A_{n+1} označavaju početak, odnosno kraj projekta i nazivamo ih fiktivne aktivnosti (engl. *dummy activities*). Sve aktivnosti moraju zadovoljavati redoslijed izvršavanja dan uvjetima prednosti (engl. *precedence constraints*) koji prisiljavaju da aktivnost A_j ne započne prije nego što se završe sve njezine neposredne prethodne aktivnosti dane skupom P_j. [3]

Izvođenje aktivnosti, u ovom radu promatranih problema, zahtijeva sredstva koja su obnovljiva, tj. njihova količina je u svim vremenskim periodima jednaka. Imamo K vrsta sredstava koji su dani skupom $\mathcal{K} = \{1, \ldots, K\}$. Da bi bila obrađena, aktivnost A_j zahtijeva $r_{j,k}$ jedinica sredstva $k \in \mathcal{K}$ tijekom cijelog razdoblja njenog trajanja p_j. Za fiktivne aktivnosti, potrebna količina pojedinog sredstva, kao i vrijeme izvršenja, jednaka je 0, tj. $r_{0,k} = r_{n+1,k} = 0$, $\forall k \in \mathcal{K}$ i $p_0 = p_{n+1} = 0$.

Cilj problema raspoređivanja s ograničenim sredstvima je pronaći dopustivi raspored s minimalnim ukupnim trajanjem tako da su uvjeti prednosti i ograničenja na sredstva zadovoljeni.

3 | Metode rješavanja problema

Metode rješavanja problema raspoređivanja s ograničenim sredstvima dijelimo na egzaktne metode i heuristike. Egzaktne metode mogu jamčiti optimalnost rezultata. Ove metode pretražuju čitav prostor dopustivih rješenja kako bi došle do optimalnog. No, ovi pristupi su nepraktični i gotovo neupotrebljivi za veće instance problema zbog veličine prostora rješenja. Heuristike, za razliku od egzaktnih metoda, pretražuju samo dio prostora rješenja što ne jamči optimalnost. Ipak, u većini slučajeva generiranje dopustivog i dovoljno dobrog rješenja u ograničenom vremenu puno je važnije od optimalnosti. Stoga su heuristički pristupi popularna i korisna opcija za rješavanje problema raspoređivanja s ograničenim sredstvima. Među njima jedna od najčešće korištenih su prioritetna pravila.[1][5]

3.1 Prioritetna pravila

Heuristike temeljene na prioritetnim pravilima su u širokoj i općoj upotrebi te se smatraju najvažnijom tehnikom za rješavanje problema raspoređivanja s ograničenim sredstvima. Jednostavne su za razumjeti i implementirati, uz to daju dovoljno dobre rezultate čak i za velike probleme. [5]

Za izradu kompletnog rasporeda koristi se kombinacija shema za generiranje rasporeda i prioritetna pravila. Prioritetna pravila biraju sljedeću aktivnost koju će shema za generiranje rasporeda dodati u raspored.[2][4]

Uvodimo sljedeće oznake:

- S_j^* - skup svih sljedbenika aktivnosti A_j
- S_j - skup svih direktnih sljedbenika aktivnosti A_j
- LS_j - najkasniji vremenski trenutak u kojem aktivnost A_j može početi sa svojim izvršavanjem
- LF_j - najkasniji vremenski trenutak u kojem aktivnost A_j može završiti sa svojim izvršavanjem
- ES_j - najraniji vremenski trenutak u kojem aktivnost A_j može početi sa svojim izvršavanjem

Prioritetna pravila koja su korištena u ovom radu dana su u Tablici 3.1. Stupac **Pravilo** sadrži oznaku prioritetnog pravila, stupac **Opis** sadrži hrvatski i engleski naziv danog pravila (oznaka je zapravo skraćenica engleskog naziva pravila), a stupac **Sortiranje** označava odabire li se aktivnost s najmanjom (oznaka *min*) ili najvećom (oznaka *max*) vrijednosti izračunate po formuli iz stupca **Formula**.

Pravilo	Opis	Sortiranje	Formula
GRPW*	Najveći težinski pozicijski rang sljedbenika (engl. *Greatest rank positional weight all*)	max	$p_j + \sum_{i \in S_j^*} p_i$
LST	Najkasniji početak (engl. *Latest starting time*)	min	LS_j
LFT	Najkasniji završetak (engl. *Latest finish time*)	min	LF_j
GRPW	Najveći težinski pozicijski rang direktnih sljedbenika (engl. *Greatest rank positional weight*)	max	$p_j + \sum_{i \in S_j} p_j$
SPT	Najkraće trajanje izvršavanja (engl. *Shortest processing time*)	min	p_j
MSL	Najmanja vremenska odgoda (engl. *Minimum slack time*)	min	$LS_j - ES_j$
MIS	Najveći broj direktnih sljedbenika (engl. *Most immediate successors*)	max	$\lvert S_j \rvert$
MTS	Najveći broj sljedbenika (engl. *Most total successors*)	max	$\lvert S_j^* \rvert$

Tablica 3.1: Prioritetna pravila

3.1.1 Implementacija

Pogledajmo kako se neka od prioritetnih pravila iz Tablice 3.1 implementiraju. Najjednostavnija pravila za računanje su SPT i MIS.

SPT prioritetno pravilo izabere aktivnost s najkraćim vremenom izvršavanja dok MIS izabere aktivnost s najvećim brojem direktnih sljedbenika.

```
1  def SPT(lst):
2      min = inf
3      ret_index = None
4      for el in lst:
5          if el.dur < min:
6              min = el.dur
7              ret_index = el.index
8      return ret_index
9
10
11 def MIS(lst):
12     max = -inf
```

```
13      ret_index = None
14      for el in lst:
15          if len(el.succ) > max:
16              max = len(el.succ)
17              ret_index = el.index
18      return ret_index
```

Za računanje pravila LST i LFT koristi se pomoćna funkcija *calc_LS_LF* koja vraća listu najkasnijih vremenskih trenutaka u kojem aktivnosti mogu početi, odnosno završiti sa svojim izvršavanjem.

```
1  def calc_LS_LF(activities):
2      n = len(activities)
3      LS = [None]*n
4      LF = [None]*n
5      LS[n-1] = sum([activities[i].dur for i in range(n)])
6      LF[n-1] = sum([activities[i].dur for i in range(n)])
7      for j in reversed(range(n-1)):
8          LF[j] = min([LS[i] for i in activities[j].succ])
9          LS[j] = LF[j] - activities[j].dur
10     return LS, LF
11
12 def LST(lst, LS):
13     min = inf
14     ret_index = None
15     for el in lst:
16         if LS[el.index] < min:
17             min = LS[el.index]
18             ret_index = el.index
19     return ret_index
20
21 def LFT(lst, LF):
22     min = inf
23     ret_index = None
24     for el in lst:
25         if LF[el.index] < min:
26             min = LF[el.index]
27             ret_index = el.index
28     return ret_index
```

4 | Sheme za generiranje rasporeda

Shema za generiranje rasporeda (engl. *Schedule generation sheme*, skraćeno SGS) pravi raspored polazeći od praznog skupa uzimajući u obzir uvjete prednosti i ograničenja na sredstva. Raspored se konstruira postupnim proširenjem djelomičnog rasporeda - rasporeda u kojem je raspoređen samo podskup od $n+2$ aktivnosti.

Razlikujemo dvije sheme za generiranje rasporeda: slijednu i usporednu shemu. Glavna razlika između ove dvije sheme je kako grade djelomične rasporede. Kod slijedne sheme u svakoj iteraciji se raspoređuje jedna aktivnost, dok kod usporedne sheme raspoređuje se skup aktivnosti (koji može bit prazan).[4]

Za implementaciju definiramo sljedeće skupove i varijable:

- $A = \{A_0, A_1, \ldots, A_n, A_{n+1}\}$ - skup svih aktivnosti sadržanih u projektu, pri čemu su A_0 i A_{n+1} fiktivne aktivnosti
- $A' = \{A_1, \ldots, A_n\}$ - skup aktivnosti iz kojeg su izostavljene fiktivne aktivnosti
- $p \in \mathbb{N}_0^{n+2}$ - vektor vremena trajanja aktivnosti, pri čemu je p_j, j-ta komponenta vektora p, vrijeme trajanja aktivnosti A_j
- $R = \{R_1, \ldots, R_k\}$ - skup količine dostupnih sredstava koje imamo na raspolaganju za izvršenje projekta
- $\mathcal{K} = \{1, \ldots, K\}$ - skup tipova sredstava
- P_j - skup neposrednih prethodnika aktivnosti A_j
- F_j - vrijeme završetka aktivnosti A_j

4.1 Slijedna shema

Slijedna shema se sastoji od $g = 1, \ldots, n$ iteracija. U svakoj iteraciji, aktivnost se odabire prema svom prioritetu i dodaje se u djelomični raspored u najraniji mogući trenutak uz poštivanje uvjeta prednosti i ograničenja na sredstva. Svakoj iteraciji g pridružena su dva disjunktna skupa aktivnosti: S_g, skup raspoređenih aktivnosti, i E_g, skup dostupnih aktivnosti za raspoređivanje. U svakoj iteraciji, odabire se aktivnost iz skupa E_g pomoću prioritetnog pravila. Nakon što je dana

aktivnost raspoređena, ona se uklanja iz skupa E_g i dodaje u skup S_g. Primijetimo da unija skupova S_g i E_g ne daje skup svih aktivnosti A, jer mogu postojati aktivnosti koje nisu raspoređene i ne mogu se rasporediti u iteraciji g jer nisu svi njihovi prethodnici raspoređeni. Slijedni SGS završava kada S_g sadrži sve postojeće aktivnosti.

Neka je $\mathcal{A}(t) = \{A_j \in A : F_j - p_j \leq t < F_j\}$ skup aktivnosti koje su aktivne, odnosno koje se izvršavaju u trenutku t i neka je $\tilde{R}_k(t) = R_k - \sum_{A_j \in \mathcal{A}(t)} r_{j,k}$ preostala količina sredstva tipa k u trenutku t. Nadalje, neka je $G_g = \{F_j : A_j \in S_g\}$ skup svih vremena završetka aktivnosti koje su do g-te iteracije raspoređene. Skup dostupnih aktivnosti preciznije definiramo formulom $E_g = \{A_j \in A \setminus S_g : P_j \subseteq S_g\}$.

Prema [3] imamo sljedeći algoritam za slijedni SGS.

Algoritam 1 Slijedni SGS

1: **Inicijaliziraj**: $F_0 = 0, S_0 = \{0\}$
2: **Za** $g = 1$ do n **čini**
3: Izračunaj E_g, G_g, $\tilde{R}_k(t)$ $(k \in \mathcal{K}; t \in G_g)$
4: Odaberi jedan $A_j \in E_g$
5: $EF_j = \max_{h \in P_j}\{F_h\} + p_j$
6: $F_j = min\{t \in [EF_j - p_j, LF_j - p_j] \cap G_g : r_{j,k} \leq \tilde{R}_k(\tau), k \in \mathcal{K}, \tau \in [t, t + p_j] \cap G_g\} + p_j$
7: $S_g = S_{g-1} \cup \{A_j\}$
8: $F_{n+1} = max_{h \in P_{n+1}}\{F_h\}$

Inicijalizacija postavlja završno vrijeme fiktivne aktivnosti A_0 na 0 i dodaje ju u djelomični raspored. Na početku svake iteracije g, računa se skup dostupnih aktivnosti E_g, skup završnih vremena G_g i preostala količina sredstva $\tilde{R}_k(t)$ u završnim vremenima $t \in G_g$. Nakon što se pomoću prioritetnog pravila odabere jedna aktivnost iz skupa dostupnih aktivnosti, njeno završno vrijeme se računa tako što se prvo odredi najranija moguća završna vremena EF_j, a zatim se izračuna najranije vrijeme završetaka F_j unutar $[EF_j, LF_j]$ koje je izvedivo prema uvjetima prednosti i ograničenju na sredstva. Raspored S je dan vektorom vremena završetka $(F_1, F_2, \ldots, F_n)$.[3]

Za implementaciju slijednog SGS-a u praktičnom dijelu projekta, koju možemo vidjeti u nastavku, korištene su pomoćne funkcije *Union* i *Intersection* koje računaju uniju, odnosno presjek između dva skupa. Funkcija *calc_Rk* računa četiri skupa sredstava jer u skupovima podataka korištenih u praktičnom dijelu projekta su dana četiri skupa sredstava.

```
1 from prepare_data import init_lists
2 from priority_rules import priority_rule, calc_ES_EF, calc_LS_LF
3
4 def Union(lst1: list, lst2: list):
5     final_list = list(set(lst1) | set(lst2))
```

```
6       return final_list
7
8   def Intersection(lst1: list, lst2: list):
9       final_list = list(set(lst1).intersection(set(lst2)))
10      return final_list
11
12  def calc_Rk(R: list, A: list, t: int):
13      Rk1 = R[0] - sum([A[t][i].res[0] for i in range(len(A[t]))])
14      Rk2 = R[1] - sum([A[t][i].res[1] for i in range(len(A[t]))])
15      Rk3 = R[2] - sum([A[t][i].res[2] for i in range(len(A[t]))])
16      Rk4 = R[3] - sum([A[t][i].res[3] for i in range(len(A[t]))])
17      return [Rk1, Rk2, Rk3, Rk4]
18
19  def calc_A(activities: list, F: list, t: int):
20      ret_lst = []
21      for j in range(len(activities)):
22          try:
23              if F[j]-activities[j].dur <= t and t < F[j]:
24                  ret_lst.append(activities[j])
25          except:
26              continue
27      return ret_lst
28
29  def calc_Eg(activities: list, S: list, g: int):
30      ret_lst = []
31      J = list(set(activities) - set(S[g]))
32      J.sort(key=lambda j: j.index)
33      for j in J:
34          if set(j.pred) <= set([S[g][i].index for i in range(len(S[g
        ]))]):
35              ret_lst.append(j)
36      return ret_lst
37
38  def calc_F(activities: list, j: int, G: list, g: int, F: list, A:
        list, Rk: list, LF: list):
39      if activities[j].pred == []:
40          EFj = activities[j].dur
41      else:
42          EFj = max([F[h] for h in activities[j].pred]) + activities[
        j].dur
43      t_temp = [i for i in range(EFj-activities[j].dur, LF[j]-
        activities[j].dur+1)]
44      t = Intersection(t_temp, G[g])
45      t.sort()
46      ok = []
47      for i in t:
48          bool = 1
49          tau_temp = [i for i in range(i,i+activities[j].dur+1)]
50          tau = Intersection(tau_temp, G[g])
51          tau.sort()
52          for k in tau:
53              condition = activities[j].res[0] <= Rk[k][0] and
        activities[j].res[1] <= Rk[k][1] and activities[j].res[2] <= Rk[
        k][2] and activities[j].res[3] <= Rk[k][3]
54              if Rk[k] != None:
55                  if not condition:
```

```
                    bool = 0
            else:
                A[k] = calc_A(activities, G, k)
                Rk[k] = calc_Rk(4, A, k)
                if not condition:
                    bool=0
        if bool:
            ok.append(i)
        else:
            if ok==[]:
                continue
            elif i-min(ok) < activities[j].dur:
                for i in range(min(ok)+activities[j].dur):
                    if i in ok:
                        ok.remove(i)
    return min(ok) + activities[j].dur

def serial_SGS(pr, activities, R):
    _, F, S, _, G, Rk, A, E = init_lists(activities)
    F[0] = 0
    S[0] = [activities[0]]
    A[0] = [activities[0]]
    j = 0
    ES, _ = calc_ES_EF(activities)
    LS, LF = calc_LS_LF(activities)
    for g in range(1,len(activities)):
        E[g] = calc_Eg(activities, S, g-1)
        G[g] = [F[j.index] for j in S[g-1]]
        for t in [t for t in G[g] if t!=None]:
            A[t] = calc_A(activities, F, t)
            Rk[t] = calc_Rk(R, A, t)
        j = priority_rule(pr ,E[g], activities, ES, LS, LF)
        F[j] = calc_F(activities, j, G, g, F, A, Rk, LF)
        S[g] = Union(S[g-1], [activities[j]])
        S[g].sort(key=lambda j: j.index)
    return G
```

5 | Postupak rješavanja problema

Pogledajmo primjer problema i kako dolazimo do rasporeda koristeći slijedni SGS dan algoritmom 1.

Pokažimo prvo kako računamo LS_j i LF_j, za što nam je potreban krajnji trenutak u planiranju T koji se računa formulom:

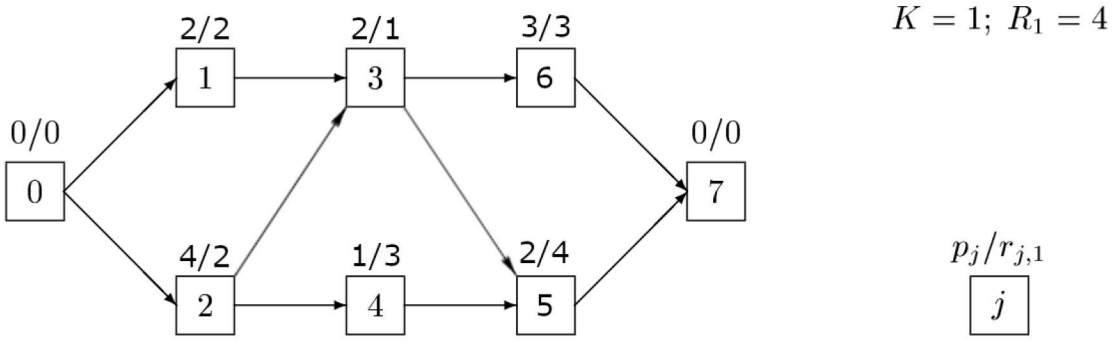
$$T = \sum_{j=1}^{n} p_j. \tag{5.1}$$

Vrijednost T dobivena prethodnom formulom obično je mnogo veća od vrijednosti dobivene nekom od heuristika, jer je u formuli pretpostavljeno da se svaka aktivnost izvršava pojedinačno, tj. nije dopušteno paralelno izvršavanje više aktivnosti.

Računanje vrijednosti LS_j i LF_j radimo iterativno, krenuvši od vrijednosti za zadnju aktivnost pa do prve aktivnosti:

$$\begin{aligned} &LS_{n+1} = LF_{n+1} = T; \\ &LF_j = \min\{LS_i : i \in S_j\}; \\ &LS_j = LF_j - p_j, j \in \{n, \ldots, 0\}. \end{aligned} \tag{5.2}$$

Sada možemo početi s rješavanjem problema. Na Slici 5.1 dan je projekt s $n = 6$ aktivnosti i dvije fiktivne aktivnosti. Potrebno ih je rasporediti uz obnovljivo sredstvo $K = 1$ s kapacitetom $R_1 = 4$. Svakoj aktivnosti dano je trajanje p_j i potražnja za sredstvom $r_{j,1}$. Iz svake aktivnosti se granaju jedna ili dvije strelice čiji vrhovi pokazuju na sljedbenike. Prethodnike neke aktivnosti možemo iščitati prateći strelice unatrag od vrha do baze. Npr. sljedbenici aktivnosti A_3 su aktivnosti A_5 i A_6, a njeni prethodnici su aktivnosti A_1 i A_2. Vrijednosti LF_j svake aktivnosti dani su u Tablici 5.1, a izračunate su koristeći formule dane s (5.2).



Slika 5.1: Primjer problema sa 6 aktivnosti

A_j	1	2	3	4	5	6
LF_j	9	9	11	12	14	14

Tablica 5.1: Najkasniji vremenski trenutci završetka izvršavanja aktivnosti A_j

Po Algoritmu 1, pri inicijalizaciji u skup S_0 dodajemo fiktivnu aktivnost A_0 i njeno vrijeme završetka je $F_0 = 0$ jer joj je trajanje 0.

Zatim ulazimo u petlju i u prvoj iteraciji $g = 1$ imamo $E_1 = \{A_1, A_2\}$, $G_1 = \{0\}$ i $\tilde{R}_k = 4$. Aktivnosti A_1 i A_2 su izabrane u skup dostupnih aktivnosti za raspoređivanje E_1 jer se njihov prethodnik A_0 nalazi u skupu S_0 tj. vrijedi $P_1 \subseteq S_0$ i $P_2 \subseteq S_0$. Koristeći SPT prioritetno pravilo iz Tablice 3.1, iz skupa E_1 izabiremo aktivnost A_1 jer ima kraće trajanje nego aktivnost A_2 ($p_1 = 2 < p_2 = 4$).

Za izračun EF_j potrebno je odrediti najveće završno vrijeme prethodnika aktivnosti A_j i zbrojiti s njegovim trajanjem p_j, a kako je jedini prethodnik od A_1 aktivnost A_0 imamo $EF_1 = max\{F_0\} + p_1 = 0 + 2 = 2$.

Kod računanja vremena završetka F_j, odabire se najraniji vremenski trenutak iz presjeka $[EF_j - p_j, LF_j - p_j] \cap G_g$ takav da su ograničenja na sredstva zadovoljena tijekom cijelog vremena trajanja aktivnosti, što provjerimo prema formuli $r_{j,k} \leq \tilde{R}_k(\tau), k \in \mathcal{K}, \tau \in [t, t + p_j] \cap G_g$, i taj vremenski trenutak zbrojimo s p_j. U našem slučaju za aktivnost A_1 dobijemo $F_1 = min\{[0,7] \cap \{0\}\} + 2 = min\{0\} + 2 = 2$. Primijetimo da je u jedinici vremena $t = 0$ preostali kapacitet sredstva $\tilde{R}_1$ jednak 4, a aktivnost A_1 zahtjeva 2 jedinice sredstva tj. imamo $r_{1,1} = 2 \leq \tilde{R}_1 = 4$ što znači da je u $t = 0$ ograničenje na sredstvo zadovoljeno.

Nakon toga, A_1 se dodaje u skup raspoređenih aktivnosti i dobijemo $S_1 = \{A_0, A_1\}$.

U nastavku se nalazi ostatak računa za preostale aktivnosti. Kao rezultat dobivamo raspred prikazan na Slici 5.2 s vremenom završetka projekta $t = 11$.

$g = 2$

$E_2 = \{A_2\}, G_2 = \{0,2\}, \tilde{R}_1 = \{2,4\};$
Odabrana aktivnost: A_2;
$EF_2 = max\{F_0\} + p_2 = 0 + 4 = 4;$
$F_2 = min\{[0,5] \cap \{0,2\}\} + 4 = min\{0,2\} + 4 = 0 + 4 = 4;$
$S_2 = S_1 \cup \{A_2\} = \{A_0, A_1, A_2\}$

$g = 3$

$E_3 = \{A_3, A_4\}, G_3 = \{0,2,4\}, \tilde{R}_1 = \{0,2,4\};$
Odabrana aktivnost: A_4;
$EF_4 = max\{F_2\} + p_4 = 4 + 1 = 5;$
$F_4 = min\{[4,11] \cap \{0,2,4\}\} + 1 = min\{4\} + 1 = 5;$
$S_3 = S_2 \cup \{A_4\} = \{A_0, A_1, A_2, A_4\}$

$g = 4$

$E_4 = \{A_3\}, G_4 = \{0,2,4,5\}, \tilde{R}_1 = \{0,2,1,4\};$
Odabrana aktivnost: A_3;
$EF_3 = max\{F_1, F_2\} + p_3 = max\{2,4\} + 2 = 4 + 2 = 6;$
$F_3 = min\{[4,9] \cap \{0,2,4,5\}\} + 2 = min\{4,5\} + 2 = 4 + 2 = 6;$
$S_4 = S_3 \cup \{A_3\} = \{A_0, A_1, A_2, A_4, A_3\}$

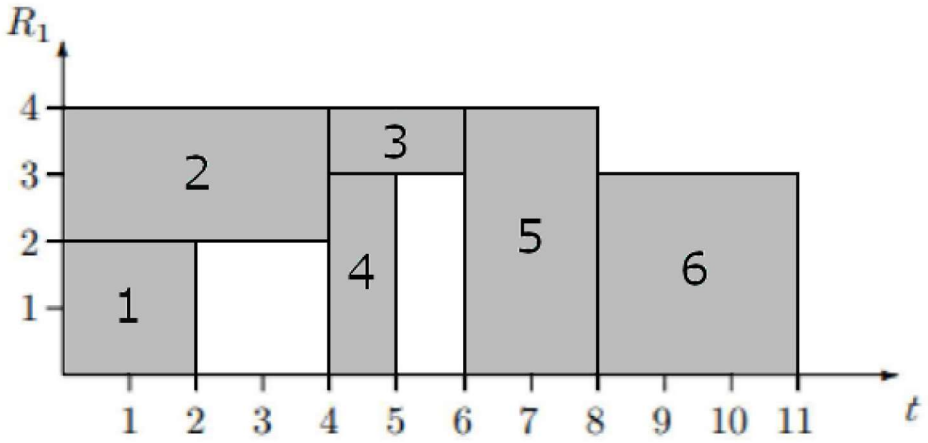
$g = 5$

$E_5 = \{A_5, A_6\}, G_5 = \{0,2,4,5,6\}, \tilde{R}_1 = \{0,2,0,3,4\};$
Odabrana aktivnost: A_5;
$EF_5 = max\{F_3, F_4\} + p_5 = max\{6,5\} + 2 = 6 + 2 = 8;$
$F_5 = min\{[6,12] \cap \{0,2,4,5,6\}\} + 2 = min\{6\} + 2 = 8;$
$S_5 = S_4 \cup \{A_5\} = \{A_0, A_1, A_2, A_4, A_3, A_5\}$

$g = 6$

$E_6 = \{A_6\}, G_6 = \{0,2,4,5,6,8\}, \tilde{R}_1 = \{0,2,0,3,0,4\};$
Odabrana aktivnost: A_6;
$EF_6 = max\{F_3\} + p_6 = 6 + 3 = 9;$
$F_6 = min\{[6,11] \cap \{0,2,4,5,6,8\}\} + 3 = min\{6,8\} + 3 = 8 + 3 = 11;$
$S_6 = S_5 \cup \{A_6\} = \{A_0, A_1, A_2, A_4, A_3, A_5, A_6\}$

$F_7 = max\{F_5, F_6\} = max\{8,11\} = 11$



Slika 5.2: Konačan raspored za 6 aktivnosti uz korištenje SPT prioritetnog pravila

6 | Podaci

6.1 Korišteni podaci

U praktičnom dijelu projekta korišteni su podaci iz PBSLIB biblioteke [6]. Ova biblioteka sadrži skupove problema za različite tipove problema raspoređivanja s ograničenim sredstvima, kao i njihova poznata optimalna i heuristička rješenja. Prisutni su skupovi podataka za probleme s 30, 60, 90 i 120 aktivnosti.

6.2 Učitavanje podataka

Iz skupa podataka moguće je iščitati informacije o sljedbenicima, trajanju i potrebnoj količini sredstva za svaku aktivnost te dostupnost za svaki tip sredstva. Na Slici 6.1 je prikazan primjer jedne takve datoteke.

Funkcijom *read_txt_multi_lines* odabranu datoteku s podacima *file_name* pretvorimo u listu lista s linijama od *start_line* do *finish_line* iz datoteke bez razmaka i znakova novog reda.
Funkcija *read_text_single_line* vrati jednu liniju iz odabrane datoteke *file_name* bez razmaka i znaka novog reda.

```
************************************************************************
file with basedata            : j30_21.bas
initial value random generator: 1057955717
************************************************************************
projects                      :  1
jobs (incl. supersource/sink ):  32
horizon                       :  168
RESOURCES
  - renewable                 :  4   R
  - nonrenewable              :  0   N
  - doubly constrained        :  0   D
************************************************************************
PROJECT INFORMATION:
pronr.  #jobs rel.date duedate tardcost  MPM-Time
    1     30      0       58       12       58
************************************************************************
PRECEDENCE RELATIONS:
jobnr.    #modes  #successors   successors
   1        1          3           2   3   4
   2        1          2          10  31
   3        1          3           5  14  22
   4        1          2           6   7
   5        1          2           8  15
   6        1          2           9  19
   7        1          1          31
   8        1          3          11  16  17
   9        1          1          12
  10        1          2          13  19
  11        1          1          13
  12        1          2          14  26
  13        1          2          29  30
  14        1          1          28
  15        1          1          18
  16        1          1          20
  17        1          2          19  21
  18        1          1          31
  19        1          1          29
  20        1          1          30
  21        1          1          23
  22        1          2          24  27
  23        1          2          24  26
  24        1          1          25
  25        1          2          28  29
  26        1          1          27
  27        1          1          28
  28        1          1          30
  29        1          1          32
  30        1          1          32
  31        1          1          32
  32        1          0
************************************************************************
REQUESTS/DURATIONS:
jobnr. mode duration  R 1  R 2  R 3  R 4
------------------------------------------------------------------------
  1      1     0       0    0    0    0
  2      1     3       0    0    9    0
  3      1     9       0    7    0    0
  4      1     4       9    0    0    2
  5      1    10      10    3    3   10
  6      1     9       0    0    8    4
  7      1     2       2    6    1    0
  8      1     4       8    0    8    4
  9      1     4       0    0    4    0
 10      1    10       0    0    8    0
 11      1     7      10    0    0    0
 12      1     7       0    2    5    6
 13      1     1       3    0    0    0
 14      1     6       0   10    8    0
 15      1     7       0    1    5    1
 16      1     1       0    1    0    5
 17      1     1       3    0    0    0
 18      1     4      10    6    1    0
 19      1     6       9    2    5    9
 20      1     6       0    6    0    3
 21      1     5       6    0    4    0
 22      1     6       5    0    0    0
 23      1     6       0    6    0    8
 24      1    10       0    8    0    5
 25      1     4       0    4    0    9
 26      1     7       9    0    5    0
 27      1     3       0    9    0    5
 28      1     6       2    8    3    8
 29      1     9       0    0    0    7
 30      1     1       0    0    0    4
 31      1    10       5    0    0    7
 32      1     0       0    0    0    0
************************************************************************
RESOURCEAVAILABILITIES:
  R 1  R 2  R 3  R 4
   15   14   12   12
************************************************************************
```

Slika 6.1: Primjer datoteke iz PBSLIB biblioteke s 30 aktivnosti

```
def read_txt_muli_lines(file_name, start_line, finish_line, del_num
    ):
    with open(file_name) as f:
        lines = f.readlines()
        temp = lines[start_line:finish_line] # extract lines
        # list with lines without new line char
        list = []
        for line in temp:
            list.append(line.rstrip('\n'))

        return_list = []
        for l in list:
            lsplit = l.split(' ') # split by space
            # remove spaces from list
            temp_list = []
            for el in lsplit:
                if el.strip():
                    el = int(el)
                    temp_list.append(el)

            del temp_list[1:del_num]
            return_list.append(temp_list)

        return return_list

def read_txt_single_line(file_name, line):
    with open(file_name) as f:
        lines = f.readlines()
        return_list = lines[line].rstrip('\n').split()
        for i in range(len(return_list)):
            return_list[i] = int(return_list[i])
        return return_list
```

Funkcija *prepare_data* za proslijeđeni broj aktivnosti *num* i datoteku s podacima *file_dir* dobavlja za sve aktivnosti informacije o sljedbenicima (*successors*), trajanju i potrebnoj količini sredstva (*dur_res*) te dostupnost za svaki tip sredstva (R). Informacije o sljedbenicima spremamo u rječnik *succ_dict* gdje je svaka aktivnost ključ i pridružena joj je lista njenih sljedbenika. Funkcija vraća listu aktivnosti *activities* s navedenim podacima i listom prethodnika te dostupnost za svaki tip sredstva R. Listu prethodnika željene aktivnosti vraća pomoćna funkcija *get_pred* i za to koristi rječnik sljedbenika.

```
from read_txt import read_txt_muli_lines, read_txt_single_line
from activity import Activity

def get_pred(succ_dict, val):
    if val == 0:
        return []
    ret_lst = []
    for key, value in succ_dict.items():
        for el in value:
            if val == el:
```

```
11                 ret_lst.append(key)
12                 continue
13     return ret_lst
14
15 def prepare_data(num, file_dir):
16     if num=='30':
17         successors = read_txt_muli_lines(file_dir, 18, 50, 3)
18         dur_res = read_txt_muli_lines(file_dir, 54, 86, 2)
19         R = read_txt_single_line(file_dir, 89)
20     elif num=='60':
21         successors = read_txt_muli_lines(file_dir, 18, 80, 3)
22         dur_res = read_txt_muli_lines(file_dir, 84, 146, 2)
23         R = read_txt_single_line(file_dir, 149)
24     elif num=='90':
25         successors = read_txt_muli_lines(file_dir, 18, 110, 3)
26         dur_res = read_txt_muli_lines(file_dir, 114, 206, 2)
27         R = read_txt_single_line(file_dir, 209)
28     elif num=='120':
29         successors = read_txt_muli_lines(file_dir, 18, 140, 3)
30         dur_res = read_txt_muli_lines(file_dir, 144, 266, 2)
31         R = read_txt_single_line(file_dir, 269)
32     else:
33         raise ValueError('Invalid string passed')
34
35     succ_dict = {}
36     for el in successors:
37         for i in range(len(el)):
38             el[i] = el[i]-1
39         succ_dict[el[0]] = el[1:]
40
41     activities = []
42     for i in range(int(num)+2):
43         activities.append(Activity(i, dur_res[i][1], dur_res[i
       ][2:], succ_dict[i], get_pred(succ_dict, i)))
44     return activities, R
```

7 | Korisničko sučelje

7.1 Tkinter

Grafički prikaz rasporeda aktivnosti prikazujemo unutar GUI-ja (engl. *Graphical User Interface*) kojeg stvaramo pomoću Tkinter frameworka za Python. Tkinter je najčešće korištena biblioteka za razvoj korisničkog sučelja u Pythonu poznata po svojoj jednostavnosti i prilagodljivosti.

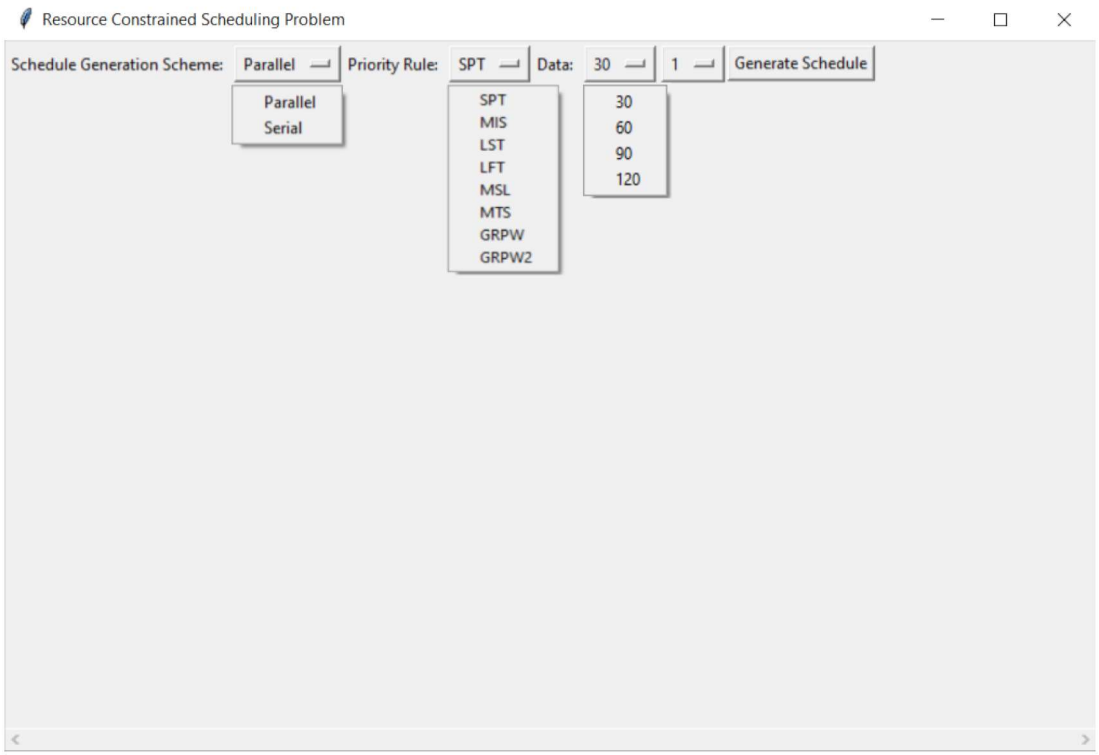
Tkinter koristi *widgete* koji se odnose na pojedinačne elemente sučelja. Svaki widget ima više različitih opcija i mnogo načina za njihovo korištenje i rukovanje. Neki od primjera su gumbi, oznake, izbornik, traka za pomicanje, platno i slično. [7]

7.2 Generiranje rasporeda

Pokretanjem *gui.py* datoteke unutar terminala otvorimo grafičko sučelje prikazano na Slici 7.1. Iz padajućih izbornika možemo birati shemu, prioritetno pravilo i skup podataka sa željenim brojem aktivnosti. Nakon odabira, raspored generiramo pritiskom gumba *Generate Schedule*. Raspored se generira na temelju prethodno odabranih postavki.

Aktivnosti, njihovo trajanje i potražnju za sredstvima u rasporedu crtamo pomoću Python funkcije *broken_barh*( ). Ona se koristi za iscrtavanje horizontalnog stupčastog grafa. Na x-osi prikazano je vrijeme, a na y-osi kolika je potreba za sredstvima pojedine aktivnosti. Aktivnostima su pridružene različite boje kako bi se pojednostavilo iščitavanje iz grafičkog sučelja. Parovi boja i pridruženih aktivnosti se mogu vidjeti u legendi za pojedinačni raspored.

Rasporede sa 60 ili više aktivnosti nije moguće prikazati cijele na zaslonu širine 1920 piksela. Kako bi zaobišli to ograničenje implementirana je traka za pomicanje (engl. *scrollbar*). Trakom za pomicanje moguće je pomicati se lijevo i desno po sučelju za pregled cijelog rasporeda i pripadne legende.



Slika 7.1: Grafičko sučelje s padajućim izbornicima

```
        # drawing activities on a graph
        for i in range(len(J)):
            schedule.broken_barh(
                # [(start_time, duration)],
                [(F[J[i].index]-J[i].dur, J[i].dur)],
                (height[i], J[i].res[r]), # (lower_yaxis, height),
                facecolor=list_of_colors[J[i].index],
                edgecolor='black',
                linewidth = 2,
                label=J[i].index
                )

        # Set x and y axis numbers
        schedule.set_xticks(list(range(maxF+1)))
        schedule.set_yticks(list(range(R[r]+1)))

        # Label ticks and axes
        if r > 0:
            schedule.set_xticklabels('')
        else:
            schedule.set_xlabel('Time', fontsize=15)

        schedule.set_ylabel('R'+str(r+1), fontsize=15)
```

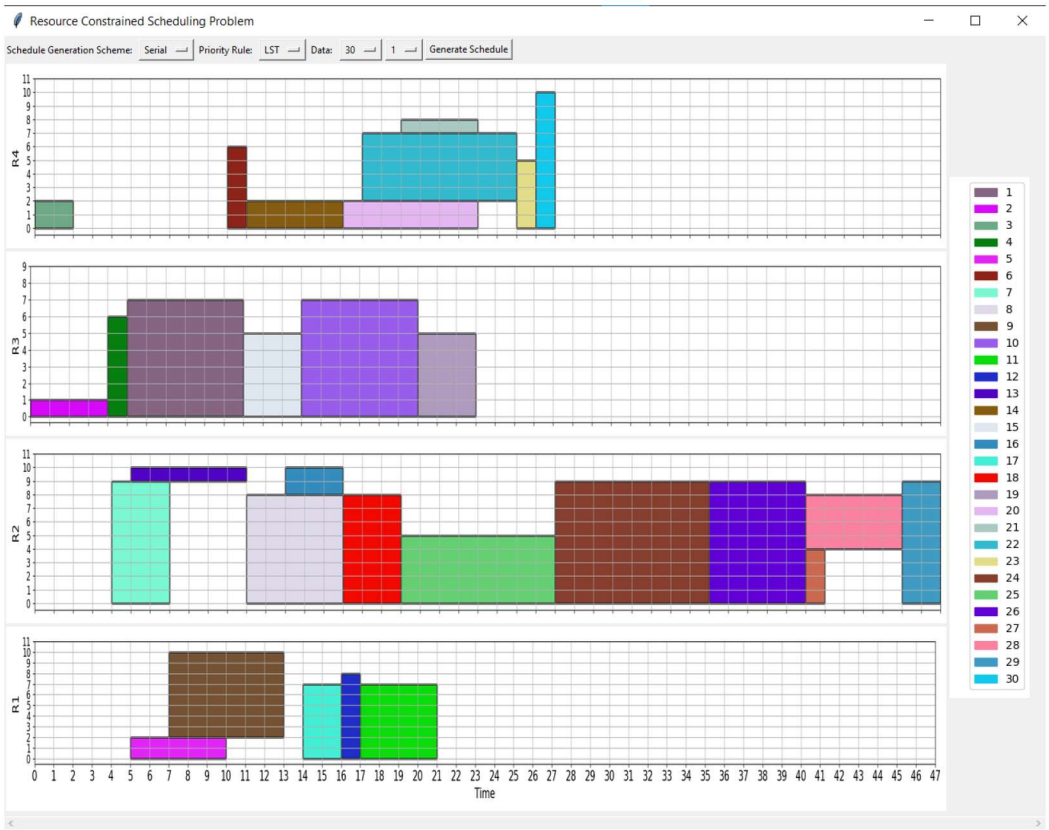
```
# Add a scrollbar to canvas
scrollbar = ttk.Scrollbar(main_frame, orient=HORIZONTAL, command=
    canvas.xview)
scrollbar.pack(side=TOP, fill=X)
```

7.3 Rezultati

Pogledajmo primjere generiranih rasporeda sa 30, 60, 90 i 120 aktivnosti u sučelju za slijedni SGS. Uzmimo za svaki primjer prvi skup podataka ponuđen u izborniku. Za svaki primjer usporedimo koje prioritetno pravilo iz Tablice 3.1 daje najranije, odnosno najkasnije vrijeme završetka rasporeda.

7.3.1 Raspored s 30 aktivnosti

Za odabrani skup podataka za raspored s 30 aktivnosti, prioritetna pravila koja daju najranije vrijeme završetka su LST i GRPW* u trenutku $t = 47$. Najkasnije vrijeme završetka daju pravila SPT i GRPW, $t = 59$. Vremena završetka rasporeda nakon primjene svakog prioritetnog pravila nalaze se u Tablici 7.1.



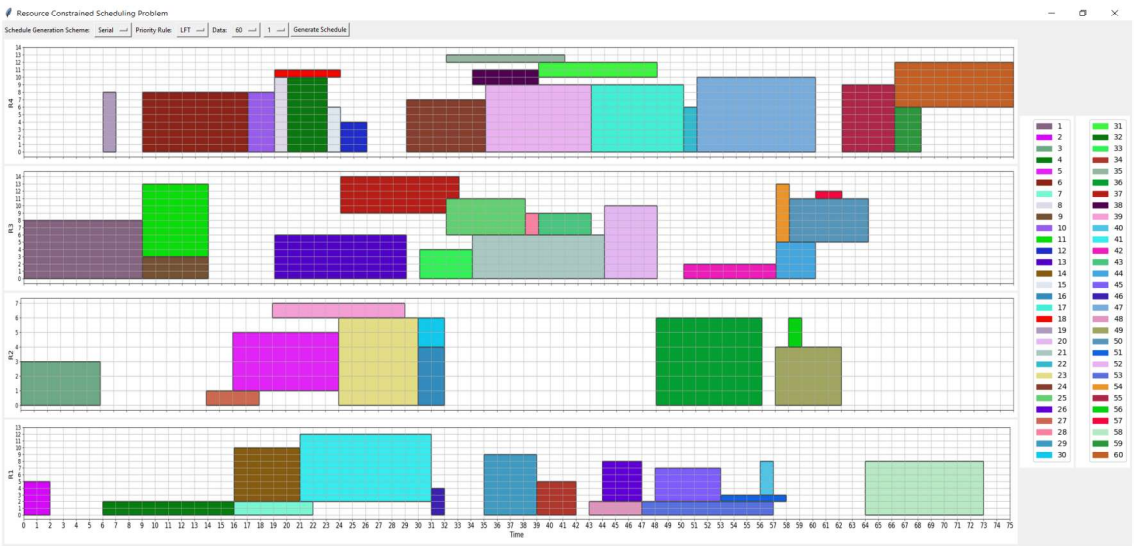
Slika 7.2: Raspored s 30 aktivnosti

Prioritetno pravilo	Vrijeme završetka
SPT	59
MIS	48
LST	47
LFT	50
MSL	55
MTS	50
GRPW	59
GRPW*	47

Tablica 7.1: Vremena završetka rasporeda s 30 aktivnosti

7.3.2 Raspored s 60 aktivnosti

U primjeru za skup podataka sa 60 aktivnosti, prioritetno pravilo koje daje najranije vrijeme završetka je LFT u trenutku $t = 75$. Najkasnije vrijeme završetka daje SPT, $t = 137$. Vremena završetka rasporeda nakon primjene svakog prioritetnog pravila nalaze se u Tablici 7.2.



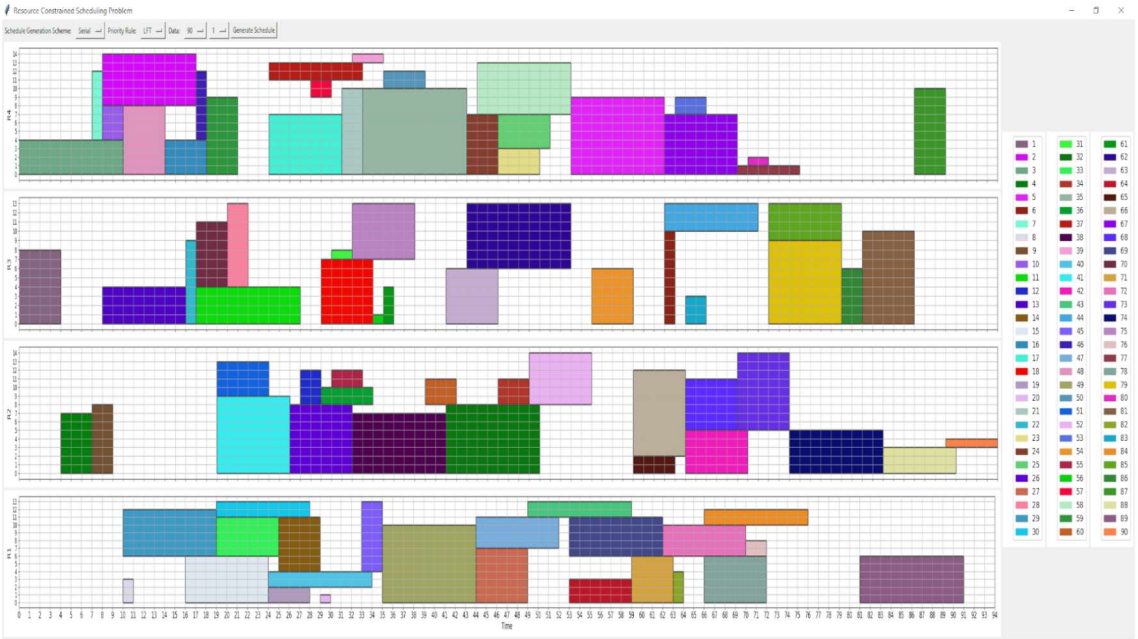
Slika 7.3: Raspored sa 60 aktivnosti

Prioritetno pravilo	Vrijeme završetka
SPT	137
MIS	85
LST	77
LFT	75
MSL	77
MTS	79
GRPW	96
GRPW*	85

Tablica 7.2: Vremena završetka rasporeda sa 60 aktivnosti

7.3.3 Raspored s 90 aktivnosti

U odabranom primjeru za skup podataka s 90 aktivnosti, prioritetno pravilo koje daje najranije vrijeme završetka je LFT u trenutku $t = 94$. Najkasnije vrijeme završetka daje SPT pravilo u trenutku $t = 138$. Vremena završetka rasporeda nakon primjene svakog prioritetnog pravila nalaze se u Tablici 7.3.



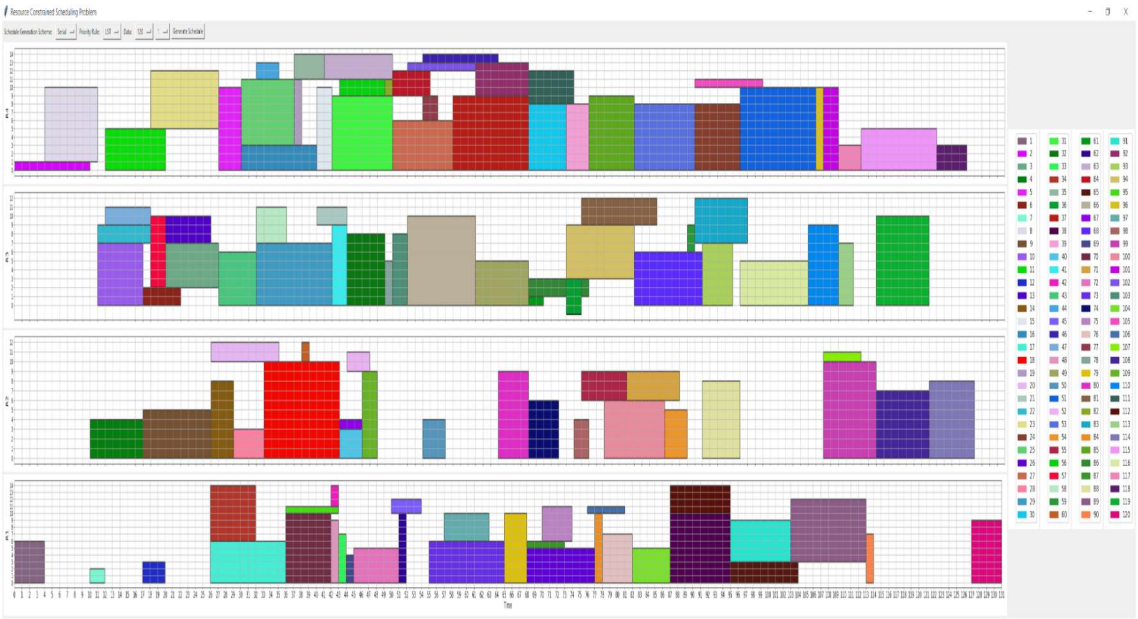
Slika 7.4: Raspored s 90 aktivnosti

Prioritetno pravilo	Vrijeme završetka
SPT	138
MIS	109
LST	95
LFT	94
MSL	124
MTS	101
GRPW	124
GRPW*	99

Tablica 7.3: Vremena završetka rasporeda s 90 aktivnosti

7.3.4 Raspored sa 120 aktivnosti

U odabranom primjeru za skup podataka za raspored sa 120 aktivnosti, prioritetna pravila koja daju najranije vrijeme završetka su LST i MTS u trenutku $t = 131$. Najkasnije vrijeme završetka daje GRPW pravilo u trenutku $t = 191$. Vremena završetka rasporeda nakon primjene svakog prioritetnog pravila nalaze se u Tablici 7.4.



Slika 7.5: Raspored sa 120 aktivnosti

Prioritetno pravilo	Vrijeme završetka
SPT	175
MIS	164
LST	131
LFT	133
MSL	144
MTS	131
GRPW	191
GRPW*	137

Tablica 7.4: Vremena završetka rasporeda sa 120 aktivnosti

Literatura

[1] Mohammad Abdolshah, *A Review of Resource-Constrained Project Scheduling Problems (RCPSP) Approaches and Solutions*, International Transaction Journal of Engineering, Management, Applied Sciences and Technologies, 2014.

[2] Mateja Đumić, Dominik Šišejković, Rebeka Čorić, Domagoj Jakobović, *Evolving priority rules for resource constrained project scheduling problem with genetic programming*, Future Generation Computer Systems, vol. 86, 2018., str. 211-221

[3] Reiner Kolisch and Sönke Hartman, *Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis*, Christian-Albrechts-Universität zu Kiel, 1998.

[4] Reiner Kolisch, *Serial and parallel resource-constrained project scheduling methodes revisited: Theory and computation*, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 344, 1994.

[5] Antonio Lova, Pilar Tormos, Federico Barber, *Multi-Mode Resource Constrained Project Scheduling: Scheduling Schemes, Priority Rules and Mode Selection Rules*, Revista Iberoamericana de Inteligencia Artificial, vol. 10, 2006., str. 69-86

[6] Web izvor dostupan na `https://www.om-db.wi.tum.de/psplib/library.html` (PSBLIB biblioteka).

[7] Web izvor dostupan na `https://coderslegacy.com/python/python-gui/` (Python GUI with Tkinter).

Sažetak

U ovom radu opisan je problem raspoređivanja s ograničenim sredstvima i metode koje koristimo za rješavanje danog problema. Upoznajemo se s najpoznatijim prioritetnim pravilima iz literature i pokazujemo kako generirati raspored koristeći slijednu shemu za generiranje rasporeda i prioritetna pravila. U praktičnom dijelu rada, napravljena je implementacija prioritetnih pravila i slijedne sheme za generiranje rasporeda te njihovim korištenjem riješeno nekoliko primjera problema raspoređivanja s ograničenim sredstvima.

Ključne riječi

prioritetna pravila, raspoređivanje, aktivnosti, sredstva, slijedna shema za generiranje rasporeda

Serial schedule generation scheme and priority rules for solving resource constrained scheduling problem

Summary

This paper describes the resource constrained project scheduling problem and the methods we use to solve this problem. We familiarize ourselves with the most well-known priority rules from the literature and show how to generate a schedule using the serial schedule generation scheme and priority rules. In the practical part of the work, the implementation of priority rules and the serial scheme for generating schedules were made, and several examples of resource constrained scheduling problem were solved using them.

Keywords

priority rules, scheduling, activities, resources, serial schedule generation scheme

Životopis

Moje ime je Anita Hančić i rođena sam 24. siječnja 2001. u Slavonskom Brodu. Osnovnu školu Vladimir Nazor pohađala sam od 2007. do 2015. godine u Slavonskom Brodu. Zatim od 2015. do 2019. sam pohađala Klasičnu gimnaziju fra Marijana Lanosovića s pravom javnosti isto u Slavonskom Brodu. Po završetku srednjoškolskog obrazovanja upisujem prijediplomski studij Matematike i računarstva na tadašnjem Odjelu za matematiku Sveučilišta Josipa Jurja Strossmayera u Osijeku, sadašnjem Fakultetu primjenjene matematike i informatike.