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

Ladnjak, Mirna

Sveučilište Josipa Jurja Strossmayera u Osijeku

Odjel za matematiku

Sveučilišni preddiplomski studij Matematika i Računarstvo

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

Završni rad

Autor:

Mirna Ladnjak

Osijek, 2022

Sveučilište Josipa Jurja Strossmayera u Osijeku

Odjel za matematiku

Sveučilišni preddiplomski studij Matematika i Računarstvo

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

Završni rad

Mentor:

doc.dr.sc. Domagoj Ševerdija

Autor:

Mirna Ladnjak

Komentori:

dr.sc. Mateja Đumić

dr.sc. Rebeka Čorić

Osijek, 2022

Sažetak

U ovom završnom radu bit će opisan problem raspoređivanja s ograničenim sredstvima te metode kojima se koristimo za rješavanje tog problema. Upoznat ćemo se s prioritetnim pravilima te vidjeti kako pomoću njih uz usporednu shemu za generiranje rasporeda možemo doći do rješenja problema. Kako bismo bolje razradili ovu temu i vidjeli djelovanje prioritetnih pravila na proizvoljan skup aktivnosti, izrađen je i praktični dio završnog rada. Na kraju rada bit će prikazani primjeri nekoliko generiranih rasporeda.

Ključne riječi

prioritetna pravila, raspoređivanje, sredstva, usporedna shema za generiranje rasporeda

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

Summary

This paper describes the resource constrained project scheduling problem and the methods that we use for solving this problem. We will get familiar with priority rules and see how we can reach a solution to the problem by using them along with a parallel schedule generation scheme. In order to better elaborate this topic and to see the effect of priority rules on an arbitrary set of activities, a practical part of the paper was made. At the end of this paper, examples of several generated schedules will be presented and analyzed.

Key words

priority rules, scheduling, resources, parallel schedule generation scheme

Sadržaj

Uvod

Problem raspoređivanja s ograničenim sredstvima (engl. RCPSP - *Resource Constrained Project Scheduling Problem*) sastoji se od skupa sredstava koja imaju ograničenu dostupnost i aktivnosti čije je trajanje i potražnja za sredstvima poznato, a te aktivnosti povezane su informacijom o prethodnicima. [2] Uveo ga je James E. Kelley Jr. davne 1963. godine, no i dalje se smatra standardnim problemom kada govorimo o projektima s ograničenim sredstvima.

Metoda raspoređivanja pomoću prioritetnih pravila jedan je od načina rješavanja ovog problema prigodan za okruženja u kojima se događaju promjene unutar sustava. Kako bi došli do rješenja problema, koristimo prioritetna pravila u kombinaciji sa shemom za generiranje rasporeda (engl. SGS - *Schedule Generation Scheme*). [5] U ovom radu usredotočit ćemo se na usporednu shemu za generiranje rasporeda.

Riješiti problem raspoređivanja s ograničenim sredstvima znači pronaći raspored za dani problem pri čemu je potrebno optimizirati jedan ili više kriterija. Najčešći kriterij koji se koristi prilikom rješavanja problema je minimizacija vremena završetka projekta.

1 | Problem raspoređivanja s ograničenim sredstvima

Problem raspoređivanja s ograničenim sredstvima sastoji se od skupa aktivnosti $\mathcal{J} = \{0, 1, ..., n, n+1\}$ gdje aktivnosti *0* i *n+1* nazivamo fiktivnim aktivnostima koje označavaju početak odnosno kraj projekta. Aktivnosti su međusobno povezane uvjetima prednosti (engl. *precedence constraints*) koji ne dopuštaju da aktivnost j započne prije nego što su završile sve aktivnosti sadržane u skupu P_j, gdje je P_j skup svih neposrednih prethodnika aktivnosti j.

S druge strane, svaka aktivnost pri izvršavanju ima potražnju za sredstvima čija je količina u svim vremenskim jedinicama jednaka što znači da je sredstvo je obnovljivo, a dostupnost im je ograničena. Imamo K tipova sredstava koji su označeni skupom $\mathcal{K} = \{1, ..., K\}$. Za vrijeme izvršavanja aktivnost j zahtijeva $r_{j,k}$ jedinica sredstva tipa $k \in \mathcal{K}$ tijekom cijelog razdoblja svog izvršavanja koje ima trajanje p_j. Za fiktivne aktivnosti 0 i $n+1$ vrijedi $p_j = 0$ i $r_{j,k} = 0$, za svaki $k \in \mathcal{K}$.

Za rješavanje problema raspoređivanja s ograničenim sredstvom pomoću prioritetnih pravila potrebno je uvesti sljedeće:

- $F_j^*(F_j)$ - skup svih (direktnih) sljedbenika aktivnosti j
- LS_j - najkasniji vremenski trenutak u kojem aktivnost j može početi sa svojim izvršavanjem
- LF_j - najkasniji vremenski trenutak u kojem aktivnost j može završiti sa svojim izvršavanjem
- ES_j - najraniji vremenski trenutak u kojem aktivnost j može početi sa svojim izvršavanjem
- ES_j - najraniji vremenski trenutak u kojem aktivnost j može početi sa svojim izvršavanjem

Cilj inačice problema raspoređivanja s ograničenim sredstvima koja je proučavana u ovom radu jest pronaći takav raspored da ograničenja na sredstva i uvjeti prednosti budu zadovoljeni pri čemu je vrijeme završetka projekta minimalno. [4]

2 | Metode rješavanja problema

Postoje dvije skupine metoda rješavanja problema raspoređivanja s ograničenim sredstvima: egzaktne i heuristike. Egzaktne metode imaju mogućnost dobivanja optimalnog rješenja pretraživanjem cijelog dopustivog prostora rješenja. Zbog kompleksnosti problema raspoređivanja s ograničenim sredstvima, egzaktnim metodama možemo riješiti samo probleme manje veličine. Zato se u praksi uglavnom koriste heuristike. Umjesto pretraživanja cijelog dopustivog prostora rješenja, heuristike pretražuju samo dio prostora, zbog čega ne garantiraju optimalno rješenje ali daju dovoljno dobra rješenja za veće projekte. Jedna od najčešće korištenih heuristika su prioritetna pravila. [1]

2.1 Prioritetna pravila

Heuristike temeljene na prioritetnim pravilima smatraju se jednima od najbitnijih tehnika za rješavanje problema raspoređivanja s ograničenim sredstvima. Razlog tome je što su prioritetna pravila vrlo laka za razumjeti i implementirati te daju za praksu dovoljno dobra rješenja čak i za veće projekte. [6]

Prioritetna pravila primjenjuju se unutar sheme za generiranje rasporeda i služe za odabir jedne od aktivnosti koja će biti sljedeća na redu za stavljanje u raspored. Ovisno o pravilu, uzima se aktivnost s najvećom ili najmanjom vrijednošću prioriteta. Do sada je zabilježeno postojanje čak i do 73 prioritetna pravila, no mi ćemo uzeti u obzir ona koja su do sada najčešće korištena i koja su pokazala najbolje rezultate. [3] Prioritetna pravila koja se najčešće koriste u literaturi [4], a koja smo koristili u ovom radu navedena su u Tablici 2.1.

Prioritetno pravilo	Opis	Tip sortiranja	Formula za računanje
GRPW*	najveći težinski pozicijski rang sljedbenika (engl. *greatest rank positional weight all*)	max	$p_j + \Sigma_{i \in F_j^*} p_i$
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	$p_j + \Sigma_{i \in F_j} p_i$
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	$\mid F_j \mid$
MTS	najveći broj sljedbenika (engl. *most total successors*)	max	$\mid F_j^* \mid$

Tablica 2.1: Prioritetna pravila

2.1.1 Implementacija

U nastavku možemo vidjeti način na koji su neka prioritetna pravila iz Tablice 2.1 implementirana u završnom praktičnom projektu. Najjednostavnije je bilo izračunati formulu za SPT prioritetno pravilo prikazano na Slici 2.1 koje uzima aktivnost s najkraćim vremenom izvršavanja.

```
26  def SPT(lst):
27      min = inf
28      ret_index = None
29      for el in lst:
30          if el.dur < min:
31              min = el.dur
32              ret_index = el.index
33      return ret_index
```

Slika 2.1: SPT prioritetno pravilo

Za izračun formule za MTS prioritetno pravilo potrebno je znati ukupan broj sljedbenika svake aktivnosti kako bismo znali koja aktivnost ih ima najviše. Zato nam je za implementaciju tog prioritetnog pravila bila potrebna pomoćna funkcija *total_successors* (vidi Sliku 2.2).

```
109 def total_successors(activities, j, lst):
110     if activities[j].succ == []:
111         return
112     else:
113         for i in activities[j].succ:
114             lst.append(i)
115             total_successors(activities, i, lst)
116         lst = list(dict.fromkeys(lst))
117         lst.sort()
118         return lst
119
120 def MTS(lst, activities):
121     max = -inf
122     ret_index = None
123     for el in lst:
124         if el == activities[-1]:
125             return el.index
126         total_succ = total_successors(activities, el.index, [])
127         if len(total_succ) > max:
128             max = len(total_succ)
129             ret_index = el.index
130     return ret_index
```

Slika 2.2: MTS prioritetno pravilo i pomoćna funkcija

3 | Sheme za generiranje rasporeda

Postoje dvije vrste shema za generiranje rasporeda, usporedna i slijedna shema. Obje sheme generiraju dopustive rasporede za dane instance problema uz korištenje prioritetnih pravila. Sheme započinju s praznim rasporedom te kroz iteracije grade djelomične rasporede (engl. *partial schedule*) sve dok postoje aktivnosti koje nisu raspoređene. Djelomični raspored je raspored gdje je samo podskup od $n+2$ aktivnosti raspoređen, odnosno dodijeljeno im je početno vrijeme. Glavna razlika između usporedne i slijedne sheme je kako grade djelomične rasporede. Kod slijedne sheme za generiranje rasporeda u svakoj iteraciji se raspoređuje točno jedan posao, dok se kod usporedne sheme u pojedinoj iteraciji može rasporediti više od jedne aktivnosti. [5]

Prilikom implementacije shema korišteni su sljedeći skupovi, odnosno varijable:

- $\mathcal{J} = \{0, ..., n+1\}$ - skup svih aktivnosti sadržanih u projektu, pri čemu su s 0 i $n+1$ označene fiktivne aktivnosti koje označavaju početak, odnosno završetak projekta i čije je trajanje jednako 0
- $\mathcal{J}' = \{1, ..., n\}$ - skup svih aktivnosti iz kojeg su izostavljene fiktivne aktivnosti
- $p \in N_0^{n+2}$ - vrijeme izvršenja aktivnosti dano vektorom, pri čemu je i-ta komponenta p_i vektora p vrijeme izvršavanja aktivnosti A_i i također vrijedi $p_0 = p_{n+1} = 0$
- $R = \{R_1, ..., R_k\}$ - skup sredstava koje imamo na raspolaganju za izvršavanje projekta
- $\tilde{R}_k(t) = R_k - \Sigma_{j \in A(t)} r_{j,k}$ - preostala količina sredstva tipa k u trenutku t
- P_j - skup neposrednih prethodnika aktivnosti s indeksom j

3.1 Usporedna shema

Usporedna shema za generiranje rasporeda temelji se na inkrementaciji vremenskih trenutaka (engl. *time-incrementation*). To znači da je svaka iteracija g povezana s nekim vremenom t_g. U svakoj pojedinoj iteraciji može se rasporediti više od jedne aktivnosti, a broj iteracija usporedne sheme jednak je broju vremenskih jedinica u kojem je postojala mogućnost da se neka aktivnost rasporedi.

Aktivnosti koje su raspoređene do iteracije g nalaze se ili u skupu završenih aktivnosti C_g ili u skupu aktivnih aktivnosti A_g. Skup završenih aktivnosti $C_g = \{j \in \mathcal{J} \mid F_j \leq t_g\}$ sadrži sve aktivnosti koje su završile prije trenutka t_g, dok skup aktivnih aktivnosti $A_g = A(t_g) = \{j \in \mathcal{J} \mid F_j - p_j \leq t \leq F_j\}$ sadrži sve aktivnosti koje su raspoređene ali nisu još završile s izvršavanjem do trenutka t_g. Trenutno dostupne aktivnosti (engl. *eligible activities*) koje mogu započeti u vremenu t_g nalaze se u skupu $E_g = \{j \in \mathcal{J} \backslash (C_g \cup A_g) \mid P_j \subseteq C_g \wedge r_{j,k} \leq \tilde{R}_k(t_g)(k \in \mathcal{K})\}$. [4]

Za usporedni SGS imamo sljedeći algoritam prema [4].

Algoritam 1 Usporedni SGS

1: **Inicijalizacija** : $g = 0, t_g = 0, A_0 = \{0\}, C_0 = \{0\}, \tilde{R}_k(0) = R_k$
2: **Dok je** $|A_g \cup C_g| \leq n$ **radi**
3: $g = g + 1$
4: $t_g = min_{j \in A_g}\{F_j\}$
5: Izračunaj: $A_g, C_g, \tilde{R}_k(t_g), E_g$
6: **Dok je** $|E_g| > 0$ **radi**
7: Odaberi $j \in E_g$
8: $F_j = t_g + p_j$
9: Izračunaj: $A_g, \tilde{R}_k(t_g), E_g$
10: **kraj**
11: **kraj**
12: $F_{n+1} = max_{h \in P_{n+1}} F_h$

Inicijalizacija postavlja vrijeme na 0 i prvo stavlja fiktivnu aktivnost u raspored te postavi slobodne jedinice sredstva na odgovarajuću vrijednost. Prvi korak je odrediti trenutno vrijeme raspoređivanja t_g na način da se uzme minimalno završno vrijeme svih do sada raspoređenih aktivnosti. Nakon toga računaju se skupovi C_g, A_g i E_g zajedno s trenutnom dostupnošću odgovarajućeg sredstva.

Drugi korak svake iteracije je staviti u raspored one aktivnosti iz skupa E_g koje mogu započeti u vremenu t_g. Implementaciju usporednog SGS-a iz praktičnog dijela završnog rada možemo vidjeti u nastavku gdje su *union* i *not_scheduled* dvije pomoćne funkcije.

```
from prepare_data import init_lists
from priority_rules import priority_rule, calc_ES_EF, calc_LS_LF

def union(lst1, lst2):
    final_list = list(set(lst1) | set(lst2))
    return final_list

def not_scheduled(J, C):
    return list(set(J) - set(C))

def cacl_Cg(activities, F, tg) :
    ret_lst = []
    for j in range(len(activities)):
        try:
            if F[j] <= tg:
                ret_lst.append(activities[j])
        except:
            continue
    return ret_lst

def calc_Ag(activities, F, tg):
    ret_lst = []
    for j in range(len(activities)):
        try:
            if F[j]-activities[j].dur <= tg and tg < F[j]:
                ret_lst.append(activities[j])
        except: continue
    return ret_lst
```

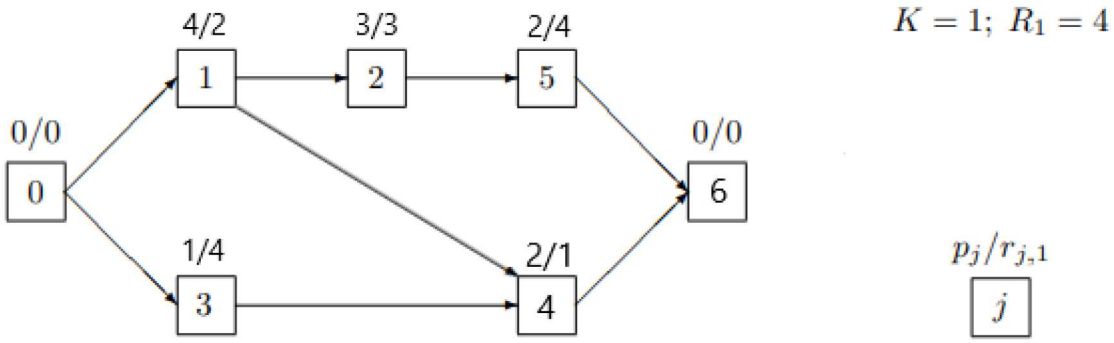
```
def calc_Rk(R, A, g):
    Rk1 = R[0] - sum([A[g][i].res[0] for i in range(len(A[g]))])
    Rk2 = R[1] - sum([A[g][i].res[1] for i in range(len(A[g]))])
    Rk3 = R[2] - sum([A[g][i].res[2] for i in range(len(A[g]))])
    Rk4 = R[3] - sum([A[g][i].res[3] for i in range(len(A[g]))])
    return [Rk1, Rk2, Rk3, Rk4]

def calc_Eg(activities, C, A, Rk, t, g):
    ret_lst = []
    J = list(set(activities) - set(Union(C[g],A[g])))
    J.sort(key=lambda j: j.index)
    for j in J:
        if (set(j.pred) <= set([C[g][i].index for i in
          range(len(C[g]))]) and (j.res[0] <= Rk[t[g]][0]
            and j.res[1] <= Rk[t[g]][1] and j.res[2] <=
            Rk[t[g]][2] and j.res[3] <= Rk[t[g]][3])):
                ret_lst.append(j)
    return ret_lst

def parallel_SGS(pr, activities, R):
    t, F, _, A, C, Rk, _, E = init_lists(activities)
    g=0
    t[g] = 0
    A[0] = [activities[0]]
    C[0] = [activities[0]]
    Rk[0] = [R,R,R,R]
    F[0] = 0
    ES, _ = calc_ES_EF(activities)
    LS, LF = calc_LS_LF(activities)
    while len(Union(A[g],C[g])) <= len(activities)-2:
        g += 1
        t[g] = min([F[j.index] for j in A[g-1]])
        C[g] = cacl_Cg(activities, F, t[g])
        A[g] = calc_Ag(activities, F, t[g])
        Rk[t[g]] = calc_Rk(R, A, g)
        E[g] = calc_Eg(activities, C, A, Rk, t, g)
```

```
        while len(E[g]) > 0:
            j = priority_rule(pr, E[g], activities, ES, LS, LF)
            F[j] = t[g] + activities[j].dur
            A[g] = calc_Ag(activities, F, t[g])
            Rk[t[g]] = calc_Rk(R, A, g)
            E[g] = calc_Eg(activities, C, A, Rk, t, g)
    F[len(activities)-1] = max([F[h] for h in
    activities[len(activities)-1].pred])
    return F
```

4 | Postupak rješavanja problema

Pogledajmo jedan primjer problema i kako dolazimo do rješenja.

Neka je na Slici 4.1 dan projekt s $n = 5$ aktivnosti, uz dodatne dvije fiktivne aktivnosti. Potrebno ih je staviti u raspored uz obnovljivo sredstvo tipa $K = 1$ s kapacitetom od 4 jedinice. Za svaku aktivnost zadano je njezino trajanje p_j i potražnja za sredstvom $r_{j,1}$.



Slika 4.1: Primjer problema s 5 aktivnosti

Ako odaberemo SPT za prioritetno pravilo tada koristeći usporednu shemu za generiranje rasporeda dobivamo sljedeće.

Prateći Algoritam 3.1, prva aktivnost koju stavljamo u raspored je nulta odnosno fiktivna aktivnost $j = 0$. Zbog toga imamo $t_1 = 0, A_1 = \{\}, C_1 = \{0\}$ i $\tilde{R}_k(0) = 4$. Prateći definiciju skupa mogućih aktivnosti imamo da je $E_1 = \{1, 3\}$. Razlog tome je što aktivnosti 1 i 3 obje za prethodnika imaju aktivnost 0 koja je sada u skupu C_g pa vrijedi uvjet $P_1 \subseteq C_1, P_3 \subseteq C_1$ i obje aktivnosti imaju potražnju za sredstvom manju ili jednaku 4. S obzirom da se koristi SPT, sljedeću uzimamo onu aktivnost iz skupa E_1 koja ima najkraće vrijeme izvršavanja što će u ovom slučaju biti aktivnost $j = 3$ jer je $p_3 = 1 \leq p_1 = 4$. Sada imamo $F_3 = 1, A_1 = \{3\}, \tilde{R}_k(0) = 0$ i $E_1 = \{\ \}$.

Nastavljajući ovaj postupak i za preostale aktivnosti, kao rezultat dobivamo raspored prikazan na Slici 4.2 s vremenom završetka projekta u trenutku $t = 10$.

$g = 2$

$F_3 = 1, A_1 = \{\}, C_1 = \{0\}$
$t_2 = 1$
Izračunano:
$A_2 = \{\}, C_2 = \{0,3\}, \tilde{R}_k(1) = 4$ i
$E_2 = \{1\}$
Odabrano: $j = 1$
$F_1 = 5$
Izračunano: $A_2 = \{1\}, \tilde{R}_k(1) = 2$ i
$E_2 = \{\ \}$

$g = 3$

$A_2 = \{\}, C_2 = \{0,3\}$
$t_3 = 5$
Izračunano:
$A_3 = \{\}, C_3 = \{0,3,1\}, \tilde{R}_k(5) = 4$ i
$E_3 = \{2,4\}$
Odabrano: $j = 4$
$F_4 = 7$
Izračunano: $A_3 = \{4\}, \tilde{R}_k(5) = 3$ i
$E_3 = \{2\}$
Odabrano: $j = 2$
$F_2 = 8$
Izračunano: $A_3 = \{2,4\}, \tilde{R}_k(5) = 0$ i
$E_3 = \{\ \}$

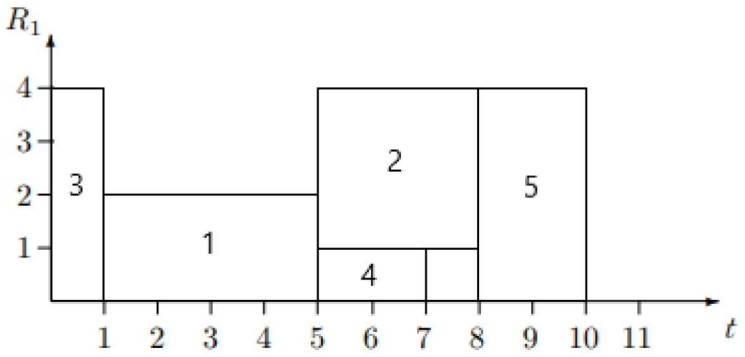
$g = 4$

$A_3 = \{\}, C_3 = \{0,3,1\}$
$t_4 = 7$
Izračunano: $A_4 = \{\}, C_4 = \{0,3,1,4,2\}$,
$\tilde{R}_k(7) = 1$ i $E_4 = \{\ \}$

$g = 5$

$A_4 = \{\}, C_4 = \{0,3,1,4,2\}$
$t_5 = 8$
Izračunano: $A_5 = \{\}, C_5 = \{0,3,1,4,2\}$,
$\tilde{R}_k(8) = 4$ i $E_5 = \{5\}$
Odabrano: $j = 5$
$F_5 = 10$
Izračunano: $A_5 = \{5\}, \tilde{R}_k(8) = 0$ i
$E_5 = \{\ \}$

$F_{n+1} = F_6 = max_{h \in P_7} F_h = 10$



Slika 4.2: Konačan raspored za 5 aktivnosti uz SPT

5 | Podaci

5.1 Korišteni podaci

Za izradu praktičnog dijela završnog rada korišteni su podaci iz PSPLIB biblioteke koja se može pronaći na linku [7]. Biblioteka sadrži skupove problema za različite vrste problema planiranja projekta s ograničenim sredstvima, kao i optimalna i heuristička rješenja. Odlaskom na *Download data* i zatim na *Resource-Constrained Project Scheduling Problem* (*RCPSP*) stranicu, dolazimo do skupova podataka za probleme s 30, 60, 90 ili 120 aktivnosti.

5.2 Učitavanje podataka

Iz odabrane datoteke s podacima moguće je za svaku aktivnost dohvatiti informacije o sljedbenicima, trajanju i potrebnoj količini sredstava. Za svako sredstvo unaprijed je određena njegova dostupnost. Primjer jedne takve datoteke prikazan je na Slici 5.1.

Funkcija *read_text_multi_lines* prikazana na Slici 5.2 uzima linije koda od *start_line* do $finish\_line$-a iz odabrane datoteke $file\_name$ te uklanja sve znakove novog reda $\backslash n$ iz koda. Svaka linija iz datoteke, nakon što uklonimo razmake, sprema se kao lista. Funkcija vraća listu listi bez praznih znakova.

Funkcija *read_txt_single_line* prikazana na Slici 5.3 učita pojedinačnu liniju iz datoteke, ukloni znakove novog reda i pozove $split()$ metodu na njima.

```
******************************************************************************
file with basedata            : j30_17.bas
initial value random generator: 1989763511
******************************************************************************
projects                      :  1
jobs (incl. supersource/sink ):  32
horizon                       :  119
RESOURCES
  - renewable                 :  4   R
  - nonrenewable              :  0   N
  - doubly constrained        :  0   D
******************************************************************************
PROJECT INFORMATION:
pronr.  #jobs rel.date duedate tardcost  MPM-Time
    1     30      0       31       24       31
******************************************************************************
PRECEDENCE RELATIONS:
jobnr.    #modes  #successors   successors
   1        1          3           2   3   4
   2        1          3          11  12  15
   3        1          3           5   8  16
   4        1          1          27
   5        1          3           6  12  14
   6        1          1           7
   7        1          2           9  28
   8        1          2          10  20
   9        1          2          13  21
  10        1          2          17  25
  11        1          2          25  28
  12        1          2          25  29
  13        1          3          22  23  31
  14        1          1          17
  15        1          1          24
  16        1          1          18
  17        1          1          26
  18        1          2          19  23
  19        1          1          22
  20        1          1          29
  21        1          1          31
  22        1          1          24
  23        1          1          24
  24        1          1          29
  25        1          1          30
  26        1          1          28
  27        1          1          30
  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     6       0    0    7    0
  3      1     4       0    0    1    0
  4      1     2       0    0    0    2
  5      1     1       0    0    6    0
  6      1     5       2    0    0    0
  7      1     1       0    0    0    6
  8      1     3       0    9    0    0
  9      1     5       0    8    0    0
 10      1     6       8    0    0    0
 11      1     6       0    0    7    0
 12      1     4       7    0    0    0
 13      1     1       8    0    0    0
 14      1     6       0    1    0    0
 15      1     5       0    0    0    2
 16      1     3       0    0    5    0
 17      1     3       0    2    0    0
 18      1     2       7    0    0    0
 19      1     3       0    8    0    0
 20      1     3       0    0    5    0
 21      1     7       0    0    0    2
 22      1     4       0    0    0    1
 23      1     8       0    0    0    5
 24      1     1       0    0    0    5
 25      1     8       0    9    0    0
 26      1     8       0    5    0    0
 27      1     5       0    9    0    0
 28      1     1       0    4    0    0
 29      1     5       0    4    0    0
 30      1     2       0    9    0    0
 31      1     1       0    0    0   10
 32      1     0       0    0    0    0
******************************************************************************
RESOURCEAVAILABILITIES:
  R 1  R 2  R 3  R 4
   11   11    9   11
******************************************************************************
```

Slika 5.1: Primjer datoteke s podacima za 30 aktivnosti

```
def read_txt_muli_lines(file_name, start_line, finish_line, del_num):
    """
    This method extract lines from start_line to finish_line from file_name
    removing new lines. From that, it then strips lines by spaces and removes
    all white spaces from them resulting in list with non white space characters.
    Returns list of lists with lines without white spaces and new line.
    """
    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
```

Slika 5.2: *read_text_multi_lines* funkcija

```
def read_txt_single_line(file_name, line):
    """
    This method reads line from file, strips it by new line
    and splits it by white spaces. Return list with line
    without white spaces and new 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
```

Slika 5.3: *read_txt_single_line* funkcija

Tijekom učitavanja podataka, informaciju o sljedbenicima spremali smo u rječnik *successor_dictionary* gdje je svaka aktivnost ključ, a pridružena joj je lista njezinih sljedbenika. Taj rječnik se kasnije koristi za dohvaćanje liste prethodnika željene aktivnosti (vidi Sliku 5.4). Lista prethodnika potrebna je prilikom računanja skupa dostupnih aktivnosti E_g u algoritmu usporedne sheme za generiranje rasporeda. Intuitivno, za prethodnike nulte aktivnosti dobivamo praznu listu.

```
19  def get_predecessors(successors_dictionary, val):
20
21      if val == 0:
22          return []
23
24      ret_lst = []
25      for key, value in successors_dictionary.items():
26          for el in value:
27              if val == el:
28                  ret_lst.append(key)
29                  continue
30
31      return ret_lst
```

Slika 5.4: Funkcija za dohvaćanje liste prethodnika

6 | Korisničko sučelje

6.1 Tkinter

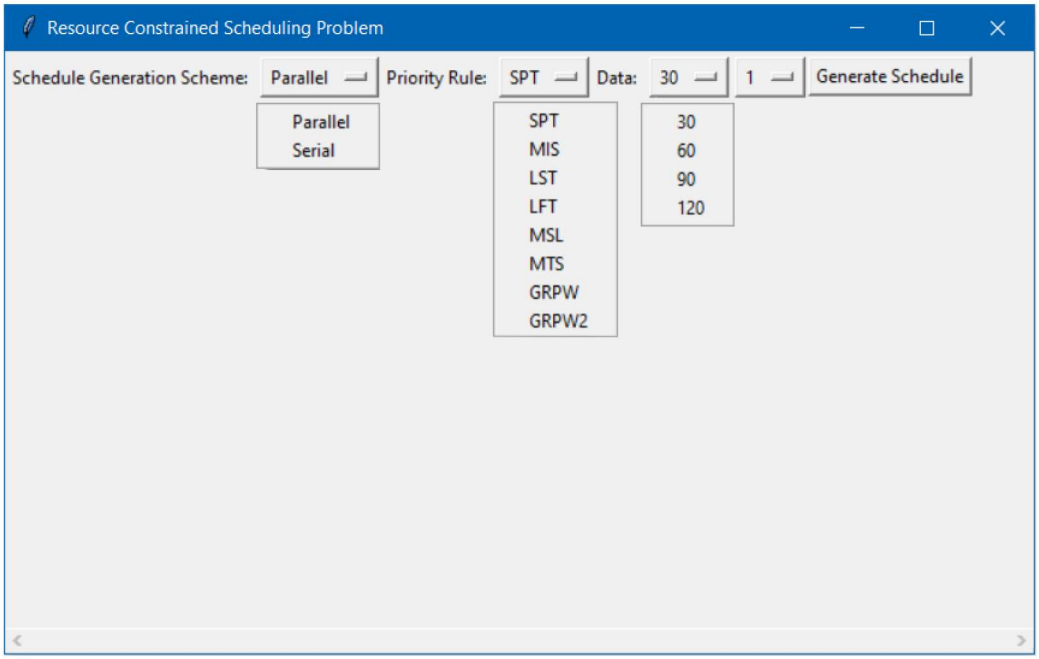
Vizualizaciju rasporeda odnosno kreiranje grafičkog sučelja postigli smo korištenjem Tkinter framework-a za Python GUI (engl. *Graphical User Interface*). Python u kombinaciji s Tkinter-om nudi brz i jednostavan način izrade korisničkih sučelja za aplikacije. Ono po čemu se Tkinter najviše ističe je njegova modularnost s manjim dijelovima programa. Svaki dio postoji kao zasebna komponenta koja se može prilagoditi. Ti dijelovi mogu se s lakoćom kombinirati kako bi se izgradio kompletan GUI. Neki od primjera za to su okviri, gumbi, oznake, platno, izbornik i slično. [8][9]

6.2 Generiranje rasporeda

Prikaz generiranog rasporeda dobivamo pokretanjem *gui.py* datoteke unutar terminala nakon čega se prikaže kreirano GUI sučelje. Iz padajućeg izbornika odabiremo željenu shemu, prioritetno pravilo, te skup podataka sa željenim brojem aktivnosti. Nakon toga pritiskom na gumb *„Generate Schedule“* raspored se generira u odnosu na prethodno odabrane stavke. Prikaz sučelja i navedenih mogućnosti možemo vidjeti na Slici 6.1.

Aktivnosti, njihovo trajanje i potražnju za sredstvima prikazali smo uz pomoć korištenja *broken_barh*() funkcije. Ona nam omogućuje iscrtavanje horizontalnog niza pravokutnika. Na x-osi nalazi se informacija o trajanju pojedine aktivnosti, a na y-osi kolika je potreba te aktivnosti za pojedinim sredstvom. Svaka aktivnost je drugačije boje što se može iščitati iz pripadne legende svakog rasporeda.

U slučaju kada imamo raspored sa 60 ili više aktivnosti, nije moguće odjednom ga cijelog prikazati na zaslonu. Zbog toga smo implementirali *„scrollbar“* mogućnost. Odnosno, moguće je pomicati se lijevo-desno po sučelju kako bi dobili pregled nad cijelim rasporedom i pripadnom legendom.



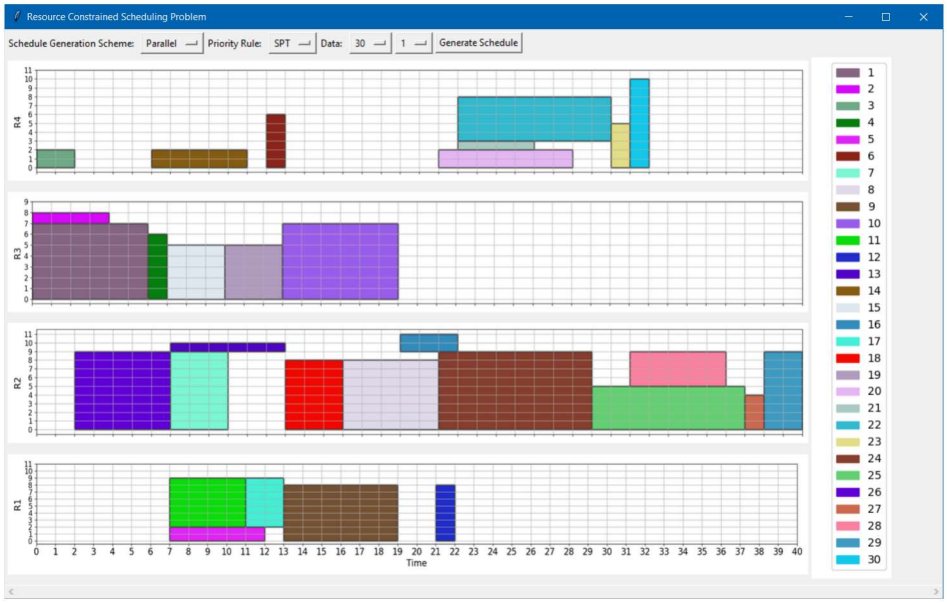
Slika 6.1: Prikaz izbornika unutar korisničkog sučelja

6.3 Rezultati

Pogledajmo kroz nekoliko primjera kako izgleda generirani raspored unutar korisničkog sučelja uz usporedni SGS i koje prioritetno pravilo daje najbolje pronađeno vrijeme završetka svakog od njih. Radi jednostavnosti, za svaki primjer uzet ćemo prvi skup podataka ponuđen u izborniku.

6.3.1 Raspored s 30 aktivnosti

S obzirom na odabrani skup podataka, prioritetno pravilo koje daje najranije vrijeme završetka u trenutku $t = 40$ za raspored s 30 aktivnosti prikazan na Slici 6.2 je SPT. Ostala pravila koja su dala približno dobar rezultat s vremenom završetka u trenutku $t = 41$ su MIS, LST, LFT, MTS i GRPW*. Najkasnije vrijeme završetka u ovom slučaju daje nam MSL. Točno vrijeme završetka nakon primjene svakog prioritetnog pravila na ovaj problem može se vidjeti u Tablici 6.1.



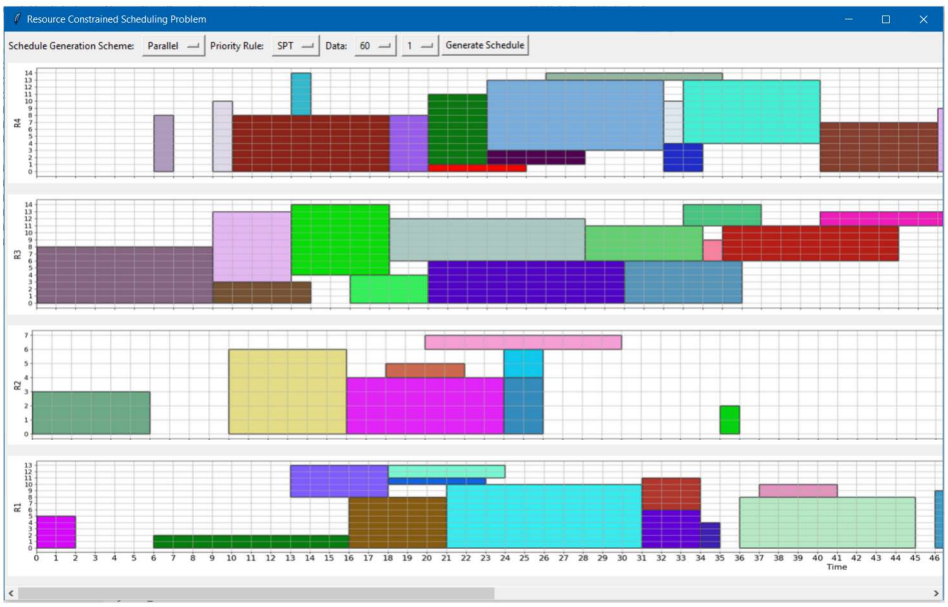
Slika 6.2: Raspored s 30 aktivnosti

30 aktivnosti	
Prioritetno pravilo	Vrijeme završetka
SPT	40
MIS	41
LST	41
LFT	41
MSL	50
MTS	41
GRPW	47
GRPW*	41

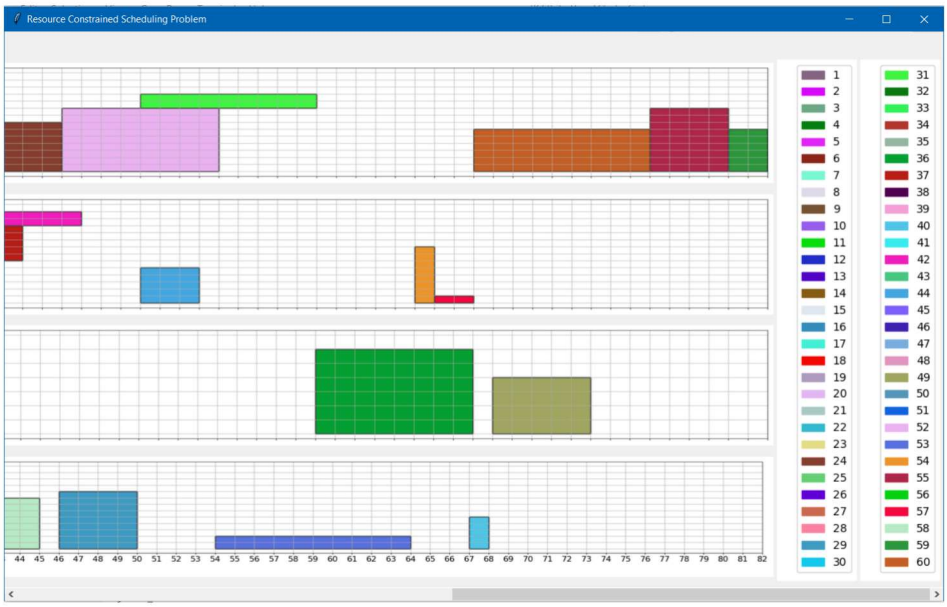
Tablica 6.1: Sva vremena završetka rasporeda s 30 aktivnosti

6.3.2 Raspored sa 60 aktivnosti

S obzirom na odabrani skup podataka, prioritetno pravilo koje daje najranije vrijeme završetka u trenutku $t = 82$ za raspored sa 60 aktivnosti prikazan na Slici 6.3 je SPT odnosno MIS. Najkasnije vrijeme završetka u ovom slučaju daje nam GRPW. Točno vrijeme završetka nakon primjene svakog prioritetnog pravila na ovaj problem može se vidjeti u Tablici 6.2.



(a) 1. dio



(b) 2. dio

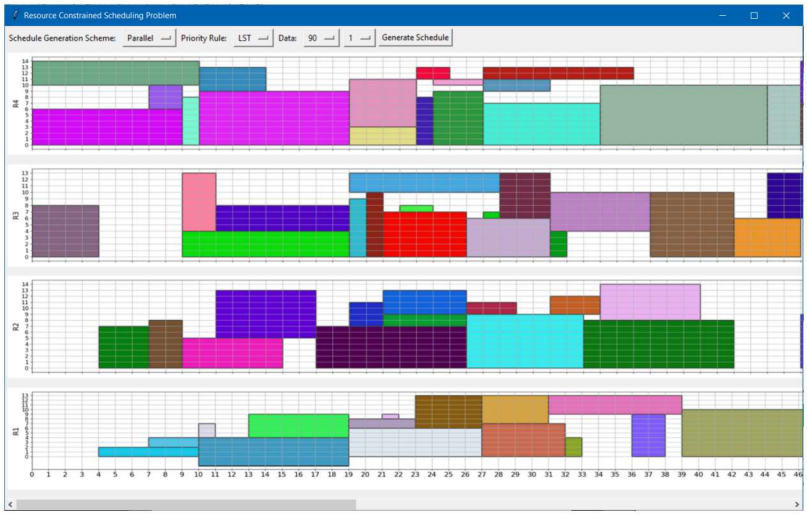
Slika 6.3: Raspored sa 60 aktivnosti

60 aktivnosti	
Prioritetno pravilo	Vrijeme završetka
SPT	82
MIS	82
LST	83
LFT	83
MSL	83
MTS	83
GRPW	88
GRPW*	83

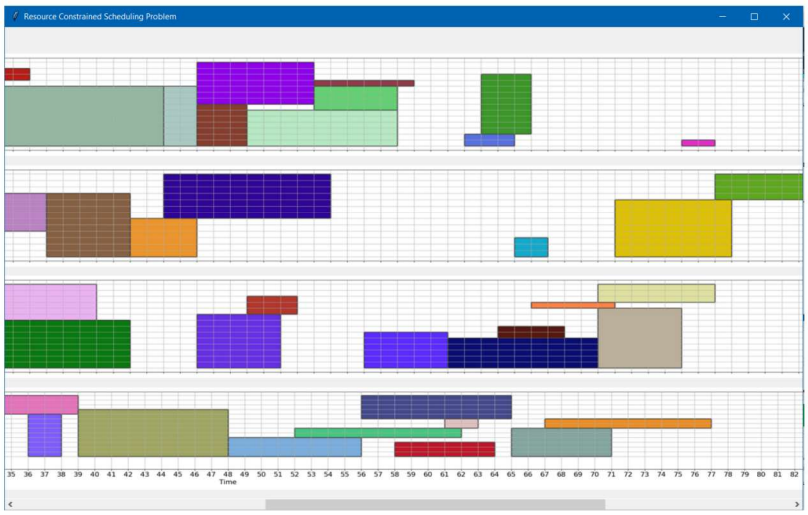
Tablica 6.2: Sva vremena završetka rasporeda sa 60 aktivnosti

6.3.3 Raspored s 90 aktivnosti

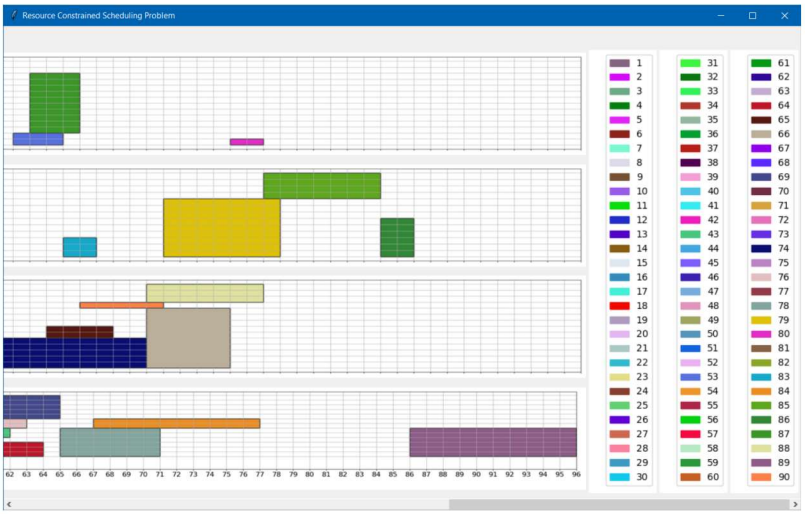
S obzirom na odabrani skup podataka, najranije vrijeme završetka za raspored s 90 aktivnosti prikazan na Slici 6.4 je u trenutku $t = 96$. Najlošije rezultate u ovom slučaju daju nam GRPW s $t = 116$ i SPT s $t = 110$. Točno vrijeme završetka nakon primjene svakog prioritetnog pravila na ovaj problem može se vidjeti u Tablici 6.3.



(a) 1. dio



(b) 2. dio



(c) 3. dio

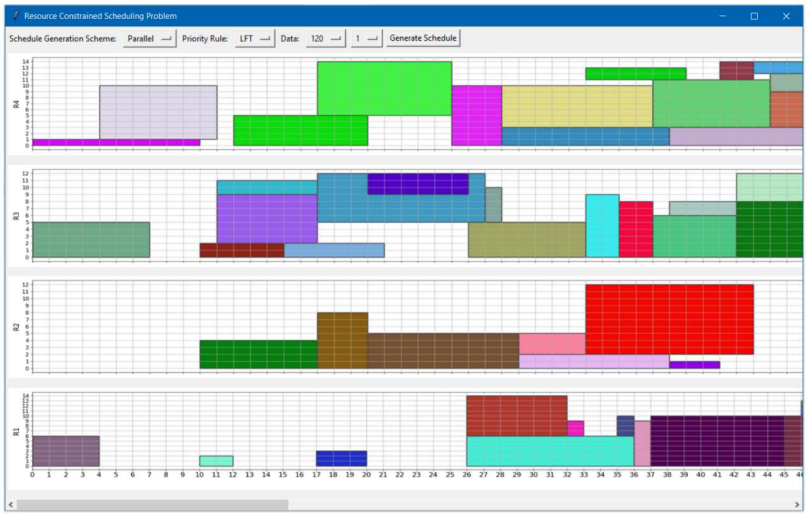
Slika 6.4: Raspored s 90 aktivnosti

90 aktivnosti	
Prioritetno pravilo	Vrijeme završetka
SPT	110
MIS	101
LST	96
LFT	96
MSL	96
MTS	96
GRPW	116
GRPW*	96

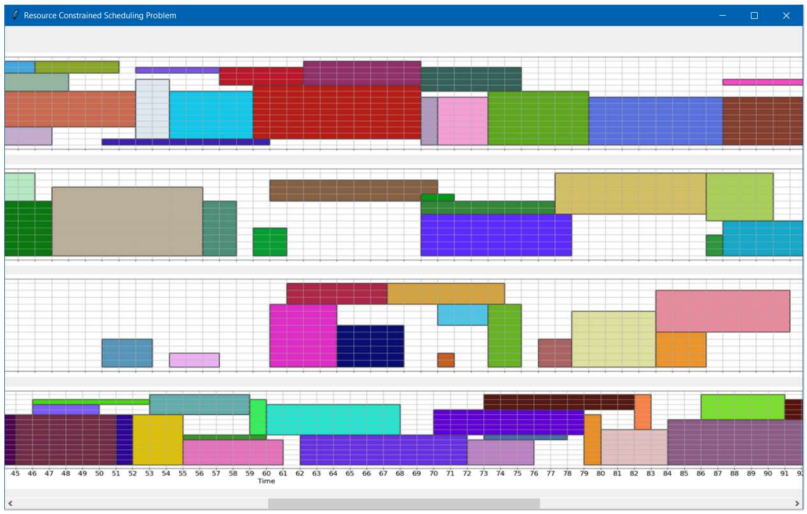
Tablica 6.3: Sva vremena završetka rasporeda s 90 aktivnosti

6.3.4 Raspored sa 120 aktivnosti

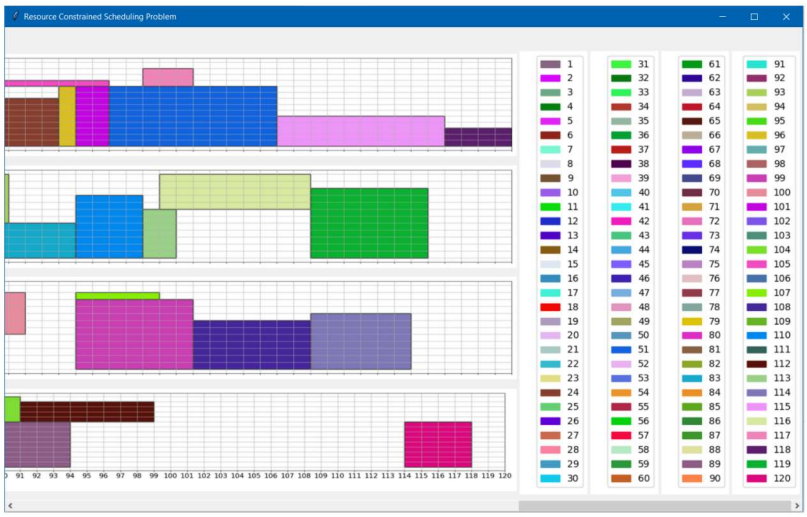
S obzirom na odabrani skup podataka, prioritetno pravilo koje daje najranije vrijeme završetka u trenutku $t = 120$ za raspored sa 120 aktivnosti prikazan na Slici 6.5 je LFT. Najkasnije vrijeme završetka u ovom slučaju daje nam GRPW u trenutku $t = 154$. Točno vrijeme završetka nakon primjene svakog prioritetnog pravila na ovaj problem može se vidjeti u Tablici 6.4.



(a) 1. dio



(b) 2. dio



(c) 3. dio

Slika 6.5: Raspored sa 120 aktivnosti

120 aktivnosti	
Prioritetno pravilo	Vrijeme završetka
SPT	144
MIS	132
LST	125
LFT	120
MSL	142
MTS	124
GRPW	154
GRPW*	124

Tablica 6.4: Sva vremena završetka rasporeda sa 120 aktivnosti

Popis slika

Popis tablica

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] Christian Artigues, Sophie Demassey, and Emmanuel Neron., *Resource constrained project scheduling: models, algorithms, extensions and applications.*, John Wiley & Sons, 2013.

[3] Robert Klein., *Scheduling of Resource-Constrained Projects*, Springer, 1999.

[4] Rainer Kolisch and Sönke Hartmann., *Heuristic algorithms for the resource constrained project scheduling problem: Classification and computational analysis.*, Springer, 1999.

[5] Rainer Kolisch., *Serial and Parallel Resource-Constrained Project Scheduling Methods Revisited: Theory and Computation.*, No. 344, Christian-Albrechts-University of Kiel, May 1994.

[6] Antonio Lova, Pilar Tormos, Federico Barber., *Multi-Mode Resource Constrained Project Scheduling: Scheduling Schemes, Priority Ruler and Mode Selection Rules*, Revista Iberoamericana de Inteligencia Artificial, vol. 10, num. 30, 2006., pp. 69-86

[7] Web izvor dostupan na http://www.om-db.wi.tum.de/psplib/library.html. (*PSBLIB biblioteka*).

[8] Web izvor dostupan na https://www.seeedstudio.com/blog/2021/07/19/what-is-python-gui-programming-uses-frameworks-tutorial/. (*Python GUI Programming: Uses, Frameworks & Tutorial*).

[9] Web izvor dostupan na https://www.tutorialspoint.com/python/python_gui_programming.htm. (*Python - GUI Programming (Tkinter)*).