Taula de continguts

Nom de l'activitat avaluable	% pes sobre la nota final	Evidència	Nota obtinguda	Pàgina
RS Manual	5%	Enunciat Rúbrica XR ML AT VA	- - 0 6,4 8,2 10	1 3 5 6 7 8
RS - Heuristic code	10%	Enunciat Rúbrica AG OA MK BB	- - 0 6,2 8,4 10	9 12 14 18 23 27
SA - Heuristic code	35%	Enunciat Rúbrica BE JP BB AD	- - 2,5 6,7 8 10	34 39 41 59 78 89

RS Manual

Enunciat

Activity 1 **RS** Manual

Solve the Runway Sequencing problem of your Toy model with the help of a spreadsheet and describe the rules followed.

Activity 1 **RS** Manual

Input data

Toy model

RS_**XX**.csv

	callsign	a_d	ac_model	time_atm
1:	TOY001	A	A380	10:00:25
2:	TOY002	D	B773	10:00:29
3:	TOY003	D	B752	10:00:40
4:	TOY004	A	TU22	10:01:34
5:	TOY005	D	A124	10:01:45

796:	TOY796	A	A124	20:40:30
797:	TOY797	D	B789	20:43:09
798:	TOY798	A	B788	20:44:46
799:	TOY799	D	B783	20:45:21
800:	TOY800	D	A380	20:47:36

First 20 departures

or

First 20 arrivals

+

AC types

ac_types.csv

+

WT TOY

WT_TOY.csv

Activity 1 **RS** Manual

Output data

Helping spreadsheet

rst_manual.xlsx

+

Result

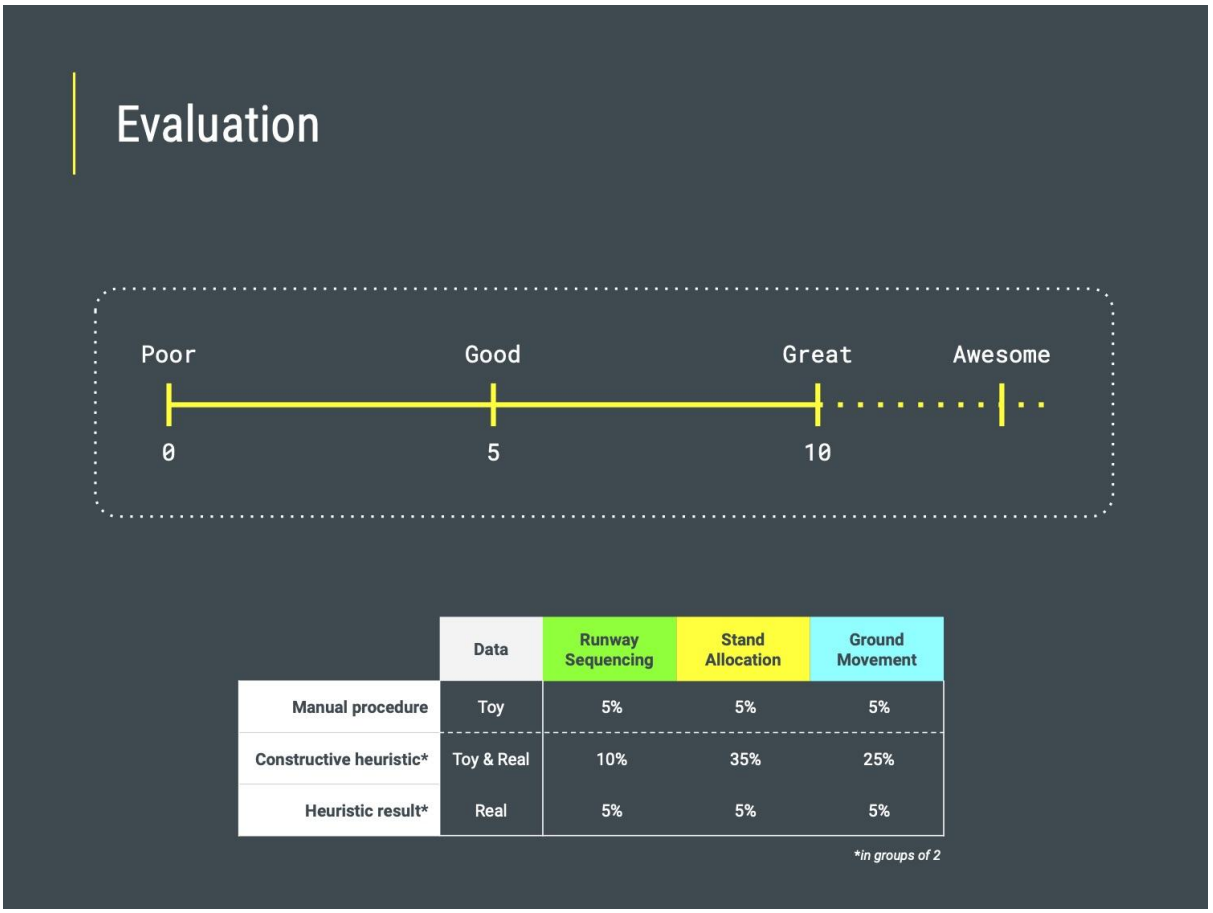
rst_result.csv

	callsign	ac_cat	time_atm
1:	TOY001	A	10:00:25
2:	TOY002	B	10:00:29
3:	TOY005	A	10:00:40
4:	TOY004	D	10:01:34
5:	TOY003	D	10:01:45

Evaluation: rst_eval_manual.csv

Rúbrica

El càlcul del delay del seu resultat és comparat amb els valors de la rúbrica. Qualsevol valor intermig tindrà una nota intermitja seguint una llei lineal.



Student	Group	File	A_D	Optim	Great	Good	Poor
1	1	RS_01	A	2799	3330	3861	4734
2	1	RS_01	D	2638	3186	3734	4636
3	2	RS_02	A	2191	2863	3534	4520
4	2	RS_02	D	2937	3554	4170	5076
5	3	RS_03	A	2304	2878	3451	4624
6	3	RS_03	D	2863	3181	3499	4508
7	4	RS_04	A	2300	2981	3661	4648
8	4	RS_04	D	2755	3337	3919	4809
9	5	RS_05	A	2883	3374	3865	4739
10	5	RS_05	D	2709	3154	3598	4693

11	6	RS_06	A	2672	3258	3844	4798
12	6	RS_06	D	2734	3358	3982	4859
13	7	RS_07	A	2709	3337	3965	4940
14	7	RS_07	D	2581	3080	3579	4413
15	8	RS_08	A	2333	2936	3539	4452
16	8	RS_08	D	2861	3295	3729	4667
17	9	RS_09	A	3172	3525	3878	4864
18	9	RS_09	D	2785	3367	3948	4907
19	10	RS_10	A	2980	3445	3910	4788
20	10	RS_10	D	2585	3189	3792	4947
21	11	RS_11	A	2642	3069	3496	4394
22	11	RS_11	D	2561	3211	3860	4803
23	12	RS_12	A	2736	3138	3539	4551
24	12	RS_12	D	2533	3096	3659	4653
25	13	RS_13	A	2471	3029	3586	4704
26	13	RS_13	D	2344	2938	3532	4560
27	14	RS_14	A	2865	3443	4021	5006
28	14	RS_14	D	2687	3151	3614	4578
29	15	RS_15	A	2030	2616	3202	4387
30	15	RS_15	D	2646	3098	3549	4403

Entregues

XR - 0

callsign	aircraft type	time_atm
TOY001	C	10:00:11
TOY004	C	10:00:45
TOY002	A	10:00:23
TOY006	B	10:02:49
TOY007	B	10:03:34
TOY010	A	10:06:31
TOY014	D	10:09:36
TOY016	A	10:12:20
TOY018	C	10:12:51
TOY017	A	10:12:43
TOY021	D	10:13:55
TOY023	A	10:15:31
TOY027	A	10:18:30
TOY028	B	10:18:46
TOY030	D	10:21:43
TOY031	B	10:23:12
TOY033	B	10:23:36
TOY035	D	10:26:54
TOY038	C	10:29:34

ML - 6,4

callsign	ac_type	time_atm
TOY001	B	10:00:10
TOY004	A	10:04:48
TOY008	C	10:06:56
TOY009	A	10:09:27
TOY011	C	10:09:57
TOY012	C	10:10:19
TOY013	D	10:11:10
TOY014	D	10:12:46
TOY016	D	10:13:38
TOY021	B	10:18:04
TOY024	C	10:22:04
TOY019	A	10:15:29
TOY027	A	10:24:28
TOY033	B	10:27:33
TOY031	B	10:27:02
TOY036	B	10:29:09
TOY028	D	10:25:16
TOY035	A	10:29:05
TOY038	A	10:31:31
TOY041	C	10:33:14

AT - 8,2

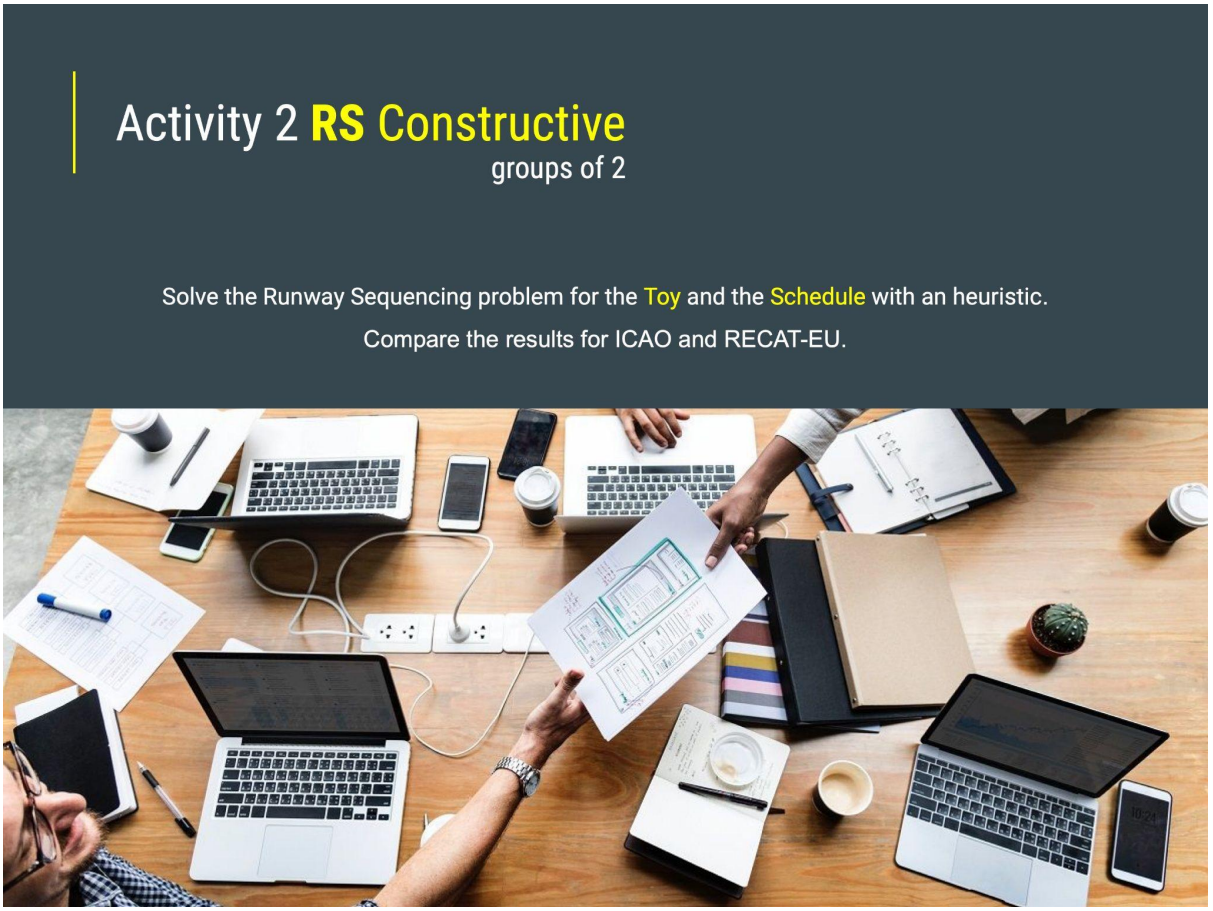
callsign	ac_cat	time_atm
TOY002	D	10:00:02
TOY001	A	10:01:14
TOY004	D	10:05:02
TOY006	C	10:06:22
TOY009	C	10:07:42
TOY013	C	10:09:02
TOY012	D	10:10:52
TOY010	B	10:12:02
TOY016	B	10:13:12
TOY011	A	10:14:24
TOY019	C	10:17:44
TOY022	B	10:18:54
TOY024	A	10:20:06
TOY026	A	10:21:42
TOY029	A	10:23:18
TOY028	B	10:26:19
TOY033	C	10:27:59
TOY031	D	10:29:49
TOY037	A	10:31:01
TOY039	B	10:34:02

VA - 10

callsign	ac_cat	time_atm
TOY004	C	10:01:34
TOY001	A	10:02:46
TOY007	A	10:04:22
TOY009	B	10:07:23
TOY011	A	10:08:36
TOY013	C	10:11:56
TOY018	D	10:13:46
TOY015	B	10:14:56
TOY017	A	10:16:08
TOY023	A	10:17:44
TOY019	C	10:21:04
TOY027	C	10:22:24
TOY031	C	10:23:44
TOY029	B	10:24:54
TOY025	B	10:26:04
TOY035	B	10:27:14
TOY032	D	10:29:24
TOY037	D	10:30:54
TOY039	D	10:32:24
TOY036	A	10:33:36

RS Constructive

Enunciat



Activity 2 **RS** Constructive

Input data

Toy model

+

AC types

+

WT TOY

Real schedule

```
     callsign airp a_d airl ac_model   tail time_sch time_atm
  1:  VLG181J EDDM   D  VLG     A320 EC-MBL 07:00:00 07:03:55
  2:  VLG142E LEBB   D  VLG     A320 EC-MNZ 07:00:00 07:05:59
  3:  VLG7010 LEMD   D  VLG     A320 EC-MVM 07:00:00 07:08:37
  4:  VLG34DP LIPZ   D  VLG     A321 EC-MHS 07:00:00 07:10:10
  5:  VLG108G LFPO   D  VLG     A20N EC-NCG 07:00:00 07:12:54
 ---
750:  EJU84CE LIMC   D  EZY     A319 OE-LQV 21:50:00 21:53:45
751:  VLG35AW LEIB   D  VLG     A320 EC-LVO 21:55:00 21:56:02
752:   WZZ172 LROP   D  WZZ     A321 HA-LXZ 21:55:00 21:57:28
753:   RYR2UG LEPA   A  RYR     B738 EI-EMA 22:00:00 21:46:41
754:  AEA6103 LEPA   D  AEA     B738 EC-ISN 22:00:00 22:00:00
```

+

AC types

+

WT RECAT-EU or WT ICAO

Activity 2 **RS** Constructive

Heuristic **specs**

```
rs_heur <- function(schedule,
                    sep_time = c("recat", "icao", "toy")){
  if(sep_time == "toy") {
    sch <- fread(paste0("rs_toys/", schedule))
  } else {
    sch <- fread(paste0("schedules/", schedule))
  }
  ac_types <- fread("data/ac_types.csv")
  if(sep_time == "recat") wt <- fread("data/WT_RECAT-EU.csv")
  if(sep_time == "icao")  wt <- fread("data/WT_ICAO.csv")
  if(sep_time == "toy")   wt <- fread("data/WT_TOY.csv")

  # your code

  return(solution)
}
```

up to **2 loops**

Solution: data.table with *callsign*, *ac_cat* and *time_atm*

Activity 2 **RS** Constructive

Output data

Heuristic

rs_heur.txt

+

Result

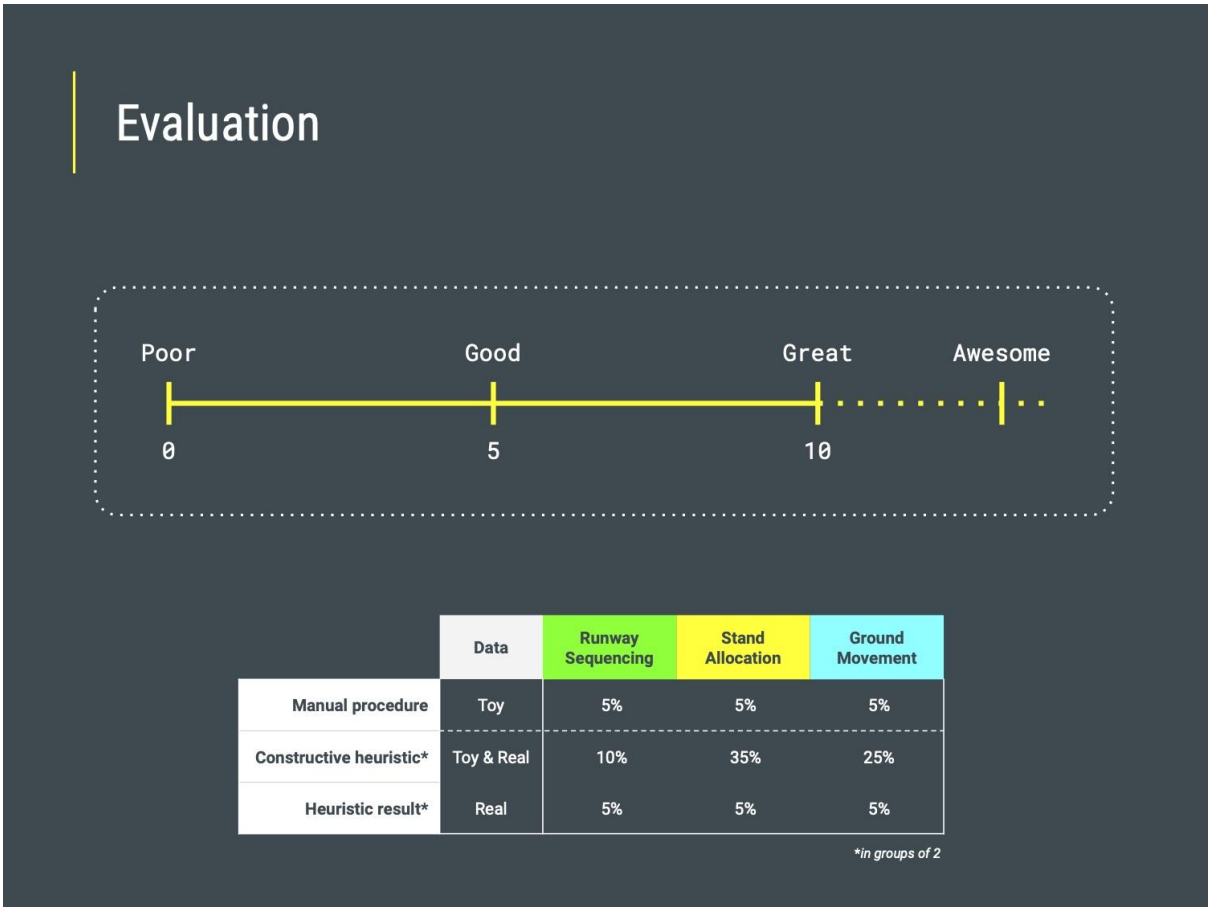
rs_icao.csv
rs_recat.csv

```
   callsign ac_cat time_atm
1:  VLG181J      A 10:00:25
2:  VLG142E      B 10:00:29
3:  VLG7010      A 10:00:40
4:  VLG34DP      D 10:01:34
5:  VLG108G      D 10:01:45
---
```

Evaluation: rst_eval_heuristic.csv

Rúbrica

El resultat de la seva funció és comparat amb els valors de la rúbrica. Qualsevol valor intermig tindrà una nota intermitja seguint una llei lineal. La funció es testeja amb varies dades externes i es fa la mitjana de la nota per cada base de dades.



File	Optim	Great	Good	Poor
RS_01	202015	503118	1205689	2445386
RS_02	192292	498991	1214620	2547357
RS_03	181809	517893	1302087	2567114
RS_04	181673	481012	1179468	2530736
RS_05	198667	507002	1226449	2653609
RS_06	182667	517323	1298184	2529490
RS_07	193316	455449	1067092	2545859
RS_08	194875	508011	1238660	2374813
RS_09	196214	581395	1480150	2733819
RS_10	196843	504205	1221381	2436048

RS_11	179554	425215	998423	2454425
RS_12	187816	451631	1067198	2534332
RS_13	189869	489811	1189674	2690474
RS_14	210406	578827	1438476	2836763
RS_15	189762	474294	1138202	2619081
RS_16	193704	537006	1338044	2819999
RS_17	188316	510779	1263190	2580985
RS_18	200234	472816	1108840	2440191
RS_19	189275	472103	1132033	2491958
RS_20	181126	510068	1277597	2520215

Entregues

AG - 0

```
install.packages("data.table")

library(data.table)
el_fold <- c("~/Desktop/RSHEUR/")
setwd(el_fold)

Allowed <- fread("allow_func.csv")

getDelay <- function(wtBo,Plan) {

  start=2
  Delay=Plan[,id]
  TipusCodi=Plan[,ac_type_code]
  Tipus=Plan[,ac_type]
  HoresPlan=Plan[,time_atm]
  HoresArriva=Plan[,time_atm]
  HoresPlan=strtoi(as.difftime(HoresPlan, format = "%H:%M:%S", units = "secs"))
  HoresArriva=strtoi(as.difftime(HoresPlan, format = "%H:%M:%S", units = "secs"))
  #lletra es follow
  Aux <- data.table (Value=1:(length(Delay)))
  Aux=Aux/Aux
  Aux1=Aux
  Aux2=Aux
  Aux1[length(Delay)]=0
  Aux2[1]=0
  Primer = TipusCodi*Aux1
  Segon = TipusCodi*Aux2
  Primer=Primer[!(apply(Primer, 1, function(y) any(y == 0))),]
  Segon=Segon[!(apply(Segon, 1, function(y) any(y == 0))),]

  PrimerSegon <- data.frame(Primer=Primer,
                            Segon=Segon)
  setnames(PrimerSegon, 'Value', 'x')
  setnames(PrimerSegon, 'Value.1', 'y')

  PrimerSegon$z <- wtBo[as.matrix(PrimerSegon)]
  DelayTotal=Plan[,ac_type_code]
  AltreAux <- data.table (Value=1:(length(HoresArriva)-1))
  AltreAux = AltreAux[,Value]
  Delaytotal=0
  DelayMinim = PrimerSegon[,3]
  Delaylist = DelayMinim
  for (i in AltreAux) {
    Delaylist[i] = (HoresPlan[i+1]-HoresArriva[i])
    if (Delaylist[i] >= DelayMinim[i]){
      HoresArriva[i+1] = HoresPlan[i+1]
```

```
    } else{
      HoresArriva[i+1] = HoresArriva[i]+DelayMinim[i]
    }
    Delaytotal=Delaytotal+HoresArriva[i+1]-HoresPlan[i+1]
    DelayTotal[i+1]=Delaytotal
  }


  Plan$HoresArriva <- HoresArriva
  Plan$HoresPlan <- HoresPlan
  Plan$DelayTotal <- DelayTotal
  Delay[1]=0
  ForceDelay <- data.table(Value=2:length(Delay))
  return(Plan)
}

acondicionAvions <- function(SCH, ac_types) {
  ac_typematcher=SCH[,.(id,ac_model)]
  ac_aux=SCH[,ac_model]

  A=ac_types[,A]
  B=ac_types[,B]
  C=ac_types[,C]
  D=ac_types[,D]
  E=ac_types[,E]
  F=ac_types[,F]

  AvionsA=match(ac_aux,A,nomatch = 0, incomparables = NULL)
  AvionsB=match(ac_aux,B,nomatch = 0, incomparables = NULL)
  AvionsC=match(ac_aux,C,nomatch = 0, incomparables = NULL)
  AvionsD=match(ac_aux,D,nomatch = 0, incomparables = NULL)
  AvionsE=match(ac_aux,E,nomatch = 0, incomparables = NULL)
  AvionsF=match(ac_aux,F,nomatch = 0, incomparables = NULL)

  AvionsA=round(AvionsA/(AvionsA+0.5))
  AvionsB=round(AvionsB/(AvionsB+0.5))*2
  AvionsC=round(AvionsC/(AvionsC+0.5))*3
  AvionsD=round(AvionsD/(AvionsD+0.5))*4
  AvionsE=round(AvionsE/(AvionsE+0.5))*5
  AvionsF=round(AvionsF/(AvionsF+0.5))*6

  Avions2 <- AvionsA+AvionsB+AvionsC+AvionsD+AvionsE+AvionsF
  Avions <- AvionsA+AvionsB+AvionsC+AvionsD+AvionsE+AvionsF

  Avions=gsub("1", "A", Avions, ignore.case = FALSE, perl = FALSE,
              fixed = FALSE, useBytes = FALSE)
  Avions=gsub("2", "B", Avions, ignore.case = FALSE, perl = FALSE,
              fixed = FALSE, useBytes = FALSE)
  Avions=gsub("3", "C", Avions, ignore.case = FALSE, perl = FALSE,
              fixed = FALSE, useBytes = FALSE)
  Avions=gsub("4", "D", Avions, ignore.case = FALSE, perl = FALSE,
```

```
              fixed = FALSE, useBytes = FALSE)
  Avions=gsub("5", "E", Avions, ignore.case = FALSE, perl = FALSE,
              fixed = FALSE, useBytes = FALSE)
  Avions=gsub("6", "F", Avions, ignore.case = FALSE, perl = FALSE,
              fixed = FALSE, useBytes = FALSE)


  SCH <- cbind(ac_type=Avions, SCH)
  SCH <- cbind(ac_type_code=Avions2, SCH)
  return(SCH)
}

rs_heur <- function(schedule,
                    sep_time = c("recat", "icao", "toy")){
  if(sep_time == "toy") {
    sch <- fread(paste0("RS_02.csv"))
  } else {
    sch <- fread(paste0("SCH_02.csv"))
  }
  ac_types <- fread("ac_types.csv")
  if(sep_time == "recat") {
    wt <- fread("WT_RECAT-EU.csv")
  }
  if(sep_time == "icao") {
    wt <- fread("WT_ICAO.csv")
  }
  if(sep_time == "toy") {
    wt <- fread("WT_TOY.csv")
  }

  # Time to create tables
  Original <- rownames(sch)
  SCH <- cbind(id=Original, sch)
  SCH$id <- as.numeric(as.character(SCH$id))

  SCH=acondicionAvions(SCH,ac_types)


  SCH_reduced=SCH[,.(id,ac_type,ac_type_code,a_d,time_sch,time_atm)]

  wtA=wt[,A]
  wtB=wt[,B]
  wtC=wt[,C]
  wtD=wt[,D]
  wtE=wt[,E]
  wtF=wt[,F]
  wtBo=wt[,.(A,B,C,D,E,F)]
  wtBo=as.matrix(wtBo)

  HoresOrdre=SCH[,time_atm]
  HoresOrdre=strtoi(as.difftime(HoresPlan, format = "%H:%M:%S", units = "secs"))
```

```
  SCH$HoresOrdre <- HoresOrdre
  #Preparem Delayer
  SCH_FIFO=SCH[order(HoresOrdre)]
  Delay=getDelay(wtBo,SCH_FIFO)


  Arrivals <- SCH_FIFO[a_d=="A"]
  Departures <- SCH_FIFO[a_d=="D"]



  DelayArrivals=getDelay(wtBo,Arrivals)
  DelayDepartures=getDelay(wtBo,Departures)


   # your code



  return(solution)
}
```

OA - 6,2

```
rs_heur <- function(schedule, sep_time = c("recat", "icao", "toy")){

  if(sep_time == "toy"){
    sch <- fread(paste0("RS_", schedule,".csv"))
  }else {
    sch <- fread(paste0("SCH_", schedule,".csv"))
  }
  ac_types <- fread("ac_types.csv")
  if(sep_time == "recat") wt <- fread("WT_RECAT-EU.csv")
  if(sep_time == "icao")  wt <- fread("WT_ICAO.csv")
  if(sep_time == "toy")  wt <- fread("WT_TOY.csv")


  #library("data.table")


  ### First we preprocess the data: ###


  #1. We relate each aircraft with its type (A,B,C,D,E,F)

  ac_types[,x:=seq(.N)]
  ac_types<-melt(ac_types,"x")
  ac_types<-ac_types[value!=""]
  ac_types[,x:=NULL]
  names(ac_types)<-c("ac_type","ac_model")
  sch<-merge(sch,ac_types,by="ac_model",sort=FALSE)
  sch<-sch[,ac_model:=NULL]
  sch[,time_atm:=as.ITime(time_atm)]


  #2. We distinguish between Departures (D) and arrivals (A)

  departures <- sch[a_d=="D"]
  departures[,a_d:=NULL]
  arrivals <- sch[a_d=="A"]
  arrivals[,a_d:=NULL]


  #3. We prepare the WT Separation Time data in order to be processed

  wt<-melt(wt, "Lead/Follow")
  names(wt)<-c("leading","following","separation_time")



  ### Secondly we prepare special functions in order to compute the separation time
and the actual landing/departure times
```

```
  #4. Function "separation_time_computation" to get the separation times induced by
the leading/following WT Aircraaft method

  separation_time_computation<-function(f){

    f[,leading:=ac_type]
    f$leading[2:nrow(f)]<-f$leading[1:nrow(f)-1]
    f[,following:=ac_type]
    f<-merge(f, wt, by = c("leading","following"),sort=FALSE)
    f$separation_time[1]<-0
    f[,separation_time:=as.ITime(separation_time)]

    return(f)

  }

  arrivals<- separation_time_computation(arrivals)
  departures<- separation_time_computation(departures)



  #5. Function "actual" to get actual landing/departure times

  actual<-function(f){

    f[,time_act:=time_atm]

    for(i in 2:nrow(f)){
      if(f$time_act[i-1]+f$separation_time[i]>f$time_atm[i]){
        f$time_act[i]<-f$time_act[i-1]+f$separation_time[i]
      }
    }
    return(f)
  }

  arrivals<-actual(arrivals)
  departures<-actual(departures)



  ### Thirdly we proceed to do the total delay computation (arrivals +
departures)for the FIFO case in order to verify the coding


  delay_computation<-function(f){
    f[,delay:=time_act-time_atm]
    return(f)}
```

```
total_delay<-function(f){
  delay_computation(f)
  return(sum(f$delay))}

arrivals<-delay_computation(arrivals)
departures<-delay_computation(departures)
delay_FIFO<-total_delay(arrivals)+total_delay(departures)
#print("The FIFO accumulated delay for Arrivals+Departures is [seconds]:")
#show(delay_FIFO)



### Finally we take into account the method in order to reduce the delay. That
will be to displace each flight into 2 positions, that is because in most of the
times, displacing
### a flight more positions does not help reducing the delay, so we decided to
"optimize" the switched version by displacing the flights 2 rows each and
recomputing the
### obtained delay to see if that positioning makes the delay lower. If not, the
original position is maintained.



delay_recomputing <- function(f){
  f[,separation_time:=NULL]
  f[,time_act:=NULL]
  f[,delay:=NULL]
  f<-separation_time_computation(f)
  f<-actual(f)
  return(f)
}

rows_movement<-function(y,row1,row2){
  y[,numrow:=.I]
  y$numrow[row1]=row2
  y$numrow[row2]=row1
  y<-y[order(numrow)]
  y[,numrow:=NULL]
  return(y)
}

flightorder<-function(f,flight_1,flight_2){

  f<-rows_movement(f,flight_1,flight_2)
  f<-delay_recomputing(f)
  f<-delay_computation(f)
  return(f)
}

SWITCH_2<-function(f){
  for(i in 1:nrow(f)-2){
```

```
      test_position <- flightorder(f,i,i+2)
      time_delay <- total_delay(f)

      if(total_delay(test_position)<time_delay)
      {
        f<-test_position
      }

    }
    return(f)
  }

  arrivals_optimous<-SWITCH_2(arrivals)
  departures_optimous<-SWITCH_2(departures)

  delay_optimous<-total_delay(arrivals_optimous)+total_delay(departures_optimous)
  #print("The Optimsed accumulated delay for Arrivals+Departures is [seconds]:")
  #show(delay_optimous)

  #We prepare the data in order to export in .csv files (callsign, ac_cat,
time_atm)

  arrivals_optimous[,leading:=NULL]
  arrivals_optimous[,following:=NULL]
  arrivals_optimous[,separation_time:=NULL]
  arrivals_optimous[,time_act:=NULL]
  arrivals_optimous[,delay:=NULL]
  arrivals_optimous[,numrow:=NULL]
  arrivals_optimous[,airp:=NULL]
  arrivals_optimous[,airl:=NULL]
  arrivals_optimous[,tail:=NULL]
  arrivals_optimous[,time_sch:=NULL]
  setcolorder(arrivals_optimous,c("callsign","ac_type","time_atm"))

  departures_optimous[,leading:=NULL]
  departures_optimous[,following:=NULL]
  departures_optimous[,separation_time:=NULL]
  departures_optimous[,time_act:=NULL]
  departures_optimous[,delay:=NULL]
  departures_optimous[,numrow:=NULL]
  departures_optimous[,airp:=NULL]
  departures_optimous[,airl:=NULL]
  departures_optimous[,tail:=NULL]
  departures_optimous[,time_sch:=NULL]
  setcolorder(departures_optimous,c("callsign","ac_type","time_atm"))

  #S'adjunta un mateix CSV on s'inclouen tant arrivals com departures (primer les
arrivals i després les departures)

  Total_Order=rbind(arrivals_optimous, departures_optimous)
  solution<-data.table(Total_Order)
```

```
  return(solution)

}
```

MK - 8,4

```
rs_heur <- function(schedule,sep_time = c("recat", "icao", "toy")){
  if(sep_time == "toy") {
    sch <- fread(paste0("RS_", schedule,".csv"))
  } else {
    sch <- fread(paste0("SCH_", schedule,".csv"))
  }
  ac_types <- fread("data/ac_types.csv")
  if(sep_time == "recat") wt <- fread("data/WT_RECAT-EU.csv")
  if(sep_time == "icao") wt <- fread("data/WT_ICAO.csv")
  if(sep_time == "toy") wt <- fread("data/WT_TOY.csv")


  #Create ac_cat column as a function of ac_model and ac_types columns
  sch[ac_model%in%t(ac_types[,1]), `:=`(ac_cat =names(ac_types)[1])]
  sch[ac_model%in%t(ac_types[,2]), ac_cat := names(ac_types)[2]]
  sch[ac_model%in%t(ac_types[,3]), ac_cat := names(ac_types)[3]]
  sch[ac_model%in%t(ac_types[,4]), ac_cat := names(ac_types)[4]]
  sch[ac_model%in%t(ac_types[,5]), ac_cat := names(ac_types)[5]]
  sch[ac_model%in%t(ac_types[,6]), ac_cat := names(ac_types)[6]]

  #Time PT as a function of time_atm, in seconds

hours=3600*as.numeric(paste0(tstrsplit(sch$time_atm,"")[[1]],tstrsplit(sch$time_atm
,"")[[2]]))

minutes=60*as.numeric(paste0(tstrsplit(sch$time_atm,"")[[4]],tstrsplit(sch$time_atm
,"")[[5]]))

seconds=as.numeric(paste0(tstrsplit(sch$time_atm,"")[[7]],tstrsplit(sch$time_atm,""
)[[8]]))

  #Total sum of the time terms
  sch[, `:=`(PT =hours+minutes+seconds)]

  sch[, `:=`(AT=1.0E10)] # We create AT column
  sch[, `:=`(delay=0)] # We create delay column

  max_iterations=60 #For each package
  package_size=5 # Number of flights which will be shuffled
  #The optimal combination of max_iterations-package_size has been empirically
derived,
  #taking into account the total computing time

  arrivals_departures=c("A","D")
  count=1
  while(count<=length(arrivals_departures)){ #For arrivals and departures

    sch_eval=sch[a_d==arrivals_departures[count]]
```

```
    leftover_row=1+nrow(sch_eval)-nrow(sch_eval)%%package_size #varible to control
fligths which don't fit in the last package
    contador_paquetes=0
    iterations=1

    i=1
    while (i <=nrow(sch_eval)+1){

      if((i-1)%%package_size==0){
        if((iterations<=max_iterations)&(contador_paquetes!=0)){

if(i==nrow(sch_eval)&(contador_paquetes==ceiling(nrow(sch_eval)/package_size))){
              i=leftover_row
          }else{
            i=i-package_size
          }
        }else{
          contador_paquetes=contador_paquetes+1
          iterations=1
          best_delay=1E10
        }
        if(i<(nrow(sch_eval)+1)){
          if(i==leftover_row){
            if(i==nrow(sch_eval)){
              newOrder=i
            }
            else{
              newOrder=sample(c(i:nrow(sch_eval)))
            }
          }else{
            newOrder=sample(c((i):(i+package_size-1)))
          }
        }
      }

      if(i<(nrow(sch_eval)+1)){
        if(i!=1){
          i_local=i-(package_size*(contador_paquetes-1))
        } else{
          i_local=i
        }

        #turbulent wake assignment in AT column as a function of wt
        if(i_local!=1){
          leader=sch_eval[newOrder]$ac_cat[i_local-1]
        } else{
          if(i!=1){
            leader=sch_eval$ac_cat[i-1]
          }
        }
```

```
        if(i!=1){

          follow=sch_eval[newOrder]$ac_cat[i_local]
          at=wt[wt[,`Lead/Follow`]==leader,..follow]

          #The AT has to be > PT
          if(i_local==1){
            if (sch_eval[newOrder]$PT[i_local]>=at+sch_eval[order(AT)]$AT[i-1]){
              sch_eval[(newOrder[i_local]), AT :=sch_eval[newOrder]$PT[i_local]]
            }else{
              sch_eval[(newOrder[i_local]), AT :=at+sch_eval[order(AT)]$AT[i-1]]
            }
          } else{
            if
(sch_eval[newOrder]$PT[i_local]>=at+sch_eval[newOrder]$AT[i_local-1]){
              sch_eval[(newOrder[i_local]), AT :=sch_eval[newOrder]$PT[i_local]]
            }
            else{
              sch_eval[(newOrder[i_local]), AT
:=at+sch_eval[newOrder]$AT[i_local-1]]
            }
          }

        }
        if(i==1){
          sch_eval[(newOrder[i_local]), AT :=sch_eval[newOrder]$PT[i_local]]
        }

sch_eval$delay[(newOrder[i_local])]=sch_eval[newOrder]$AT[i_local]-sch_eval[newOrde
r]$PT[i_local]

        if ((i_local==package_size)|(i==nrow(sch_eval))){

          new_delay=sum(sch_eval[order(AT)]$delay)

          if(new_delay<best_delay){
            best_sch=copy(sch_eval[order(AT)])
            best_delay=new_delay
          }else{
            sch_eval=copy(best_sch)
          }

          iterations=iterations+1
        }
      }
      i=i+1
   }
    if(count==1){
      winner_sch=best_sch
    }else{
      winner_sch=rbind(winner_sch,best_sch)
```

```
    }
    count=count+1
  }

  #Time AT to atm
  winner_sch[, `:=`(AT_atm =as.ITime(winner_sch$AT, format = "%H:%M:%S"))]
  setcolorder(winner_sch, c("callsign", "ac_cat", "AT_atm"))
  solution=winner_sch[,1:3]
  setnames(solution,c("callsign","ac_cat","time_atm"))
 return(solution)
}
```

BB - 10

```
rs_heur <- function(schedule, sep_time = c("recat","icao","toy")) {

  #library(data.table)
  if (sep_time == "toy") {
    sch <- fread(paste0("rs_toys/",schedule))
  } else {
    sch <- fread(paste0("schedules/",schedule))
  }

  ac_types <- fread("data/ac_types.csv")

  if (sep_time=="recat") {
    wt <- fread("data/WT_RECAT-EU.csv")
  }
  if (sep_time=="icao") {
    wt <- fread("data/WT_ICAO.csv")
  }
  if (sep_time=="toy") {
    wt <- fread("data/WT_TOY.csv")
  }

  #### Functions' definition
  calculate_act <- function(plt, sep_min, act_old) {
    # Function that calculates the act time
    # Inputs: plt, sep_min and previous act
    # FIRST FOR USED!
    i <- 1
    act <- plt*0
    for (x in plt) {
      if (act_old[i] + sep_min[i] > plt[i]) {
        act[i] <- act_old[i] + sep_min[i]
      } else {
        act[i] <- plt[i]
      }
      act_old[i+1] <- act[i]
      i <- i + 1
    }
    return(act)
  }

  calculate_delay <- function(sch_work_partial, wt_vector, last_letter, last_act){
    # Function that calculates values act and delay
    # Given any given sequence of flights no matter its order
    # This function only works with the fraction of the whole data.table sch_work

    sch_c <- sch_work_partial [, .(order_orig, callsign, a_d, ac_model, ac_cat,
time_atm, plt)]

    sch_c$order <- seq(from=1, to=length(sch_c$callsign))
```

```
    n <- length(sch_c$callsign)
    sch_c$lead <- c(last_letter, as.character(sch_c$ac_cat[1:n-1]))

    sch_c$lead_follow <- paste0(sch_c$lead, sch_c$ac_cat)

    sch_c <- merge(sch_c, wt_vector, by = "lead_follow")
    sch_c <- sch_c [ order( sch_c$order) , ]

    sch_c$act <- sch_c$plt
    sch_c$act <- calculate_act(sch_c$plt, sch_c$sep_min, c( last_act, sch_c$act))
    sch_c$delay <- sch_c$act - sch_c$plt

    sch_c <- sch_c [, .(order_orig, callsign, a_d, ac_model, time_atm, ac_cat,
lead, lead_follow, plt, sep_min, act, delay)]

    return(sch_c)

  }

  optimize_delay <- function(sch_work, combis4) {
    # Functions that gets the combination of flights that provides
    # the minimum total delay

    # Preallocation of the variables and ensure match between sch_work and sch_cx
    sch_work$lead <- sch_work$ac_cat
    sch_work$lead_follow <- sch_work$ac_cat
    sch_work$plt <- as.integer(as.ITime(sch_work$time_atm))
    sch_work$sep_min <- sch_work$plt*0
    sch_work$act <- sch_work$plt
    sch_work$delay <- sch_work$plt*0

    # A column with the original order in order to check the solution
    sch_work[, order_orig := seq(.N)]

    # We order data.table sch_work
    sch_work <- sch_work [, .(order_orig, callsign, a_d, ac_model, time_atm,
ac_cat, lead, lead_follow, plt, sep_min, act, delay)]

    end <- length(sch_work$callsign)
    i <- 1L
    p <- 1L
    loadings = c(10,20,30,40,50,60,70,80,90,100)
    loadings_check = rep(FALSE,length(loadings))

    # SECOND FOR USED!
    for (x in sch_work$callsign) {
      # All combinations
      c1 <- combis4[1,]+i-1
      c2 <- combis4[2,]+i-1
      c3 <- combis4[3,]+i-1
```

```
      c4 <- combis4[4,]+i-1
      c5 <- combis4[5,]+i-1
      c6 <- combis4[6,]+i-1
      c7 <- combis4[7,]+i-1
      c8 <- combis4[8,]+i-1
      c9 <- combis4[9,]+i-1
      c10 <- combis4[10,]+i-1
      c11 <- combis4[11,]+i-1
      c12 <- combis4[12,]+i-1
      c13 <- combis4[13,]+i-1
      c14 <- combis4[14,]+i-1
      c15 <- combis4[15,]+i-1
      c16 <- combis4[16,]+i-1
      c17 <- combis4[17,]+i-1
      c18 <- combis4[18,]+i-1
      c19 <- combis4[19,]+i-1
      c20 <- combis4[20,]+i-1
      c21 <- combis4[21,]+i-1
      c22 <- combis4[22,]+i-1
      c23 <- combis4[23,]+i-1
      c24 <- combis4[24,]+i-1

      # Information of the last flight established
      # - Category and act of the flight
      if (i == 1) {
        last_letter <- "N"
        last_act <- sch_work$plt[1]
      } else {
        last_letter <- as.character(sch_work$ac_cat[i-1])
        last_act <- sch_work$act[i-1]
      }

      # Calculating delay for all possible combinations
      # Note: This long and repetitive list of statements could be simplified
      # using a for and "dynamic programming" (like eval in MATLAB or Python)
      sch_c1 <- calculate_delay( sch_work [c1,], wt_vector, last_letter, last_act)
      sch_c2 <- calculate_delay( sch_work [c2,], wt_vector, last_letter, last_act)
      sch_c3 <- calculate_delay( sch_work [c3,], wt_vector, last_letter, last_act)
      sch_c4 <- calculate_delay( sch_work [c4,], wt_vector, last_letter, last_act)
      sch_c5 <- calculate_delay( sch_work [c5,], wt_vector, last_letter, last_act)
      sch_c6 <- calculate_delay( sch_work [c6,], wt_vector, last_letter, last_act)
      sch_c7 <- calculate_delay( sch_work [c7,], wt_vector, last_letter, last_act)
      sch_c8 <- calculate_delay( sch_work [c8,], wt_vector, last_letter, last_act)
      sch_c9 <- calculate_delay( sch_work [c9,], wt_vector, last_letter, last_act)
      sch_c10 <- calculate_delay( sch_work [c10,], wt_vector, last_letter,
last_act)
      sch_c11 <- calculate_delay( sch_work [c11,], wt_vector, last_letter,
last_act)
      sch_c12 <- calculate_delay( sch_work [c12,], wt_vector, last_letter,
last_act)
```

```
      sch_c13 <- calculate_delay( sch_work [c13,], wt_vector, last_letter,
last_act)
      sch_c14 <- calculate_delay( sch_work [c14,], wt_vector, last_letter,
last_act)
      sch_c15 <- calculate_delay( sch_work [c15,], wt_vector, last_letter,
last_act)
      sch_c16 <- calculate_delay( sch_work [c16,], wt_vector, last_letter,
last_act)
      sch_c17 <- calculate_delay( sch_work [c17,], wt_vector, last_letter,
last_act)
      sch_c18 <- calculate_delay( sch_work [c18,], wt_vector, last_letter,
last_act)
      sch_c19 <- calculate_delay( sch_work [c19,], wt_vector, last_letter,
last_act)
      sch_c20 <- calculate_delay( sch_work [c20,], wt_vector, last_letter,
last_act)
      sch_c21 <- calculate_delay( sch_work [c21,], wt_vector, last_letter,
last_act)
      sch_c22 <- calculate_delay( sch_work [c22,], wt_vector, last_letter,
last_act)
      sch_c23 <- calculate_delay( sch_work [c23,], wt_vector, last_letter,
last_act)
      sch_c24 <- calculate_delay( sch_work [c24,], wt_vector, last_letter,
last_act)

      # Vector with all the total delays of the 24 combinations
      delays <-
c(sum(sch_c1$delay),sum(sch_c2$delay),sum(sch_c3$delay),sum(sch_c4$delay),

sum(sch_c5$delay),sum(sch_c6$delay),sum(sch_c7$delay),sum(sch_c8$delay),

sum(sch_c9$delay),sum(sch_c10$delay),sum(sch_c11$delay),sum(sch_c12$delay),

sum(sch_c13$delay),sum(sch_c14$delay),sum(sch_c15$delay),sum(sch_c16$delay),

sum(sch_c17$delay),sum(sch_c18$delay),sum(sch_c19$delay),sum(sch_c20$delay),

sum(sch_c21$delay),sum(sch_c22$delay),sum(sch_c23$delay),sum(sch_c24$delay)
      )

      # Get the combination that provides the minimum delay
      I <- match( min(delays), delays )

      # Save the order of the combination in the big data.table sch_work
      vec <- c(i, i+1, i+2, i+3)
      if (I==1) {
        sch_work[vec,] <- sch_c1
      } else { if (I==2) {
        sch_work[vec,] <- sch_c2
      } else { if (I==3) {
        sch_work[vec,] <- sch_c3
```

```
} else { if (I==4) {
  sch_work[vec,] <- sch_c4
} else { if (I==5) {
  sch_work[vec,] <- sch_c5
} else { if (I==6) {
  sch_work[vec,] <- sch_c6
} else { if (I==7) {
  sch_work[vec,] <- sch_c7
} else { if (I==8) {
  sch_work[vec,] <- sch_c8
} else { if (I==9) {
  sch_work[vec,] <- sch_c9
} else { if (I==10) {
  sch_work[vec,] <- sch_c10
} else { if (I==11) {
  sch_work[vec,] <- sch_c11
} else { if (I==12) {
  sch_work[vec,] <- sch_c12
} else { if (I==13) {
  sch_work[vec,] <- sch_c13
} else { if (I==14) {
  sch_work[vec,] <- sch_c14
} else { if (I==15) {
  sch_work[vec,] <- sch_c15
} else { if (I==16) {
  sch_work[vec,] <- sch_c16
} else { if (I==17) {
  sch_work[vec,] <- sch_c17
} else { if (I==18) {
  sch_work[vec,] <- sch_c18
} else { if (I==19) {
  sch_work[vec,] <- sch_c19
} else { if (I==20) {
  sch_work[vec,] <- sch_c20
} else { if (I==21) {
  sch_work[vec,] <- sch_c21
} else { if (I==22) {
  sch_work[vec,] <- sch_c22
} else { if (I==23) {
  sch_work[vec,] <- sch_c23
} else { if (I==24) {
  sch_work[vec,] <- sch_c24
}}}
```

```
        if ( loadings_check[p] == FALSE) {
          print( paste( " - ",paste0(as.character(round(i/end,1)*100),"%")) )
          loadings[p] <- TRUE
          p <- p+1L
        }

      }

      if (i==end-3L) {
        break
      }
      i <- i + 1L

    }
    return(sch_work)
  }

  #### Wake Turbulence Vector
  # - lead_follow = cat. leader + cat. follow
  # - sep min
  # We have added an row with leader "N" to compute the first iteration

  add <- data.table( t(c("N",rep(0, dim(wt)[1]))))
  colnames(add) <- colnames(wt)

  wt <- rbind(wt, add)

  wt_vector <- melt(wt, "Lead/Follow", variable.name = "follow",value.name =
"sep_min")
  wt_vector$lead_follow <- paste0(wt_vector$'Lead/Follow', wt_vector$"follow")
  wt_vector <- wt_vector[, .(lead_follow, sep_min)]
  wt_vector$sep_min <- as.integer(wt_vector$sep_min)

  rm(add)

  #### Ac_types_vector
  # - ac_model
  # - ac_cat

  ac_types[, id := seq(.N)]
  ac_types_vector <- melt( ac_types, "id", value.name = "ac_model",
variable.name="ac_cat")
  ac_types_vector <- ac_types_vector[ac_model != ""][, .(ac_model, ac_cat)]

  #### Matrix with all possible combinations with 4 numbers
  combis4 = rbind(c(1,2,3,4), c(1,2,4,3), c(1,3,2,4), c(1,3,4,2),
                  c(1,4,2,3), c(1,4,3,2), c(2,1,3,4), c(2,1,4,3),
                  c(2,3,1,4), c(2,3,4,1), c(2,4,1,3), c(2,4,3,1),
                  c(3,1,2,4), c(3,1,4,2), c(3,2,1,4), c(3,2,4,1),
                  c(3,4,1,2), c(3,4,2,1), c(4,1,2,3), c(4,1,3,2),
                  c(4,2,1,3), c(4,2,3,1), c(4,3,1,2), c(4,3,2,1) )
```

```
  #### sch manipulation
  # - Get all the columns we only need in sch
  # - Add category column
  # - Separate departures from arrivals

  sch <- sch[ , .(callsign, ac_model, time_atm, a_d)]

  # Adding cat to sch
  sch <- merge(sch, ac_types_vector, by = "ac_model")
  # With merge the data.table is unordered, we ordered by time
  sch <- sch [ order( as.ITime(sch[, time_atm])) , ]

  # Separate between departures and arrivals
  sch_departures <- sch [ sch [,a_d] == "D", ]
  sch_arrivals <- sch [ sch [,a_d] == "A", ]

  #### Optimal solution
  print("Calculating optimal solution for departures:")
  sch_dep_result <- optimize_delay(sch_departures,combis4)
    rm(sch_departures)
    delay_dep <- sum(sch_dep_result$delay)
    print(paste("Total delay for departures:",delay_dep))
  print("Calculating optimal solution for arrivals:")
  sch_arr_result <- optimize_delay(sch_arrivals,combis4)
    rm(sch_arrivals)
    delay_arr <- sum(sch_arr_result$delay)
    print(paste("Total delay for arrivals:",delay_arr))

  #### Combining both data.tables
  delay_total <- delay_dep + delay_arr
  print(paste("Total delay:",delay_total))
  solution <- rbind(sch_dep_result, sch_arr_result)

  solution <- solution[, .(callsign, ac_cat, a_d, time_atm)]
  solution <- solution[, .(callsign, ac_cat, time_atm)]

  return(solution)

} #end function rs_heur
```

SA Constructive

Enunciat

Activity 4 **SA** Constructive

part 1 **Input** data

Real schedule

	callsign	airp	a_d	airl	ac_model	tail	time_sch	time_atm
1:	VLG181J	EDDM	D	VLG	A320	EC-MBL	07:00:00	07:03:55
2:	VLG142E	LEBB	D	VLG	A320	EC-MNZ	07:00:00	07:05:59
3:	VLG7010	LEMD	D	VLG	A320	EC-MVM	07:00:00	07:08:37
4:	VLG34DP	LIPZ	D	VLG	A321	EC-MHS	07:00:00	07:10:10
5:	VLG108G	LFPO	D	VLG	A20N	EC-NCG	07:00:00	07:12:54

750:	EJU84CE	LIMC	D	EZY	A319	OE-LQV	21:50:00	21:53:45
751:	VLG35AW	LEIB	D	VLG	A320	EC-LVO	21:55:00	21:56:02
752:	WZZ172	LROP	D	WZZ	A321	HA-LXZ	21:55:00	21:57:28
753:	RYR2UG	LEPA	A	RYR	B738	EI-EMA	22:00:00	21:46:41
754:	AEA6103	LEPA	D	AEA	B738	EC-ISN	22:00:00	22:00:00

+

Airlines/Terminal
airl_term.csv

Airports info
airp_info.csv

LEBL stands
lebl_stands.csv

FAA-AC
FAA-AC.csv

Activity 4 **SA** Constructive

part 1 Heuristic **specs**

```
stand_util <- function(schedule, stand_info){
  sch <- fread(paste0("schedules/", schedule))
  stands <- fread(stand_info)
  airl_term <- fread("data/airl_term.csv")
  airp_info <- fread("data/airp_info.csv")
  faa <- fread("data/FAA-AC.csv")

  # your code

  return(solution)
}
```

no loops

Solution: data.table with, *callsign*, *id*, *op*, *tail*, *ramp*, *stand*, *op_time1*, *op_time2*, *time1*, *time2* and *util*

Activity 4 **SA** Constructive

part 1 **Output** data

LEBL stands

lebl_stands.csv

+

Heuristic

stand_util.txt

```
        callsign  id op   tail ramp stand op_time1 op_time2    time1    time2     util
 1: SLEEP_VLG108G   1  P EC-NCG  R10   200 06:00:00 06:45:00 06:00:00 07:00:00 54.01431
 2: SLEEP_VLG108G   1  P EC-NCG  R10  200R 06:00:00 06:45:00 06:00:00 07:00:00 57.96737
 3: SLEEP_VLG108G   1  P EC-NCG  R10   202 06:00:00 06:45:00 06:00:00 07:00:00 54.01431
 4: SLEEP_VLG108G   1  P EC-NCG  R10   204 06:00:00 06:45:00 06:00:00 07:00:00 54.01431
 5: SLEEP_VLG108G   1  P EC-NCG  R10   206 06:00:00 06:45:00 06:00:00 07:00:00 54.01431
---
```

id refers to all operations within an arrival-departure stay of an aircraft

Activity 4 **SA** Constructive

part 2 groups of 2

Solve the Stand Allocation problem for the Schedule.

Activity 4 **SA** Constructive

part 2 Heuristic **specs**

```
# part1
stand_util <- function(schedule, stand_info){...}

# part2
sa_heur <- function(schedule, stand_info){
  s_util <- stand_util(schedule, stand_info)

  # your code

  return(solution)
}
```

up to **2 loops**

Solution: data.table with *callsign*, *op* and *stand*

Activity 4 **SA** Constructive

part 2 **Output** data

Heuristic

sa_heur.txt

+

Result

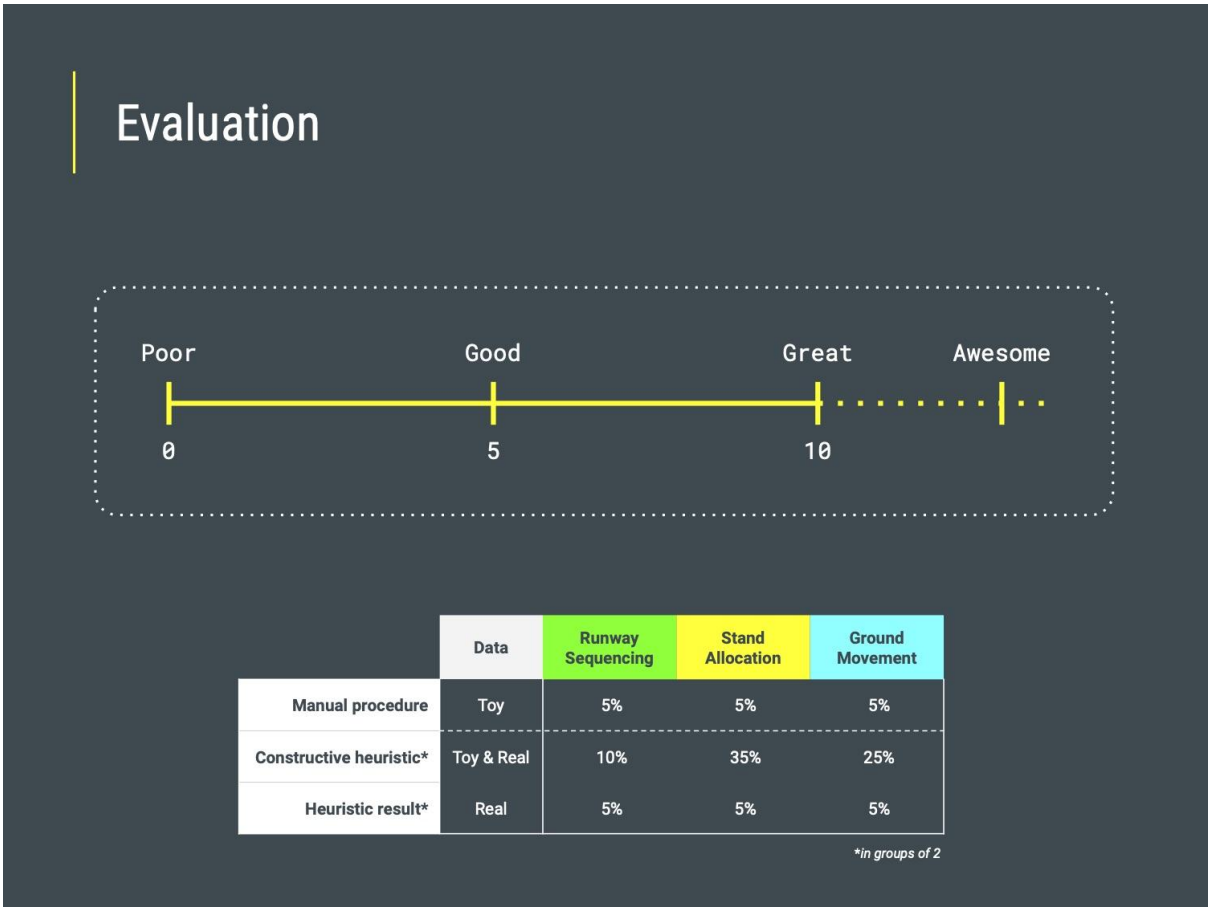
sa_heur.csv

```
   callsign op stand
 1:  VLG181J  A  200R
 2:  VLG142E  A   202
 3:  VLG7010  D   204
 4:  VLG34DP  D   208
 5:  VLG108G  A  154A
---
```

Evaluation: sa_eval_heuristic.csv

Rúbrica

El resultat de la seva funció és comparat amb els valors de la rúbrica. Qualsevol valor intermig tindrà una nota intermitja seguint una llei lineal. La funció es testeja amb varies dades externes i es fa la mitjana de la nota per cada base de dades.



File	Optim	Great	Good	Poor
SCH_01	94235	92946	91656	90367
SCH_02	106408	105080	103751	102423
SCH_03	99503	98602	97700	96799
SCH_04	101397	100587	99776	98966
SCH_05	103256	102438	101620	100802
SCH_06	107902	107507	107112	106717
SCH_07	92817	91660	90502	89345
SCH_08	105872	104823	103773	102724
SCH_09	96164	94529	92894	91259
SCH_10	92333	91664	90994	90325

SCH_11	99390	98528	97665	96803
SCH_12	102553	101814	101075	100336
SCH_13	103594	102707	101819	100932
SCH_14	92696	91609	90521	89434
SCH_15	103887	102834	101780	100727

Entregues

BE - 2,5

```
sa_heur<- function(schedule,stand_info){

#rm(list = ls())
# INITIALIZATION ----
setwd("/Users/escorbmari/Dropbox/EOIN - BARTO/AO/EOIN+BARTO/2 STAND ALLOCATION/2
HEURISTIC/BARTO")
stand_info<-"LEBL_stands_tofill.csv"
schedule<-"SCH_06.csv"

stand_util<- function(schedule,stand_info){

  # INITIALIZATION ----

  setwd("/Users/escorbmari/Dropbox/EOIN - BARTO/AO/EOIN+BARTO/2 STAND ALLOCATION/2
HEURISTIC/BARTO")

  #Valores input de funcion
  stand_info<-"LEBL_stands_tofill.csv"
  schedule<-"SCH_06.csv"

  library(data.table)

  sch<-fread(paste0("schedules/",schedule))
  stands<-fread(stand_info)
  airl_term<-fread("data/airl_term.csv")
  airp_info<-fread("data/airp_info.csv")
  faa<-fread("data/FAA-AC.csv")

   # PRE-PROCESS ----
  ## FLIGHTS CATEGORIES ----
  airp_info[,name:=NULL]
  airp_info[,city:=NULL]
  airp_info[,tz:=NULL]
  airp_info[,lat:=NULL]
  airp_info[,lon:=NULL]

names(sch)<-c("callsign","icao","a_d","airl","ac_model","tail","time_sch","time_atm
")
  sch[,time_atm:=NULL]
  sch<-merge(sch,airp_info,by="icao",sort=FALSE)

  sch$country [sch$country == "Austria"] <- "sch"
  sch$country [sch$country == "Belgium"] <- "sch"
  sch$country [sch$country == "Bulgaria"] <- "sch"
  sch$country [sch$country == "Czechia"] <- "sch"
  sch$country [sch$country == "Denmark"] <- "sch"
```

```
sch$country [sch$country == "Estonia"] <- "sch"
sch$country [sch$country == "Finland"] <- "sch"
sch$country [sch$country == "France"] <- "sch"
sch$country [sch$country == "Germany"] <- "sch"
sch$country [sch$country == "Greece"] <- "sch"
sch$country [sch$country == "Hungary"] <- "sch"
sch$country [sch$country == "Iceland"] <- "sch"
sch$country [sch$country == "Italy"] <- "sch"
sch$country [sch$country == "Latvia"] <- "sch"
sch$country [sch$country == "Lithuania"] <- "sch"
sch$country [sch$country == "Luxembourg"] <- "sch"
sch$country [sch$country == "Malta"] <- "sch"
sch$country [sch$country == "Netherlands"] <- "sch"
sch$country [sch$country == "Norway"] <- "sch"
sch$country [sch$country == "Poland"] <- "sch"
sch$country [sch$country == "Portugal"] <- "sch"
sch$country [sch$country == "Slovakia"] <- "sch"
sch$country [sch$country == "Slovenia"] <- "sch"
sch$country [sch$country == "Spain"] <- "sch"
sch$country [sch$country == "Sweden"] <- "sch"
sch$country [sch$country == "Switzerland"] <- "sch"
sch$country [sch$country == "Liechtenstein"] <- "sch"

sch$country [sch$country == "Belarus"] <- "no_sch"
sch$country [sch$country == "Croatia"] <- "no_sch"
sch$country [sch$country == "Ireland"] <- "no_sch"
sch$country [sch$country == "Moldova"] <- "no_sch"
sch$country [sch$country == "Montenegro"] <- "no_sch"
sch$country [sch$country == "North Macedonia"] <- "no_sch"
sch$country [sch$country == "Romania"] <- "no_sch"
sch$country [sch$country == "Serbia"] <- "no_sch"
sch$country [sch$country == "Turkey"] <- "no_sch"
sch$country [sch$country == "Ukraine"] <- "no_sch"
sch$country [sch$country == "United Kingdom"] <- "no_sch"

Shuttle_ICAO <- c("LEMD","LEBL")

sch$country [sch$country == "Algeria"] <- "no_ue"
sch$country [sch$country == "Argentina"] <- "no_ue"
sch$country [sch$country == "Brazil"] <- "no_ue"
sch$country [sch$country == "Canada"] <- "no_ue"
sch$country [sch$country == "Chile"] <- "no_ue"
sch$country [sch$country == "China"] <- "no_ue"
sch$country [sch$country == "Colombia"] <- "no_ue"
sch$country [sch$country == "Egypt"] <- "no_ue"
sch$country [sch$country == "Faroe Islands"] <- "no_ue"
sch$country [sch$country == "Gambia"] <- "no_ue"
sch$country [sch$country == "Georgia"] <- "no_ue"
sch$country [sch$country == "Hong Kong"] <- "no_ue"
sch$country [sch$country == "Iran"] <- "no_ue"
sch$country [sch$country == "Israel"] <- "no_ue"
```

```
  sch$country [sch$country == "Jordan"] <- "no_ue"
  sch$country [sch$country == "Lebanon"] <- "no_ue"
  sch$country [sch$country == "Mexico"] <- "no_ue"
  sch$country [sch$country == "Morocco"] <- "no_ue"
  sch$country [sch$country == "Pakistan"] <- "no_ue"

  sch$country [sch$country == "Peru"] <- "no_ue"
  sch$country [sch$country == "Qatar"] <- "no_ue"
  sch$country [sch$country == "Russia"] <- "no_ue"
  sch$country [sch$country == "Senegal"] <- "no_ue"
  sch$country [sch$country == "Singapore"] <- "no_ue"
  sch$country [sch$country == "South Korea"] <- "no_ue"
  sch$country [sch$country == "Singapore"] <- "no_ue"
  sch$country [sch$country == "Tunisia"] <- "no_ue"
  sch$country [sch$country == "United Arab Emirates"] <- "no_ue"
  sch$country [sch$country == "United States"] <- "no_ue"

  sch$country [sch$icao == "LEMD"] <- "shut"
  sch$country [sch$icao == "LEBL"] <- "shut"

  sch$border<-sch$country
  sch[,country:=NULL]
  sch<-sch[order(sch$time_sch)]


  ## TERMINAL-STANDS ----
  stands$terminal [stands$T2 == "1"] <- "2"
  stands$terminal [stands$T1 == "1"] <- "1"

  airl_term[,name:=NULL]
  names(airl_term)<-c("airl","terminal")
  sch<-merge(sch,airl_term,by="airl",sort=FALSE)

  rm(airl_term,airp_info)

  ## AREA OF AC_MODELs ----
  faa[,area:=ac_span*ac_length]
  faa[,model:=NULL]
  faa[,ac_span:=NULL]
  faa[,ac_length:=NULL]
  faa[,mtow:=NULL]
  faa[,wct:=NULL]

  area_min<-4000


names(stands)<-c("ramp","stand","icao_code","S1","S2","S3","contact","T1","T2","sch
","shut","no_sch","no_ue","park","terminal")
  stands[,order:=seq(.N)]
  stands<-merge(faa,stands,by="icao_code",sort=FALSE)
  stands<-stands[order(stands$order)]
```

```
  stands[,order:=NULL]

names(stands)<-c("ac_max","area_max","ramp","stand","S1","S2","S3","contact","T1","
T2","sch","shut","no_sch","no_ue","park","terminal")

setcolorder(stands,c("ramp","stand","ac_max","S1","S2","S3","contact","T1","T2","sc
h","shut","no_sch","no_ue","park","area_max","terminal"))

  ## INCOMPATIBILITIES ----

  stands[,S1_beta:= S1>=0]
  stands[,S2_beta:= S2>=0]
  stands[,S3_beta:= S3>=0]
  stands[,incompatibilities:= S1_beta+S2_beta+S3_beta]
  stands[,S1_beta:= NULL]
  stands[,S2_beta:= NULL]
  stands[,S3_beta:= NULL]

  sch_A<-sch[a_d=="A"]
  sch_D<-sch[a_d=="D"]



  # FLIGHTS PROCESSING ----
  ## SLEEP_XXXX ----
  time1 <- sch[, min(time_sch), by = tail]
  names(time1)<-c("tail","time_sch")
  sleep_D <- merge(sch, time1, by = "tail")

names(sleep_D)<-c("tail","airl","icao","callsign","a_d","ac_model","time_sch","bord
er","terminal","time1")
  sleep_D <- sleep_D[(sleep_D$a_d == "D") & (sleep_D$time_sch == sleep_D$time1)]

  ## XXXX_SLEEP ----
  time2 <- sch[, max(time_sch), by = tail]
  names(time2)<-c("tail","time_sch")
  sleep_A <- merge(sch, time2, by = "tail")

names(sleep_A)<-c("tail","airl","icao","callsign","a_d","ac_model","time_sch","bord
er","terminal","time1")
  sleep_A <- sleep_A[(sleep_A$a_d == "A") & (sleep_A$time_sch == sleep_A$time1)]
  #sleep_A$callsign <- paste(sleep_A$callsign,"_SLEEP",sep='')

  ## COMPLETE ROTATIONS ----
  time3 <- sch[, min(time_sch), by = tail]
  names(time3)<-c("tail","time_sch")
  COMP <- merge(sch, time3, by = "tail")

names(COMP)<-c("tail","airl","icao","callsign","a_d","ac_model","time_sch","border"
,"terminal","time1")
```

```
  time4 <- sch[, max(time_sch), by = tail]
  names(time4)<-c("tail","time_sch")
  COMP <- merge(COMP, time4, by = "tail")

names(COMP)<-c("tail","airl","icao","callsign","a_d","ac_model","time_sch","border"
,"terminal","time1","time2")

  COMP$is_sleep <- ""
  COMP$is_sleep <- ifelse(COMP$a_d == "D" & COMP$time_sch == COMP$time1, "YES",
COMP$is_sleep)
  COMP$is_sleep <- ifelse(COMP$a_d == "A" & COMP$time_sch == COMP$time2, "YES",
COMP$is_sleep)

  COMP<-COMP[(COMP$is_sleep==""),]

  COMP_A <- COMP[COMP$a_d == 'A']
  COMP_D <- COMP[COMP$a_d == 'D']

  ## ASSIGN IDs TO OPs ----
  COMP_A<-COMP_A[order(COMP_A$time_sch,tail)]
  COMP_D<-COMP_D[order(COMP_D$time_sch,tail)]

  COMP_A[,time2:=NULL]
  COMP_A[,is_sleep:=NULL]

  OPS_ID <-  rbind(sleep_A,sleep_D,COMP_A)
  OPS_ID <-OPS_ID [order(OPS_ID$time_sch,tail)]
  OPS_ID$ID <- seq.int(nrow(OPS_ID))

  COMP_A_ID <- merge(COMP_A, OPS_ID, by = "callsign")
  sleep_A_ID <- merge(sleep_A, OPS_ID, by = "callsign")
  sleep_D_ID <- merge(sleep_D, OPS_ID, by = "callsign")

  COMP_A_ID <-COMP_A_ID [order(COMP_A_ID$ID)]
  sleep_A_ID <-sleep_A_ID [order(sleep_A_ID$ID)]
  sleep_D_ID <-sleep_D_ID [order(sleep_D_ID$ID)]

  COMP_A_ID <- COMP_A_ID[,-(11:19),drop=FALSE]

names(COMP_A_ID)<-c("callsign","tail","airl","icao","a_d","ac_model","time_sch","bo
rder","terminal","time1","id")
  sleep_A_ID <- sleep_A_ID[,-(11:19),drop=FALSE]

names(sleep_A_ID)<-c("callsign","tail","airl","icao","a_d","ac_model","time_sch","b
order","terminal","time1","id")
  sleep_D_ID <- sleep_D_ID[,-(11:19),drop=FALSE]

names(sleep_D_ID)<-c("callsign","tail","airl","icao","a_d","ac_model","time_sch","b
order","terminal","time1","id")

  aux_id <- COMP_A_ID[order(COMP_A_ID$tail,time_sch)]
```

```
  id <- aux_id[,c(id)]
  COMP_D <- COMP_D[order(COMP_D$tail,time_sch)]

  COMP_D_ID <- cbind(COMP_D,id)
  COMP_D_ID <- COMP_D_ID[order(COMP_D_ID$id)]
  COMP_D_ID <- COMP_D_ID[,-(11:12),drop=FALSE]

setcolorder(COMP_D_ID,c("callsign","tail","airl","icao","a_d","ac_model","time_sch"
,"border","terminal","time1","id"))

  rm(COMP_A,COMP_D,sleep_A,sleep_D,id,sch_A,sch_D,time1,time2,time3,time4,aux_id)


  ## PARKING OP CREATION FOR COMPLETE ROTATIONS ----
  P_COMP <- merge(COMP_A_ID, COMP_D_ID, by = "id")

setcolorder(P_COMP,c("callsign.x","callsign.y","id","a_d.x","tail.x","time_sch.x","
time_sch.y","terminal.x","ac_model.x","border.x"))
  P_COMP <- P_COMP[,-(11:21),drop=FALSE]
  P_COMP[,a_d.x:= NULL]
  P_COMP$op <- "P"

names(P_COMP)<-c("callsignA","callsignD","id","tail","time1","time2","terminal","ac
_model","border","op")
  setcolorder(P_COMP,c("callsignA","callsignD","id","op","tail","time1","time2"))
  P_COMP$callsign <- paste(P_COMP$callsignA,P_COMP$callsignD,sep = '_')
  P_COMP[,callsignA:= NULL]
  P_COMP[,callsignD:= NULL]
  setcolorder(P_COMP,c("callsign","id","op","tail","time1","time2"))

  P_COMP$time2 <- as.ITime(P_COMP$time2)
  OPERATION_TIME <- "00:15:00"
  P_COMP$op_time1 <-  as.ITime(P_COMP$time1) +  as.ITime(OPERATION_TIME)
  P_COMP$op_time2 <-  as.ITime(P_COMP$time2) -  as.ITime(OPERATION_TIME)

setcolorder(P_COMP,c("callsign","id","op","tail","op_time1","op_time2","time1","tim
e2","terminal","ac_model"))


  ## CALCULATE op_time1 y op_time2 FOR ARRIVALS  ----
  A_COMP <- merge(COMP_A_ID, COMP_D_ID, by = "id")

setcolorder(A_COMP,c("callsign.x","id","a_d.x","tail.x","time_sch.x","time_sch.y","
terminal.x","ac_model.x","border.x"))
  A_COMP <- A_COMP[,-(10:21),drop=FALSE]
  A_COMP[,a_d.x:= NULL]
  A_COMP$op <- "A"


names(A_COMP)<-c("callsign","id","tail","time1","time2","terminal","ac_model","bord
er","op")
```

```
  setcolorder(A_COMP,c("callsign","id","op","tail","time1","time2"))

  A_COMP$time2 <- as.ITime(A_COMP$time2)
  OPERATION_TIME <- "00:15:00"
  A_COMP$op_time1 <-  as.ITime(A_COMP$time1)
  A_COMP$op_time2 <-  as.ITime(A_COMP$time1) +  as.ITime(OPERATION_TIME)

setcolorder(A_COMP,c("callsign","id","op","tail","op_time1","op_time2","time1","tim
e2","terminal","ac_model"))

  ## CALCULATE op_time1 y op_time2 FOR DEPARTURES ----
  D_COMP <- merge(COMP_A_ID, COMP_D_ID, by = "id")

setcolorder(D_COMP,c("callsign.y","id","tail.x","time_sch.x","time_sch.y","terminal
.x","ac_model.x","border.x"))
  D_COMP <- D_COMP[,-(9:21),drop=FALSE]
  D_COMP$op <- "D"

names(D_COMP)<-c("callsign","id","tail","time1","time2","terminal","ac_model","bord
er","op")
  setcolorder(D_COMP,c("callsign","id","op","tail","time1","time2"))

  D_COMP$time2 <- as.ITime(D_COMP$time2)

  OPERATION_TIME <- "00:15:00"
  D_COMP$op_time1 <-  as.ITime(D_COMP$time2) -  as.ITime(OPERATION_TIME)
  D_COMP$op_time2 <-  as.ITime(D_COMP$time2)

setcolorder(D_COMP,c("callsign","id","op","tail","op_time1","op_time2","time1","tim
e2","terminal","ac_model"))


  ## ERROR ## CALCULATE op_time1 y op_time2 FOR sleep_A ----
  SLEEP_A_COMPLETE <- sleep_A_ID

setcolorder(SLEEP_A_COMPLETE,c("callsign","id","tail","time_sch","terminal","ac_mod
el","border"))
  #SLEEP_A_COMPLETE <- SLEEP_A_COMPLETE[,-(7:11),drop=FALSE]
  SLEEP_A_COMPLETE$op <- "A"
  SLEEP_A_COMPLETE$time2 <- "23:00:00"
  SLEEP_A_COMPLETE$time2 <- as.ITime(SLEEP_A_COMPLETE$time2)


#names(SLEEP_A_COMPLETE)<-c("callsign","id","tail","time1","terminal","ac_model","o
p","time2")

#setcolorder(SLEEP_A_COMPLETE,c("callsign","id","op","tail","time1","time2","termin
al","ac_model"))

  OPERATION_TIME <- "00:15:00"
  SLEEP_A_COMPLETE$op_time1 <- as.ITime(SLEEP_A_COMPLETE$time_sch)
```

```
  SLEEP_A_COMPLETE$op_time2 <-  as.ITime(SLEEP_A_COMPLETE$op_time1) +
as.ITime(OPERATION_TIME)

setcolorder(SLEEP_A_COMPLETE,c("callsign","id","op","tail","op_time1","op_time2","t
ime1","time2","terminal","ac_model","border"))
  SLEEP_A_COMPLETE[,time_sch:= NULL]
  SLEEP_A_COMPLETE[,airl:= NULL]
  SLEEP_A_COMPLETE[,icao:= NULL]
  SLEEP_A_COMPLETE[,a_d:= NULL]


  ## ERROR ## CALCULATE op_time1 y op_time2 FOR sleep_D ----
  SLEEP_D_COMP <- sleep_D_ID

setcolorder(SLEEP_D_COMP,c("callsign","id","tail","time_sch","terminal","ac_model",
"border"))
  #SLEEP_D_COMP <- SLEEP_D_COMP[,-(7:11),drop=FALSE]
  SLEEP_D_COMP$op <- "D"
  SLEEP_D_COMP$time1 <- "06:00:00"


#names(SLEEP_D_COMP)<-c("callsign","id","tail","time2","terminal","ac_model","op","
time1")

#setcolorder(SLEEP_D_COMP,c("callsign","id","op","tail","time1","time2","terminal",
"ac_model"))

  OPERATION_TIME <- "00:15:00"
  SLEEP_D_COMP$time2 <- as.ITime(SLEEP_D_COMP$time_sch)
  SLEEP_D_COMP$op_time2 <- as.ITime(SLEEP_D_COMP$time_sch)
  SLEEP_D_COMP$op_time1 <-  as.ITime(SLEEP_D_COMP$time_sch) -
as.ITime(OPERATION_TIME)

setcolorder(SLEEP_D_COMP,c("callsign","id","op","tail","op_time1","op_time2","time1
","time2","terminal","ac_model","border"))

  SLEEP_D_COMP[,time_sch:= NULL]
  SLEEP_D_COMP[,airl:= NULL]
  SLEEP_D_COMP[,icao:= NULL]
  SLEEP_D_COMP[,a_d:= NULL]

  ## CALCULATE op_time1 y op_time2 FOR PARKINGS OF sleep_A ----
  aux <- SLEEP_A_COMPLETE
  SLEEP_A_P_COMP <- aux
  SLEEP_A_P_COMP[,op_time1:= NULL]
  SLEEP_A_P_COMP[,op_time2:= NULL]
  SLEEP_A_P_COMP[,op:= NULL]
  SLEEP_A_P_COMP$op <- "P"
  pm <- "23:00:00"
  SLEEP_A_P_COMP$op_time2 <- as.ITime(pm)
```

```
  OPERATION_TIME <- "00:15:00"
  SLEEP_A_P_COMP$op_time1 <-  as.ITime(SLEEP_A_P_COMP$time1) +
as.ITime(OPERATION_TIME)

setcolorder(SLEEP_A_P_COMP,c("callsign","id","op","tail","op_time1","op_time2","tim
e1","time2"))
  SLEEP_A_P_COMP$callsign <- paste(SLEEP_A_P_COMP$callsign,"_SLEEP",sep='')

  ## CALCULATE op_time1 y op_time2 FOR PARKINGS OF sleep_D ----
  aux2 <- SLEEP_D_COMP
  SLEEP_D_P_COMP <- aux2
  SLEEP_D_P_COMP[,op_time1:= NULL]
  SLEEP_D_P_COMP[,op_time2:= NULL]
  SLEEP_D_P_COMP[,op:= NULL]
  SLEEP_D_P_COMP$op <- "P"
  six_am <- "06:00:00"
  SLEEP_D_P_COMP$op_time1 <- as.ITime(six_am)

  OPERATION_TIME <- "00:15:00"
  SLEEP_D_P_COMP$op_time2 <-  as.ITime(SLEEP_D_P_COMP$time2) -
as.ITime(OPERATION_TIME)

setcolorder(SLEEP_D_P_COMP,c("callsign","id","op","tail","op_time1","op_time2","tim
e1","time2"))
  SLEEP_D_P_COMP$callsign <- paste("SLEEP_",SLEEP_D_P_COMP$callsign,sep='')

  ## SOLVING ERROR in SLEEP_A_COMPLETE & SLEEP_D_COMP----

  SLEEP_A_COMPLETE$op <- "A"
  OPERATION_TIME <- "00:15:00"
  SLEEP_A_COMPLETE$op_time1 <- as.ITime(SLEEP_A_COMPLETE$time1)
  SLEEP_A_COMPLETE$op_time2 <-  as.ITime(SLEEP_A_COMPLETE$op_time1) +
as.ITime(OPERATION_TIME)

setcolorder(SLEEP_A_COMPLETE,c("callsign","id","op","tail","op_time1","op_time2","t
ime1","time2","terminal","ac_model","border"))
  SLEEP_A_COMPLETE$time2 <- as.ITime(SLEEP_A_COMPLETE$time2)


  SLEEP_D_COMP$op <- "D"
  OPERATION_TIME <- "00:15:00"
  SLEEP_D_COMP$op_time2 <- as.ITime(SLEEP_D_COMP$time2)
  SLEEP_D_COMP$op_time1 <-  as.ITime(SLEEP_D_COMP$time2) -
as.ITime(OPERATION_TIME)

setcolorder(SLEEP_D_COMP,c("callsign","id","op","tail","op_time1","op_time2","time1
","time2","terminal","ac_model","border"))



  # FINAL STEPS
```

```
  ## ASSIGN AVAILABLE STANDS ----
  schedule <-
rbind(A_COMP,P_COMP,D_COMP,SLEEP_A_COMPLETE,SLEEP_A_P_COMP,SLEEP_D_COMP,SLEEP_D_P_C
OMP)

  schedule$terminal <- as.integer(schedule$terminal)
  stands$terminal <- as.integer(stands$terminal)

  aux <- stands
  aux3<-melt(aux,c("ramp","stand","incompatibilities","area_max","terminal"))

  schedule_stands <- merge(schedule, aux3, by.x = "border", by.y = "variable",
allow.cartesian=TRUE)
  schedule_stands$value[schedule_stands$value == "5"] <- as.integer(90)
  schedule_stands$value[schedule_stands$value == "4"] <- as.integer(70)
  schedule_stands$value[schedule_stands$value == "3"] <- as.integer(50)
  schedule_stands$value[schedule_stands$value == "2"] <- as.integer(20)
  schedule_stands$value[schedule_stands$value == "1"] <- as.integer(0)

  schedule_stands <- merge (schedule_stands,faa,
by.x="ac_model",by.y="icao_code",allow.cartesian=TRUE)
  schedule_stands<- schedule_stands[area<area_max]

  schedule_stands$ac_fit <- (schedule_stands$area - area_min) /
(schedule_stands$area_max - area_min)

  schedule_stands$incompatibilities <-
as.integer(schedule_stands$incompatibilities)
  schedule_stands$values <- as.integer(schedule_stands$values)

  schedule_stands$util1 <- as.numeric(schedule_stands$value)
  schedule_stands$util2 <- as.numeric(10*schedule_stands$ac_fit)
  schedule_stands$util3 <- as.numeric(schedule_stands$incompatibilities / 10)
  schedule_stands$util1 <- as.numeric(schedule_stands$util1)
  schedule_stands$util2 <- as.numeric(schedule_stands$util2)
  schedule_stands$util3 <- as.numeric(schedule_stands$util3)
  schedule_stands[,util:=util1+util2+util3]


setcolorder(schedule_stands,c("callsign","id","op","tail","ramp","stand","op_time1"
,"op_time2","time1","time2","util"))
  solution <- schedule_stands[,-(12:24),drop=FALSE]

  solution <-solution [order(solution$id,solution$op_time1,-solution$util)]


  return(solution)
}
stand_util <- stand_util(stand_info,schedule)

#  ---------------------------------------------------------------------------
```

```
# NEW FUNCTIONS FOR SA_HEURISTIC PART 2 ----

## 1- STANDS COMBINATION POSSIBILITIES FOR EACH ID ----
  # INPUTS: stand_util_poss, id_iteration, tow_penalty
  # OUTPUTS: stands combination possibilities for each ID, callsign, op

stands_ID <- function(stand_util_poss, id_iteration,tow_penalty){
  stands_for_ID <- stand_util_poss[(id==id_iteration)]
  stands_for_ID <- stands_for_ID[,c("id","op","stand","util","callsign")]
  departure <- stands_for_ID[(op=="D")]
  stands_for_ID <-
merge(stands_for_ID[(op=="A")],stands_for_ID[(op=="P")],by="id",allow.cartesian=TRU
E,all.x=TRUE,all.y=TRUE)

  stands_for_ID <- merge(stands_for_ID,departure,by="id",allow.cartesian = TRUE,
all.x=TRUE,all.y=TRUE)
  #stands_for_ID[is.na(stands_for_ID)]<- 0
  stands_for_ID[,total_utility:=util + util.x + util.y]

  setorder(stands_for_ID,-total_utility)
  stands_for_ID <-
stands_for_ID[,c("id","op.x","stand.x","callsign.x","op.y","stand.y","callsign.y","
op","stand","total_utility","callsign")]

  stands_for_ID[,tow1:=as.numeric((stand.x!=stand.y))]
  stands_for_ID[,tow2:=as.numeric((stand.y!=stand))]
  stands_for_ID[,total_utility:=total_utility-tow_penalty*(tow1+tow2)]

  stands_for_ID[,tow1:=NULL]
  stands_for_ID[,tow2:=NULL]

  setorder(stands_for_ID,-total_utility)
  stands_for_ID <-
stands_for_ID[,c("id","stand.x","callsign.x","stand.y","callsign.y","stand","callsi
gn","total_utility")]
  names(stands_for_ID) <-
c("id","stand_A","callsign_A","stand_P","callsign_P","stand_D","callsign_D","total_
utility")

  return(stands_for_ID)
}


## 2- CHECK IF AN STAND IS AVAILABLE ----
# INPUTS: slots_availability, stand_num, time1, time2
# OUTPUT: TRUE / FALSE

check_stand_availability <- function(slots_availability, stand_num, time1, time2){

  check <- slots_availability[stand==stand_num & slots>=time1 & slots<time2]
```

```
    if(any(check$available == "0")){return(TRUE)}else{return(FALSE)}

}

#  ---------------------------------------------------------------------------
# NEW DATA ----
## TOW PENALTY ----
  tow_penalty = 25

## Table with possibilities per callsign ----
  aux_poss <- table(unlist(stand_util$callsign))
  aux_poss <- data.table(aux_poss)
  names(aux_poss)<-c("callsign","poss")

  stand_util_poss <- merge(stand_util, aux_poss, by="callsign")
  stand_util_poss <-stand_util_poss
[order(stand_util_poss$poss,stand_util_poss$id,stand_util_poss$op_time1,-stand_util
_poss$util)]

## Table with stand, time (5min slots), boolean (0 if it's available and 1 not)
----
  AM <- as.ITime("06:00:00")
  PM <- as.ITime("23:00:00")
  incr <- as.ITime("00:05:00")
  slots <- seq(AM,PM, incr)
  rm(AM,PM,incr)
  slots <- data.table(slots)
  names(slots)<-c("slots")
  slots$aux <- "merge"

  stands<-fread(stand_info)
  setcolorder(stands,c("stand"))
  stands <- stands[,-(2:14),drop=FALSE]
  stands$aux <- "merge"

  slots_availability <- merge(slots, stands, by="aux",allow.cartesian = TRUE)
  setcolorder(slots_availability,c("stand","slots"))
  slots_availability <- slots_availability[,-(3:3),drop=FALSE]
  slots_availability$available <- 0

  # Don't know how to order properly
  # slots_availability <-slots_availability
[order(slots_availability$stand,slots_availability$slots)]

#  ---------------------------------------------------------------------------
# MAIN ------------------------------------------------------------------
  unique_ids <- data.table(unique(stand_util_poss$id))
  sa_heur <- data.table()

  sa_heur$id <- unique_ids
```

```
  sa_heur$stand_A <- ""
  sa_heur$stand_P <- ""

  sa_heur$stand_D <- ""

  sa_heur$callsign_A <- ""
  sa_heur$callsign_P <- ""
  sa_heur$callsign_D <- ""

  sa_heur$op_A <- ""
  sa_heur$op_P <- ""
  sa_heur$op_D <- ""


## FOR to compute all IDs ----
  for (i in 1:nrow(unique_ids)){
    id_iteration <- as.character(unique_ids[i])
    stands_for_ID <- stands_ID(stand_util_poss, id_iteration , tow_penalty)
    unique_stands_A <- data.table(unique(stands_for_ID$stand_A))
    unique_stands_P <- data.table(unique(stands_for_ID$stand_P))
    unique_stands_D <- data.table(unique(stands_for_ID$stand_D))

    total <- nrow(unique_ids)
    pending <- total - i

    # COMMENT THE NEXT LINE BEFORE DELIVERY (not allowed function)
    #print(paste0('Assigning stands to Barcelona Airport. Pending to compute
',pending,' out of ',total,' flights'))

    # FOR to check availability for each stand for each callsign ----
    for (j in 1:nrow(unique_stands_P)) {

      stand_num_A <- as.character(unique_stands_A[j])
      stand_num_P <- as.character(unique_stands_P[j])
      stand_num_D <- as.character(unique_stands_D[j])

      if(stand_num_A == 0 || is.na(stand_num_A)==TRUE){available_A = TRUE

        op_time1_P <-
as.ITime(unique((stand_util_poss$op_time1[stand_util_poss$id==id_iteration]))[1])
        op_time2_P <-
as.ITime(unique((stand_util_poss$op_time2[stand_util_poss$id==id_iteration]))[1])

        op_time1_D <-
as.ITime(unique((stand_util_poss$op_time1[stand_util_poss$id==id_iteration]))[2])
        op_time2_D <-
as.ITime(unique((stand_util_poss$op_time2[stand_util_poss$id==id_iteration]))[2])

        available_P <- check_stand_availability(slots_availability, stand_num_P,
op_time1_P, op_time2_P)
```

```
        available_D <- check_stand_availability(slots_availability, stand_num_D,
op_time1_D, op_time2_D)
        if(available_A==TRUE & available_P==TRUE & available_D==TRUE){break}

      }

      if(stand_num_D == 0 || is.na(stand_num_D)==TRUE){available_D=TRUE

        op_time1_A <-
as.ITime(unique((stand_util_poss$op_time1[stand_util_poss$id==id_iteration]))[1])
        op_time2_A <-
as.ITime(unique((stand_util_poss$op_time2[stand_util_poss$id==id_iteration]))[1])

        op_time1_P <-
as.ITime(unique((stand_util_poss$op_time1[stand_util_poss$id==id_iteration]))[2])
        op_time2_P <-
as.ITime(unique((stand_util_poss$op_time2[stand_util_poss$id==id_iteration]))[2])

        available_A <- check_stand_availability(slots_availability, stand_num_A,
op_time1_A, op_time2_A)
        available_P <- check_stand_availability(slots_availability, stand_num_P,
op_time1_P, op_time2_P)

        if(available_A==TRUE & available_P==TRUE & available_D==TRUE){break}

      }

      if((stand_num_A != 0 || is.na(stand_num_A)==TRUE) & (stand_num_D != 0 ||
is.na(stand_num_D)==TRUE)){

        op_time1_A <-
as.ITime(unique((stand_util_poss$op_time1[stand_util_poss$id==id_iteration]))[1])
        op_time2_A <-
as.ITime(unique((stand_util_poss$op_time2[stand_util_poss$id==id_iteration]))[1])

        op_time1_P <-
as.ITime(unique((stand_util_poss$op_time1[stand_util_poss$id==id_iteration]))[2])
        op_time2_P <-
as.ITime(unique((stand_util_poss$op_time2[stand_util_poss$id==id_iteration]))[2])

        op_time1_D <-
as.ITime(unique((stand_util_poss$op_time1[stand_util_poss$id==id_iteration]))[3])
        op_time2_D <-
as.ITime(unique((stand_util_poss$op_time2[stand_util_poss$id==id_iteration]))[3])

        available_A <- check_stand_availability(slots_availability, stand_num_A,
op_time1_A, op_time2_A)
        available_P <- check_stand_availability(slots_availability, stand_num_P,
op_time1_P, op_time2_P)
        available_D <- check_stand_availability(slots_availability, stand_num_D,
op_time1_D, op_time2_D)
```

```
        if(available_A==TRUE & available_P==TRUE & available_D==TRUE){break}

      }

    }


    if(stand_num_A == 0 || is.na(stand_num_A)==TRUE){
      slots_availability[stand==stand_num_P & slots>=op_time1_P &
slots<op_time2_P]$available <- 1
      slots_availability[stand==stand_num_D & slots>=op_time1_D &
slots<op_time2_D]$available <- 1
      sa_heur$id[i] <- id_iteration
      sa_heur$stand_P[i] <- stand_num_P
      sa_heur$stand_D[i] <- stand_num_D

      sa_heur$callsign_A[i] <- "SLEEP_CASE"
      sa_heur$callsign_P[i] <- as.character(unique(stands_for_ID$callsign_P))
      sa_heur$callsign_D[i] <- as.character(unique(stands_for_ID$callsign_D))

      sa_heur$op_A[i] <- "A"
      sa_heur$op_P[i] <- "P"
      sa_heur$op_D[i] <- "D"

    }

    if(stand_num_D == 0 || is.na(stand_num_D)==TRUE){
      slots_availability[stand==stand_num_A & slots>=op_time1_A &
slots<op_time2_A]$available <- 1
      slots_availability[stand==stand_num_P & slots>=op_time1_P &
slots<op_time2_P]$available <- 1
      sa_heur$id[i] <- id_iteration
      sa_heur$stand_A[i] <- stand_num_A
      sa_heur$stand_P[i] <- stand_num_P

      sa_heur$callsign_A[i] <- as.character(unique(stands_for_ID$callsign_A))
      sa_heur$callsign_P[i] <- as.character(unique(stands_for_ID$callsign_P))
      sa_heur$callsign_D[i] <- "SLEEP_CASE"


      sa_heur$op_A[i] <- "A"
      sa_heur$op_P[i] <- "P"
      sa_heur$op_D[i] <- "D"

    }

    #if((stand_num_A != 0 || is.na(stand_num_A)==TRUE) & (stand_num_D != 0 ||
is.na(stand_num_D)==TRUE)){

    if((is.na(stand_num_A)==FALSE & is.na(stand_num_D)==FALSE)){
```

```
      slots_availability[stand==stand_num_A & slots>=op_time1_A &
slots<op_time2_A]$available <- 1
      slots_availability[stand==stand_num_P & slots>=op_time1_P &
slots<op_time2_P]$available <- 1
      slots_availability[stand==stand_num_D & slots>=op_time1_D &
slots<op_time2_D]$available <- 1

      sa_heur$id[i] <- id_iteration
      sa_heur$stand_A[i] <- stand_num_A
      sa_heur$stand_P[i] <- stand_num_P
      sa_heur$stand_D[i] <- stand_num_D

      sa_heur$callsign_A[i] <- as.character(unique(stands_for_ID$callsign_A))
      sa_heur$callsign_P[i] <- as.character(unique(stands_for_ID$callsign_P))
      sa_heur$callsign_D[i] <- as.character(unique(stands_for_ID$callsign_D))

      sa_heur$op_A[i] <- "A"
      sa_heur$op_P[i] <- "P"
      sa_heur$op_D[i] <- "D"



    }

  }

## SLOTS AVAILABILITY VISUALIZATION ----
  slots_availability <- data.table(slots_availability)
  slots_availability_table <- dcast(slots_availability, stand ~ slots,  value.var =
"available")

  aux <- sa_heur

#  ---------------------------------------------------------------------------
#  SOLUTION ----
## PARTIAL SOLUTION FOR ARRIVALS ----
  solution_ARR <- data.table()

  unique_callsign_A <-
data.table(unique(stand_util_poss$callsign[stand_util_poss$op == "A"]))
  solution_ARR <- unique_callsign_A
  names(solution_ARR) <- c("callsign")
  solution_ARR$op <- "A"

  callsign_A <- data.table(sa_heur$callsign_A)
  callsign_A<- callsign_A[V1!= "SLEEP_CASE",]

  solution_ARR <- merge(aux,callsign_A, by.x="callsign_A",by.y="V1",allow.cartesian
= TRUE)
  solution_ARR <- solution_ARR[,c("callsign_A","op_A","stand_A")]
  names(solution_ARR)  <- c("callsign","op","stand")
```

```
  solution_ARR <- solution_ARR[(solution_ARR$callsign!="")==TRUE]



## PARTIAL SOLUTION FOR PARKING ----
  solution_PARK <- data.table()

  unique_callsign_P <-
data.table(unique(stand_util_poss$callsign[stand_util_poss$op == "P"]))
  solution_PARK <- unique_callsign_P
  names(solution_PARK) <- c("callsign")
  solution_PARK$op <- "P"

  callsign_P <- data.table(sa_heur$callsign_P)
  callsign_P<- callsign_P[V1!= "SLEEP_CASE",]

  solution_PARK <- merge(aux,callsign_P,
by.x="callsign_P",by.y="V1",allow.cartesian = TRUE)
  solution_PARK <- solution_PARK[,c("callsign_P","op_P","stand_P")]
  names(solution_PARK)  <- c("callsign","op","stand")

  solution_PARK <- solution_PARK[(solution_PARK$callsign!="")==TRUE]

## PARTIAL SOLUTION FOR DEPARTURES ----
  solution_DEP <- data.table()

  unique_callsign_D <-
data.table(unique(stand_util_poss$callsign[stand_util_poss$op == "D"]))
  solution_DEP <- unique_callsign_D
  names(solution_DEP) <- c("callsign")
  solution_DEP$op <- "D"

  callsign_D <- data.table(sa_heur$callsign_D)
  callsign_D<- callsign_D[V1!= "SLEEP_CASE",]

  solution_DEP <- merge(aux,callsign_D, by.x="callsign_D",by.y="V1",allow.cartesian
= TRUE)
  solution_DEP <- solution_DEP[,c("callsign_D","op_D","stand_D")]
  names(solution_DEP)  <- c("callsign","op","stand")

  solution_DEP <- solution_DEP[(solution_DEP$callsign!="")==TRUE]


## JOIN PARTIAL SOLUTIONS ----
  solution <- rbind(solution_ARR,solution_PARK,solution_DEP)

# COMMENT THE NEXT LINE BEFORE DELIVERY (not allowed function)
#write.csv(solution, file = "sa_heur.csv", sep=",", row.names=FALSE)
#write.table(solution, "SA_06.csv", sep = ",", row.names=FALSE, dec = " ")
```

```
# RETURN SOLUTION ----
return(solution)
}
```

JP - 6,7

```
#Part 1
stand_util <- function(schedule, stand_info){
  sch <- fread(paste0("schedules/", schedule))
  stands <- fread(stand_info)
  airl_term <- fread("data/airl_term.csv")
  airp_info <- fread("data/airp_info.csv")
  faa <- fread("data/FAA-AC.csv")

  ##Afegir parking

  sch <- sch[, .(callsign, airp, a_d, airl, ac_model, tail, time_sch)]
  time_min <- as.ITime(sch$time_sch[1]) - as.ITime("1:00")
  time_max <- as.ITime(sch$time_sch[length(sch$time_sch)]) + as.ITime("1:00")

  # Mirem operacio previa

  sch <- sch[order( sch[, c(tail )]), ]
  sch$tail_ant <- c('0',sch$tail[1:length(sch$tail)-1])
  sch$tail_sig <- c(sch$tail[2:length(sch$tail)],'0')

  sch$a_d_ant <- c('0',sch$a_d[1:length(sch$a_d)-1])
  sch$a_d_sig <- c(sch$a_d[2:length(sch$a_d)],'0')

  sch$time_ant <- as.ITime( c('0',sch$time_sch[1:length(sch$time_sch)-1]) )
  sch$time_sig <- as.ITime( c(sch$time_sch[2:length(sch$time_sch)],'0') )

  sch$callsign_ant <- c('0',sch$callsign[1:length(sch$callsign)-1])
  sch$callsign_sig <- c(sch$callsign[2:length(sch$callsign)],'0')

  sch$time1 <- as.ITime("0:00")
  sch$time2 <- sch$time_sig
  sch$op_time1 <- as.ITime("0:00")
  sch$op_time2 <- as.ITime("0:00")

  # TIME1 AND OP_TIME1  de A

  sch[sch$a_d == "A"]$op_time1 <- as.ITime(sch[sch$a_d == "A"]$time_sch)
  sch[sch$a_d == "A"]$time1 <- as.ITime(sch[sch$a_d == "A"]$time_sch)

  # A + 15
  sch[sch$a_d == "A"]$op_time2 <- as.ITime(sch[sch$a_d == "A"]$op_time1) +
as.ITime("00:15")


  # TIME2 AND OP_TIME2  de D
  sch[sch$a_d == "D"]$op_time2 <- as.ITime(sch[sch$a_d == "D"]$time_sch)
  sch[sch$a_d == "D"]$time2 <- as.ITime(sch[sch$a_d == "D"]$time_sch)
```

```
  # D - 15
  sch[sch$a_d == "D"]$op_time1 <- as.ITime(sch[sch$a_d == "D"]$op_time2) -
as.ITime("00:15")

  sch$bool_P <- 0
  sch$park_callsign_A <- "--"
  sch$park_callsign_B <- "--"

  # Per les P
  # PRIMER CAS: te A, D i P
  a <- (sch$tail == sch$tail_ant & sch$a_d_ant == "A")
  sch[ a ]$bool_P <- TRUE

  sch[a]$time1 <- sch[a]$time_ant

  sch[ a ]$park_callsign_A <- sch[ a ]$callsign_ant
  sch[ a ]$park_callsign_B <- sch[ a ]$callsign

  # SEGON CAS: NO TE A
  b <- (sch$tail != sch$tail_ant & sch$a_d == "D")
  sch[ b ] $bool_P <- TRUE

  sch[ b ]$time1 <- as.ITime(time_min)

  sch[ b ]$park_callsign_A <- "SLEEP"
  sch[ b ]$park_callsign_B <- sch[ b ]$callsign

  # TERCER CAS: NO TE D
  c <- (sch$tail != sch$tail_sig & sch$a_d == "A")
  sch[ c] $bool_P <- TRUE

  sch[ c ]$time2 <- as.ITime(time_max)

  sch[ c ] $park_callsign_A <- sch[ c ]$callsign
  sch[ c ] $park_callsign_B <- "SLEEP"


  sch_parking <- sch[sch$bool_P == 1]
  sch_parking$a_d <- "P"

  sch_parking$callsign <-
paste0(sch_parking$park_callsign_A,'_',sch_parking$park_callsign_B)

  # OP_TIME 1 i OP_TIME2

  sch_parking$op_time1 <- as.ITime(sch_parking$time1) + as.ITime("00:15")
  sch_parking$op_time2 <- as.ITime(sch_parking$time2) - as.ITime("00:15")

  # CASOS ESPECIALS:
  # 1. SLEEP-CALLSIGN
  x <- substr(sch_parking$callsign,1,5) == "SLEEP"
```

```
  sch_parking$op_time1[x] <- sch_parking$time1[x]
  # 2. callsign-SLEEP
  length <- nchar(sch_parking$callsign)
  y <- substr(sch_parking$callsign, length-4,length) == "SLEEP"
  sch_parking$op_time2[y] <- sch_parking$time2[y]

  sch_total <- rbind(sch, sch_parking)
  sch_total <- sch_total[order(as.ITime(sch_total[, op_time1])), ]
  sch_total <- sch_total[order(sch_total[, c(tail) ]), ]

  sch <- sch_total[, .(callsign, tail, airp, airl, ac_model, a_d, time_sch, time1,
time2, op_time1, op_time2)]

  # END


  stands <- stands[,contact:=NULL]
  stands <- stands[,T1:=NULL]
  stands <- stands[,T2:=NULL]
  stands <- stands[,sch:=NULL]
  stands <- stands[,shut:=NULL]
  stands <- stands[,no_sch:=NULL]
  stands <- stands[,no_ue:=NULL]
  stands <- stands[,park:=NULL]

  terminal<- data.table(ramp= c("R0", "R1", "R2", "R3", "R9", "R10", "R11", "R12",
"R13", "R14", "R15", "R16", "R17", "R32"), contact=c(0,0,1,0,0,1,1,1,1,1,0,0,0,0),
T1=c(0,0,0,0,1,1,1,1,1,1,1,1,1,0), T2=c(1,1,1,1,0,0,0,0,0,0,0,0,0,0) )
  stands <- merge(stands, terminal, by="ramp")
  border <- data.table(ramp= c("R0", "R1", "R2", "R3", "R9", "R10", "R11", "R12",
"R13", "R14", "R15", "R16", "R17", "R32"), sch=c(4,3,1,3,4,2,1,1,1,1,3,3,3,0),
shut=c(5,4,1,4,5,1,1,2,2,2,4,4,4,0), no_sch=c(5,4,0,4,5,1,1,0,0,1,4,4,4,0),
no_ue=c(5,4,0,4,5,1,1,0,0,1,4,4,4,0), park=c(4,3,3,3,4,3,3,3,3,3,3,3,3,0) )
  stands <- merge(stands,border, by="ramp")

  #R2 te alguns stands per no_sch & no_ue, pero no tots
  stands$no_sch[stands$stand=="118"]=1
  stands$no_ue[stands$stand=="118"]=1
  stands$sch[stands$stand=="118"]=0
  stands$shut[stands$stand=="118"]=0
  stands$no_sch[stands$stand=="119"]=1
  stands$no_ue[stands$stand=="119"]=1
  stands$sch[stands$stand=="119"]=0
  stands$shut[stands$stand=="119"]=0
  stands$no_sch[stands$stand=="119A"]=1
  stands$no_ue[stands$stand=="119A"]=1
  stands$no_sch[stands$stand=="120"]=1
  stands$no_ue[stands$stand=="120"]=1
  stands$sch[stands$stand=="120"]=0
  stands$shut[stands$stand=="120"]=0
  stands$no_sch[stands$stand=="121"]=1
```

```
stands$no_ue[stands$stand=="121"]=1
stands$sch[stands$stand=="121"]=0
stands$shut[stands$stand=="121"]=0
stands$no_sch[stands$stand=="121A"]=1
stands$no_ue[stands$stand=="121A"]=1
stands$sch[stands$stand=="121A"]=0
stands$shut[stands$stand=="121A"]=0
stands$no_sch[stands$stand=="122"]=1
stands$no_ue[stands$stand=="122"]=1
stands$sch[stands$stand=="122"]=0
stands$shut[stands$stand=="122"]=0
stands$no_sch[stands$stand=="123"]=1
stands$no_ue[stands$stand=="123"]=1
stands$sch[stands$stand=="123"]=0
stands$shut[stands$stand=="123"]=0
stands$no_sch[stands$stand=="124"]=1
stands$no_ue[stands$stand=="124"]=1
stands$sch[stands$stand=="124"]=0
stands$shut[stands$stand=="124"]=0
stands$no_sch[stands$stand=="124A"]=1
stands$no_ue[stands$stand=="124A"]=1
stands$sch[stands$stand=="124A"]=0
stands$shut[stands$stand=="124A"]=0
stands$no_sch[stands$stand=="125"]=1
stands$no_ue[stands$stand=="125"]=1
stands$sch[stands$stand=="125"]=0
stands$shut[stands$stand=="125"]=0
stands$no_sch[stands$stand=="126"]=1
stands$sch[stands$stand=="126"]=0
stands$shut[stands$stand=="126"]=0
stands$no_ue[stands$stand=="126A"]=1
stands$no_sch[stands$stand=="126A"]=1
stands$sch[stands$stand=="126A"]=0
stands$shut[stands$stand=="126A"]=0
stands$no_ue[stands$stand=="126"]=1
stands$no_sch[stands$stand=="127"]=1
stands$no_ue[stands$stand=="127"]=1
stands$sch[stands$stand=="127"]=0
stands$shut[stands$stand=="127"]=0
stands$no_sch[stands$stand=="128"]=1
stands$no_ue[stands$stand=="128"]=1
stands$sch[stands$stand=="128"]=0
stands$shut[stands$stand=="128"]=0
stands$no_sch[stands$stand=="128A"]=1
stands$no_ue[stands$stand=="128A"]=1
stands$sch[stands$stand=="128A"]=0
stands$shut[stands$stand=="128A"]=0
stands$no_sch[stands$stand=="129"]=1
stands$no_ue[stands$stand=="129"]=1
stands$sch[stands$stand=="129"]=0
stands$shut[stands$stand=="129"]=0
```

```
  stands$no_ue[stands$stand=="200"]=0
  stands$no_sch[stands$stand=="200"]=0
  stands$sch[stands$stand=="200"]=0
  stands$shut[stands$stand=="200"]=0
  #Requeriments especials

  stands$no_sch[stands$stand=="X1"]=0
  stands$no_ue[stands$stand=="X1"]=0
  stands$sch[stands$stand=="X1"]=0
  stands$shut[stands$stand=="X1"]=0
  stands$park[stands$stand=="X1"]=0

  stands$no_sch[stands$stand=="X2"]=0
  stands$no_ue[stands$stand=="X2"]=0
  stands$sch[stands$stand=="X2"]=0
  stands$shut[stands$stand=="X2"]=0
  stands$park[stands$stand=="X2"]=0

  stands$no_sch[stands$stand=="X3"]=0
  stands$no_ue[stands$stand=="X3"]=0
  stands$sch[stands$stand=="X3"]=0
  stands$shut[stands$stand=="X3"]=0
  stands$park[stands$stand=="X3"]=0

  #Numero de incompatibilitats
  stands <- stands[,num_incomp:=0]
  stands$num_incomp[stands$S1!=""]= 1
  stands$num_incomp[stands$S2!=""]= 2
  stands$num_incomp[stands$S3!=""]= 3



  # ASSIGNAR STANDS A AEROLINIES
  # Primer associo ramp amb terminal
  rampsT2 <- c("R0", "R1", "R2", "R3", "R30")
  rampsT1 <- c("R9", "R10", "R11", "R12", "R13", "R14", "R15", "R16", "R17")
  T1 <- rep(1, length(rampsT1))
  T2 <- rep(2, length(rampsT2))
  ramp_T <- data.table(ramp = c(rampsT1, rampsT2), terminal = c(T1, T2))
  stand_T <- merge(stands,ramp_T, by = "ramp") #amb aixo afegim a stands la
terminal que li pertoca

  airl_T <- merge(stand_T, airl_term, by = "terminal", allow.cartesian = TRUE) #
associem airline amb terminal que ja te ramp i stand associat
  setnames(airl_T,"icao", "airl")
  stand_vol <- merge(sch, airl_T, by = "airl", allow.cartesian = TRUE) # Cada vol
associat a tots els stands als quals pot anar

  # AC_FIT
  area_min <- 4000
```

```
  faa$area <- faa$ac_span*faa$ac_length
  setnames(faa,"icao_code", "ac_max") #canvi de nom
  stand_vol_areamax <- merge(faa,stand_vol, by = "ac_max") #afegim area_max
  setnames(stand_vol_areamax,"area", "area_max") #canvi de nom

  setnames(faa,"ac_max", "ac_model") #canvi de nom
  stand_vol_areanorm <- merge(stand_vol_areamax, faa, by = "ac_model")
  stand_vol_areanorm$ac_fit <- (stand_vol_areanorm$"area" -
area_min)/(stand_vol_areanorm$"area_max" - area_min)

  # TRAIEM COSES INNECESSARIES
  stand_vol_areanorm$"name" <- NULL
  stand_vol_areanorm$"ac_span.x" <- NULL
  stand_vol_areanorm$"ac_span.y" <- NULL
  stand_vol_areanorm$"ac_length.x" <- NULL
  stand_vol_areanorm$"ac_length.y" <- NULL
  stand_vol_areanorm$"mtow.x" <- NULL
  stand_vol_areanorm$"mtow.y" <- NULL
  stand_vol_areanorm$"wct.x" <- NULL
  stand_vol_areanorm$"wct.y" <- NULL
  stand_vol_areanorm$"model.x" <- NULL
  stand_vol_areanorm$"model.y" <- NULL

  ordenat <- stand_vol_areanorm[order(callsign),]
  stand_vol_areanorm <- stand_vol_areanorm[ac_fit<=1] # elimino els avions que no
caben al stand

  ## Assignem a cada aeroport categoria
  airp_info <- airp_info[,border:=0]
  airp_info$border[airp_info$country=="Algeria"]="no_ue"
  airp_info$border[airp_info$country=="Gambia"]="no_ue"
  airp_info$border[airp_info$country=="Egypt"]="no_ue"
  airp_info$border[airp_info$country=="Morocco"]="no_ue"
  airp_info$border[airp_info$country=="Senegal"]="no_ue"
  airp_info$border[airp_info$country=="Tunisia"]="no_ue"
  airp_info$border[airp_info$country=="Argentina"]="no_ue"
  airp_info$border[airp_info$country=="Colombia"]="no_ue"
  airp_info$border[airp_info$country=="United States"]="no_ue"
  airp_info$border[airp_info$country=="Peru"]="no_ue"
  airp_info$border[airp_info$country=="Mexico"]="no_ue"
  airp_info$border[airp_info$country=="Canada"]="no_ue"
  airp_info$border[airp_info$country=="Chile"]="no_ue"
  airp_info$border[airp_info$country=="Brazil"]="no_ue"
  airp_info$border[airp_info$country=="Jordan"]="no_ue"
  airp_info$border[airp_info$country=="Lebanon"]="no_ue"
  airp_info$border[airp_info$country=="United Arab Emirates"]="no_ue"
  airp_info$border[airp_info$country=="Hong Kong"]="no_ue"
  airp_info$border[airp_info$country=="Israel"]="no_ue"
  airp_info$border[airp_info$country=="Pakistan"]="no_ue"
  airp_info$border[airp_info$country=="Qatar"]="no_ue"
  airp_info$border[airp_info$country=="South Korea"]="no_ue"
```

```
airp_info$border[airp_info$country=="China"]="no_ue"
airp_info$border[airp_info$country=="Singapore"]="no_ue"
airp_info$border[airp_info$country=="Iran"]="no_ue"
airp_info$border[airp_info$country=="Russia"]="no_ue"
airp_info$border[airp_info$country=="Georgia"]="no_ue"
airp_info$border[airp_info$country=="Croatia"]="no_ue"
airp_info$border[airp_info$country=="Romania"]="no_ue"
airp_info$border[airp_info$country=="Bulgaria"]="no_ue"
airp_info$border[airp_info$country=="Serbia"]="no_ue"
airp_info$border[airp_info$country=="Moldova"]="no_ue"
airp_info$border[airp_info$country=="Turkey"]="no_ue"
airp_info$border[airp_info$country=="Ukraine"]="no_ue"
airp_info$border[airp_info$country=="Belarus"]="no_ue"
airp_info$border[airp_info$country=="Montenegro"]="no_ue"
airp_info$border[airp_info$country=="North Macedonia"]="no_ue"

airp_info$border[airp_info$country=="Spain"]="sch"
airp_info$border[airp_info$country=="Italy"]="sch"
airp_info$border[airp_info$country=="Norway"]="sch"
airp_info$border[airp_info$country=="France"]="sch"
airp_info$border[airp_info$country=="Netherlands"]="sch"
airp_info$border[airp_info$country=="Greece"]="sch"
airp_info$border[airp_info$country=="Portugal"]="sch"
airp_info$border[airp_info$country=="Hungary"]="sch"
airp_info$border[airp_info$country=="Sweden"]="sch"
airp_info$border[airp_info$country=="Switzerland"]="sch"
airp_info$border[airp_info$country=="Poland"]="sch"
airp_info$border[airp_info$country=="Austria"]="sch"
airp_info$border[airp_info$country=="Iceland"]="sch"
airp_info$border[airp_info$country=="Germany"]="sch"
airp_info$border[airp_info$country=="Belgium"]="sch"
airp_info$border[airp_info$country=="Denmark"]="sch"
airp_info$border[airp_info$country=="Finland"]="sch"
airp_info$border[airp_info$country=="Luxembourg"]="sch"
airp_info$border[airp_info$country=="Malta"]="sch"
airp_info$border[airp_info$country=="Czechia"]="sch"
airp_info$border[airp_info$country=="Latvia"]="sch"
airp_info$border[airp_info$country=="Lithuania"]="sch"

airp_info$border[airp_info$country=="Ireland"]="no_sch"
airp_info$border[airp_info$country=="United Kingdom"]="no_sch"
airp_info$border[airp_info$country=="Faroe Islands"]="no_sch"

airp_info$border[airp_info$city=="Madrid"]="shut"

setnames(airp_info, c("icao"), c("airp"))
stand_vol_areanorm<- merge(stand_vol_areanorm, airp_info, by ="airp")

# TREC COSES INNECESSARIES
stand_vol_areanorm$"lat" <- NULL
stand_vol_areanorm$"city" <- NULL
```

```
  stand_vol_areanorm$"country" <- NULL
  stand_vol_areanorm$"tz" <- NULL
  stand_vol_areanorm$"lon" <- NULL
  stand_vol_areanorm$"name" <- NULL

  ##Asignem a cada operacio el valor de border adequat
  stand_vol_areanorm <- stand_vol_areanorm[,border2:=0]
  stand_vol_areanorm$border2[stand_vol_areanorm$border
=="sch"]=stand_vol_areanorm$sch[stand_vol_areanorm$border =="sch"]
  stand_vol_areanorm$border2[stand_vol_areanorm$border
=="no_sch"]=stand_vol_areanorm$no_sch[stand_vol_areanorm$border =="no_sch"]
  stand_vol_areanorm$border2[stand_vol_areanorm$border
=="no_ue"]=stand_vol_areanorm$no_ue[stand_vol_areanorm$border =="no_ue"]
  stand_vol_areanorm$border2[stand_vol_areanorm$border
=="shut"]=stand_vol_areanorm$shut[stand_vol_areanorm$border =="shut"]
  stand_vol_areanorm$border2[stand_vol_areanorm$a_d
=="P"]=stand_vol_areanorm$park[stand_vol_areanorm$a_d =="P"]

  #Canviem numero de border per valor de border
  stand_vol_areanorm <- stand_vol_areanorm[,value_border:=0]
  stand_vol_areanorm$value_border[stand_vol_areanorm$border2==5]=0
  stand_vol_areanorm$value_border[stand_vol_areanorm$border2==4]=20
  stand_vol_areanorm$value_border[stand_vol_areanorm$border2==3]=50
  stand_vol_areanorm$value_border[stand_vol_areanorm$border2==2]=70
  stand_vol_areanorm$value_border[stand_vol_areanorm$border2==1]=90

  stand_vol_areanorm<- stand_vol_areanorm[value_border>0]

  stand_vol_areanorm<-stand_vol_areanorm[,utility:=(0)]
  stand_vol_areanorm$utility =
stand_vol_areanorm$value_border+(10*stand_vol_areanorm$ac_fit) -
(stand_vol_areanorm$num_incomp/10)

  stand_vol_areanorm$time1 <- as.character(stand_vol_areanorm$time1)

  #ID

  unique_tail <- unique(stand_vol_areanorm$tail)
  unique_time1 <- as.character(unique(stand_vol_areanorm$time1))

  id_table <- data.table(rep(unique_tail,length(unique_time1)))
  setnames(id_table,"V1","tail")

  id_table2 <- data.table(rep(unique_time1[1:length(unique_time1)],each =
length(unique_tail)))
  setnames(id_table2,"V1","time1")

  id_1 <- data.table(id_table,id_table2)

  id_1<- id_1[,id_malo:=c(1:nrow(id_1))]
```

```
  final <- merge(stand_vol_areanorm,id_1,by =c("tail","time1"))

  unique_id_malo <- unique(final$id_malo)
  id_bo <- data.table(unique_id_malo)
  setnames(id_bo, "unique_id_malo", "id_malo")
  id_bo <- id_bo[,id:=seq(1:length(unique_id_malo))]

  solution <- merge(final, id_bo, by="id_malo")

  # PREPAREM SOLUTION
  solution$"ac_model" <- NULL
  solution$"ac_max" <- NULL
  solution$"area_max" <- NULL
  solution$"airl" <- NULL
  solution$"airp" <- NULL
  solution$"terminal" <- NULL
  solution$"S1" <- NULL
  solution$"S2" <- NULL
  solution$"S3" <- NULL
  solution$"contact" <- NULL
  solution$"sch" <- NULL
  solution$"shut" <- NULL
  solution$"park" <- NULL
  solution$"no_ue" <- NULL
  solution$"no_sch" <- NULL
  solution$"border" <- NULL
  solution$"T1" <- NULL
  solution$"T2" <- NULL
  solution$"num_incomp" <- NULL
  solution$"area" <- NULL
  solution$"ac_fit" <- NULL
  solution$"border2" <- NULL
  solution$"value_border" <- NULL
  solution$"time_sch" <- NULL
  solution$"id_malo" <- NULL
  setnames(solution,"a_d", "op") #canvi de nom
  setnames(solution,"utility", "util") #canvi de nom
  setcolorder(solution, c("callsign", "id" , "op", "tail", "ramp", "stand",
"op_time1", "op_time2", "time1", "time2", "util"))
  solution <- solution[order(id),]

  return(solution)

}

# Part2
sa_heur <- function(schedule, stand_info){
  #stand_info="LEBL_stands_tofill.csv"
  #schedule= "SCH_10.csv"
  s_util <- stand_util(schedule, stand_info)
```

```
  #s_util <- fread("Manual/stand_util_10.csv")
  Unicos = unique(s_util$id)
  posib=data.table(s_util[,.N,by=id])
  s_util <- merge(s_util,posib, by="id")

  setorder(s_util,id)

  Unicos2= unique(s_util$callsign)

  vec_utility <- rep(0,length(Unicos2))
  vec_id <- rep(0,length(Unicos2))
  vec_callsign <- unique(s_util$callsign)
  vec_stand <- rep("UNO",length(Unicos2))
  vec_op<- rep("UNO",length(Unicos2))
  vec_hora_inicio <- rep(0,length(Unicos2))
  vec_hora_final <- rep(0,length(Unicos2))
  vec_stand<-as.character(vec_stand)
  s_util$stand<-as.character(s_util$stand)
  s_util$op_time1<-as.numeric(as.ITime(s_util$op_time1))
  s_util$op_time2<-as.numeric(as.ITime(s_util$op_time2))

  vec <- data.table(vec_callsign,vec_id,vec_stand, vec_op, vec_hora_inicio,
vec_hora_final, vec_utility)
  stands <- fread(stand_info)
  incomp <- data.table(stands$stand, stands$S1, stands$S2, stands$S3)
  #nrow(s_util)

  # Inicializacion
  k = 0 # Iteraciones para los ID
  j = 0 # Iteraciones dentro de un ID
  jmax = 1

  while (TRUE){

    # Comprobaciones
    if(j>jmax || k==0){
      k = k +1 # Iteracion nueva (Nuevo ID)
      if (k>length(Unicos)){break} # Condicion de finalizacion del loop While
      j = 0 # Reiniciamos el contador dentro de los ID
      Try = 0

      tx = 0 # Reiniciamos el contador de stands no compatibles de la operacion 1
      ty = 0 # Reiniciamos el contador de stands no compatibles de la operacion 2
      t = 0 # Reiniciamos el contador de stands no compatibles de la operacion 3
      standNO = data.table(.x=rep(0,500),.y=rep(0,500),.=rep(0,500)) # Reiniciamos
los stands no compatibles

      # Creamos las combinaciones dentro de un mismo ID
      dif_id<- data.table(s_util[id==k])

      A_id <-dif_id[op=="A"]
```

```
      A_id$tail<- NULL
      A_id$time1<- NULL
      A_id$time2<- NULL
      A_id$ramp<- NULL
      P_id <-dif_id[op=="P"]
      P_id$tail<- NULL
      P_id$time1<- NULL
      P_id$time2<- NULL
      P_id$ramp<- NULL
      D_id <-dif_id[op=="D"]
      D_id$tail<- NULL
      D_id$time1<- NULL
      D_id$time2<- NULL
      D_id$ramp<- NULL

      # Clasificamos seg?n utility
      if(nrow(P_id)==0){
        tram2 <- merge(A_id,D_id, by="id", allow.cartesian = TRUE)
        tram2[,tot_util:=(tram2$util.x + tram2$util.y)]

tram2$tot_util[tram2$stand.x!=tram2$stand.y]=tram2$tot_util[tram2$stand.x!=tram2$st
and.y]-25
        setorder(tram2,-tot_util)
        # Definimos dentro de este ID si tienen parking, arrival o departure
        set_a=1
        set_p=0
        set_d=1
      }
      if(nrow(A_id)==0){
        tram2 <- merge(P_id,D_id, by="id", allow.cartesian = TRUE)
        tram2[,tot_util:=(tram2$util.x + tram2$util.y)]

tram2$tot_util[tram2$stand.x!=tram2$stand.y]=tram2$tot_util[tram2$stand.x!=tram2$st
and.y]-25
        setorder(tram2,-tot_util)
        # Definimos dentro de este ID si tienen parking, arrival o departure
        set_a=0
        set_p=1
        set_d=1
      }
      if(nrow(D_id)==0){
        tram2 <- merge(A_id,P_id, by="id", allow.cartesian = TRUE)
        tram2[,tot_util:=(tram2$util.x + tram2$util.y)]

tram2$tot_util[tram2$stand.x!=tram2$stand.y]=tram2$tot_util[tram2$stand.x!=tram2$st
and.y]-25
        setorder(tram2,-tot_util)
        # Definimos dentro de este ID si tienen parking, arrival o departure
        set_a=1
        set_p=1
        set_d=0
```

```
      }
      if(nrow(A_id)!=0 && nrow(D_id)!=0 && nrow(P_id)!=0){
        tram1 <- merge(A_id,P_id, by="id", allow.cartesian = TRUE)
        tram2 <- merge(tram1,D_id, by="id", allow.cartesian = TRUE)
        tram2[,tot_util:=(tram2$util.x + tram2$util.y + tram2$util)]

tram2$tot_util[tram2$stand.x!=tram2$stand.y]=tram2$tot_util[tram2$stand.x!=tram2$st
and.y]-25

tram2$tot_util[tram2$stand.y!=tram2$stand]=tram2$tot_util[tram2$stand.x!=tram2$stan
d.y]-25
        setorder(tram2,-tot_util)
        # Definimos dentro de este ID si tienen parking, arrival o departure
        set_a=1
        set_p=1
        set_d=1
      }
      jmax = nrow(tram2) # Iteraciones maximas dentro de un ID
    }

    # Iteramos a la siguiente operacion dentro de un mismo ID
    j = j +1
    Try = Try +1
    # Buscamos si estan libres las horas y los stands de la combinacion
    ##############################################################################
    # Comprobamos la operacion 1
    if(set_a==1 | set_p==1 | set_d==1){
      vec_incomp<-incomp$V1[incomp$V2==tram2$stand.x[j] |
incomp$V3==tram2$stand.x[j] | incomp$V4==tram2$stand.x[j]]
      # Comprobamos los stands incompatibles
      if(length(vec_incomp)==0){
        # Si no hay stands incompatibles identificamos los iguales al stand que
estamos analizando
        w1=(vec$vec_stand==tram2$stand.x[j])
        w=data.table(w1)
        w$w1[w$w1==TRUE]=1
        w=w[,id:=seq(1:length(w1))]
        w_bo=w$id[w1==1]
      }else{
        # Si hay stands incompatibles identificaremos estos tambien
        if(length(vec_incomp)>=3){w1=(vec$vec_stand==tram2$stand.x[j] |
vec$vec_stand==vec_incomp[1] | vec$vec_stand==vec_incomp[2]|
vec$vec_stand==vec_incomp[3])}
        if(length(vec_incomp)==2){w1=(vec$vec_stand==tram2$stand.x[j] |
vec$vec_stand==vec_incomp[1]| vec$vec_stand==vec_incomp[2])}
        if(length(vec_incomp)==1){w1=(vec$vec_stand==tram2$stand.x[j] |
vec$vec_stand==vec_incomp[1])}
        w=data.table(w1)
        w$w1[w$w1==TRUE]=1
        w=w[,id:=seq(1:length(w1))]
        w_bo=w$id[w1==1]
```

```
      }
      # Si la hora analizada esta dentro de otra operacion
      v1=(tram2$op_time1.x[j]>=vec$vec_hora_inicio)
      v12=(tram2$op_time2.x[j]<=vec$vec_hora_final)
      v_1=data.table(v1,v12)
      v_1$v1[v_1$v1==TRUE]=1
      v_1$v12[v_1$v12==TRUE]=1
      v_1=v_1[,id:=seq(1:length(v1))]
      v1_bo=v_1$id[v1==1 & v12==1]
      # Si la hora analizada ocupa parte de la hora de otra operacion por el final
      v2= (tram2$op_time1.x[j]>=vec$vec_hora_inicio)
      v22=(tram2$op_time1.x[j]<vec$vec_hora_final)
      v23=(tram2$op_time2.x[j]>=vec$vec_hora_final)
      v_2=data.table(v2,v22,v23)
      v_2$v2[v_2$v2==TRUE]=1
      v_2$v22[v_2$v22==TRUE]=1
      v_2$v23[v_2$v23==TRUE]=1
      v_2=v_2[,id:=seq(1:length(v2))]
      v2_bo=v_2$id[v_2$v2==1 & v_2$v22==1 & v_2$v23==1]
      # Si la hora analizada ocupa parte de la hora de otra operacion por el
principio
      v3 = (tram2$op_time1.x[j]<=vec$vec_hora_inicio)
      v31 = (tram2$op_time2.x[j]<=vec$vec_hora_final)
      v32 = (tram2$op_time2.x[j]>vec$vec_hora_inicio)
      v_3=data.table(v3, v31, v32)
      v_3$v3[v_3$v3==TRUE]=1
      v_3$v31[v_3$v31==TRUE]=1
      v_3$v32[v_3$v32==TRUE]=1
      v_3=v_3[,id:=seq(1:length(v3))]
      v3_bo=v_3$id[v_3$v3==1 & v_3$v31==1 & v_3$v32==1]
      # Si la hora analizada ocupa el total de una operacion
      v4=(tram2$op_time1.x[j]<=vec$vec_hora_inicio)
      v14=(tram2$op_time2.x[j]>=vec$vec_hora_final)
      v_4=data.table(v4,v14)
      v_4$v4[v_4$v4==TRUE]=1
      v_4$v14[v_4$v14==TRUE]=1
      v_4=v_4[,id:=seq(1:length(v4))]
      v4_bo=v_4$id[v4==1 & v14==1]


      v=c(v1_bo,v2_bo,v3_bo,v4_bo)
      if(length(v)==0){v=0}
      # Identificamos aquellos que dan problemas por el Stand (w) o por la Hora (v)
      w_table <- data.table(rep(w_bo,length(v)))
      setnames(w_table,"V1","w")
      v_table <- data.table(rep(v[1:length(v)],each = length(w_bo)))
      setnames(v_table,"V1","v")

      if (length(w_bo)==0){
        comp1 <- data.table(1,2)
      }else{
```

```
        comp1 <- data.table(w_table,v_table)
      }
      # Comprobamos si es posible la operacion 1
      if(is.na(comp1) || any(comp1$w==comp1$v)){
        set_1 = 0
        tx = tx+1
        standNO$.x[tx]=tram2$stand.x[j]
        tram2<-tram2[tram2$stand.x!=standNO$.x[tx]]
        jmax = nrow(tram2)
        j = j-1 # Restamos la operacion recien borrada
      }else{
        set_1 = 1
      }
    }
    ##############################################################################
    # Comprobamos la operacion 2
    if((set_a==1 | set_p==1 | set_d==1) & set_1==1){
      vec_incomp<-incomp$V1[incomp$V2==tram2$stand.y[j] |
incomp$V3==tram2$stand.y[j] | incomp$V4==tram2$stand.y[j]]
      # Comprobamos los stands incompatibles
      if(length(vec_incomp)==0){
        # Si no hay stands incompatibles identificamos los iguales al stand que
estamos analizando
        w1=(vec$vec_stand==tram2$stand.y[j])
        w=data.table(w1)
        w$w1[w$w1==TRUE]=1
        w=w[,id:=seq(1:length(w1))]
        w_bo=w$id[w1==1]
      }else{
        # Si hay stands incompatibles identificaremos estos tambien
        if(length(vec_incomp)>=3){w1=(vec$vec_stand==tram2$stand.y[j] |
vec$vec_stand==vec_incomp[1] | vec$vec_stand==vec_incomp[2]|
vec$vec_stand==vec_incomp[3])}
        if(length(vec_incomp)==2){w1=(vec$vec_stand==tram2$stand.y[j] |
vec$vec_stand==vec_incomp[1]| vec$vec_stand==vec_incomp[2])}
        if(length(vec_incomp)==1){w1=(vec$vec_stand==tram2$stand.y[j] |
vec$vec_stand==vec_incomp[1])}
        w=data.table(w1)
        w$w1[w$w1==TRUE]=1
        w=w[,id:=seq(1:length(w1))]
        w_bo=w$id[w1==1]
      }
      # Si la hora analizada esta dentro de otra operacion
      v1=(tram2$op_time1.y[j]>=vec$vec_hora_inicio)
      v12=(tram2$op_time2.y[j]<=vec$vec_hora_final)
      v_1=data.table(v1,v12)
      v_1$v1[v_1$v1==TRUE]=1
      v_1$v12[v_1$v12==TRUE]=1
      v_1=v_1[,id:=seq(1:length(v1))]
      v1_bo=v_1$id[v1==1 & v12==1]
      # Si la hora analizada ocupa parte de la hora de otra operacion por el final
```

```
      v2= (tram2$op_time1.y[j]>=vec$vec_hora_inicio)
      v22=(tram2$op_time1.y[j]<vec$vec_hora_final)
      v23=(tram2$op_time2.y[j]>=vec$vec_hora_final)
      v_2=data.table(v2,v22,v23)
      v_2$v2[v_2$v2==TRUE]=1
      v_2$v22[v_2$v22==TRUE]=1
      v_2$v23[v_2$v23==TRUE]=1
      v_2=v_2[,id:=seq(1:length(v2))]
      v2_bo=v_2$id[v_2$v2==1 & v_2$v22==1 & v_2$v23==1]
      # Si la hora analizada ocupa parte de la hora de otra operacion por el
principio
      v3 = (tram2$op_time1.y[j]<=vec$vec_hora_inicio)
      v31 = (tram2$op_time2.y[j]<=vec$vec_hora_final)
      v32 = (tram2$op_time2.y[j]>vec$vec_hora_inicio)
      v_3=data.table(v3, v31, v32)
      v_3$v3[v_3$v3==TRUE]=1
      v_3$v31[v_3$v31==TRUE]=1
      v_3$v32[v_3$v32==TRUE]=1
      v_3=v_3[,id:=seq(1:length(v3))]
      v3_bo=v_3$id[v_3$v3==1 & v_3$v31==1 & v_3$v32==1]
      # Si la hora analizada ocupa el total de una operacion
      v4=(tram2$op_time1.y[j]<=vec$vec_hora_inicio)
      v14=(tram2$op_time2.y[j]>=vec$vec_hora_final)
      v_4=data.table(v4,v14)
      v_4$v4[v_4$v4==TRUE]=1
      v_4$v14[v_4$v14==TRUE]=1
      v_4=v_4[,id:=seq(1:length(v4))]
      v4_bo=v_4$id[v4==1 & v14==1]


      v=c(v1_bo,v2_bo,v3_bo,v4_bo)
      if(length(v)==0){v=0}
      # Identificamos aquellos que dan problemas por el Stand (w) o por la Hora (v)
      w_table <- data.table(rep(w_bo,length(v)))
      setnames(w_table,"V1","w")
      v_table <- data.table(rep(v[1:length(v)],each = length(w_bo)))
      setnames(v_table,"V1","v")

      if (length(w_bo)==0){
        comp2 <- data.table(1,2)
      }else{
        comp2 <- data.table(w_table,v_table)
      }
      # Comprobamos si es posible la operacion 2
      if(is.na(comp2) || any(comp2$w==comp2$v)){
        set_2 = 0
        ty = ty+1
        standNO$.y[ty]=tram2$stand.y[j]
        tram2<-tram2[tram2$stand.y!=standNO$.y[ty]]
        jmax = nrow(tram2)
        j = j-1 # Restamos la operacion recien borrada
```

```
      }else{
        # Escribimos si solo habian 2 operaciones para este ID
        if (set_a==1 & set_p==1 & set_d==1){
          set_2 = 1
        }else{
          set_2 = 1
          # Escribimos la operacion 1
          vec$vec_hora_inicio[tram2$callsign.x[j]==vec$vec_callsign]
<-tram2$op_time1.x[j]
          vec$vec_hora_final[tram2$callsign.x[j]==vec$vec_callsign] <-
tram2$op_time2.x[j]
          vec$vec_utility[tram2$callsign.x[j]==vec$vec_callsign] <- tram2$util.x[j]
          vec$vec_id[tram2$callsign.x[j]==vec$vec_callsign] <- tram2$id[j]
          vec$vec_stand[tram2$callsign.x[j]==vec$vec_callsign] <- tram2$stand.x[j]
          vec$vec_callsign[tram2$callsign.x[j]==vec$vec_callsign]<-
tram2$callsign.x[j]
          vec$vec_op[tram2$callsign.x[j]==vec$vec_callsign]<-tram2$op.x[j]
          # Escribimos operacion 2
          vec$vec_hora_inicio[tram2$callsign.y[j]==vec$vec_callsign]
<-tram2$op_time1.y[j]
          vec$vec_hora_final[tram2$callsign.y[j]==vec$vec_callsign] <-
tram2$op_time2.y[j]
          vec$vec_utility[tram2$callsign.y[j]==vec$vec_callsign] <- tram2$util.y[j]
          vec$vec_id[tram2$callsign.y[j]==vec$vec_callsign] <- tram2$id[j]
          vec$vec_stand[tram2$callsign.y[j]==vec$vec_callsign] <- tram2$stand.y[j]
          vec$vec_callsign[tram2$callsign.y[j]==vec$vec_callsign]<-
tram2$callsign.y[j]
          vec$vec_op[tram2$callsign.y[j]==vec$vec_callsign]<-tram2$op.y[j]
          # Dejamos de iterar dentro de un ID
          j = jmax +1
        }
      }
    }
    ##############################################################################
    # Comprobamos la operacion 3
    if(set_a==1 & set_p==1 & set_d==1 & set_1==1 & set_2==1){
      vec_incomp<-incomp$V1[incomp$V2==tram2$stand[j] | incomp$V3==tram2$stand[j] |
incomp$V4==tram2$stand[j]]
      # Comprobamos los stands incompatibles
      if(length(vec_incomp)==0){
        # Si no hay stands incompatibles identificamos los iguales al stand que
estamos analizando
        w1=(vec$vec_stand==tram2$stand[j])
        w=data.table(w1)
        w$w1[w$w1==TRUE]=1
        w=w[,id:=seq(1:length(w1))]
        w_bo=w$id[w1==1]
      }else{
        # Si hay stands incompatibles identificaremos estos tambien
```

```
        if(length(vec_incomp)>=3){w1=(vec$vec_stand==tram2$stand[j] |
vec$vec_stand==vec_incomp[1] | vec$vec_stand==vec_incomp[2]|
vec$vec_stand==vec_incomp[3])}
        if(length(vec_incomp)==2){w1=(vec$vec_stand==tram2$stand[j] |
vec$vec_stand==vec_incomp[1]| vec$vec_stand==vec_incomp[2])}
        if(length(vec_incomp)==1){w1=(vec$vec_stand==tram2$stand[j] |
vec$vec_stand==vec_incomp[1])}
        w=data.table(w1)
        w$w1[w$w1==TRUE]=1
        w=w[,id:=seq(1:length(w1))]
        w_bo=w$id[w1==1]
      }
      # Si la hora analizada esta dentro de otra operacion
      v1=(tram2$op_time1[j]>=vec$vec_hora_inicio)
      v12=(tram2$op_time2[j]<=vec$vec_hora_final)
      v_1=data.table(v1,v12)
      v_1$v1[v_1$v1==TRUE]=1
      v_1$v12[v_1$v12==TRUE]=1
      v_1=v_1[,id:=seq(1:length(v1))]
      v1_bo=v_1$id[v1==1 & v12==1]
      # Si la hora analizada ocupa parte de la hora de otra operacion por el final
      v2= (tram2$op_time1[j]>=vec$vec_hora_inicio)
      v22=(tram2$op_time1[j]<vec$vec_hora_final)
      v23=(tram2$op_time2[j]>=vec$vec_hora_final)
      v_2=data.table(v2,v22,v23)
      v_2$v2[v_2$v2==TRUE]=1
      v_2$v22[v_2$v22==TRUE]=1
      v_2$v23[v_2$v23==TRUE]=1
      v_2=v_2[,id:=seq(1:length(v2))]
      v2_bo=v_2$id[v_2$v2==1 & v_2$v22==1 & v_2$v23==1]
      # Si la hora analizada ocupa parte de la hora de otra operacion por el
principio
      v3 = (tram2$op_time1[j]<=vec$vec_hora_inicio)
      v31 = (tram2$op_time2[j]<=vec$vec_hora_final)
      v32 = (tram2$op_time2[j]>vec$vec_hora_inicio)
      v_3=data.table(v3, v31, v32)
      v_3$v3[v_3$v3==TRUE]=1
      v_3$v31[v_3$v31==TRUE]=1
      v_3$v32[v_3$v32==TRUE]=1
      v_3=v_3[,id:=seq(1:length(v3))]
      v3_bo=v_3$id[v_3$v3==1 & v_3$v31==1 & v_3$v32==1]
      # Si la hora analizada ocupa el total de una operacion
      v4=(tram2$op_time1[j]<=vec$vec_hora_inicio)
      v14=(tram2$op_time2[j]>=vec$vec_hora_final)
      v_4=data.table(v4,v14)
      v_4$v4[v_4$v4==TRUE]=1
      v_4$v14[v_4$v14==TRUE]=1
      v_4=v_4[,id:=seq(1:length(v4))]
      v4_bo=v_4$id[v4==1 & v14==1]
```

```
      v=c(v1_bo,v2_bo,v3_bo,v4_bo)
      if(length(v)==0){v=0}
      # Identificamos aquellos que dan problemas por el Stand (w) o por la Hora (v)
      w_table <- data.table(rep(w_bo,length(v)))
      setnames(w_table,"V1","w")
      v_table <- data.table(rep(v[1:length(v)],each = length(w_bo)))
      setnames(v_table,"V1","v")

      if (length(w_bo)==0){
        comp3 <- data.table(1,2)
      }else{
        comp3 <- data.table(w_table,v_table)
      }
      # Comprobamos si es posible la operacion 3
      if(is.na(comp3) || any(comp3$w==comp3$v)){
        set_3 = 0
        t = t+1
        standNO$.[t]=tram2$stand[j]
        tram2<-tram2[tram2$stand!=standNO$.[t]]
        jmax = nrow(tram2)
        j = j-1 # Restamos la operacion recien borrada
      }else{
        set_3 = 1
      }
      if(set_1==1 & set_2==1 & set_3==1){
        # Escribimos la operacion 1
        vec$vec_hora_inicio[tram2$callsign.x[j]==vec$vec_callsign]
<-tram2$op_time1.x[j]
        vec$vec_hora_final[tram2$callsign.x[j]==vec$vec_callsign] <-
tram2$op_time2.x[j]
        vec$vec_utility[tram2$callsign.x[j]==vec$vec_callsign] <- tram2$util.x[j]
        vec$vec_id[tram2$callsign.x[j]==vec$vec_callsign] <- tram2$id[j]
        vec$vec_stand[tram2$callsign.x[j]==vec$vec_callsign] <- tram2$stand.x[j]
        vec$vec_callsign[tram2$callsign.x[j]==vec$vec_callsign]<-
tram2$callsign.x[j]
        vec$vec_op[tram2$callsign.x[j]==vec$vec_callsign]<-tram2$op.x[j]
        # Escribimos operacion 2
        vec$vec_hora_inicio[tram2$callsign.y[j]==vec$vec_callsign]
<-tram2$op_time1.y[j]
        vec$vec_hora_final[tram2$callsign.y[j]==vec$vec_callsign] <-
tram2$op_time2.y[j]
        vec$vec_utility[tram2$callsign.y[j]==vec$vec_callsign] <- tram2$util.y[j]
        vec$vec_id[tram2$callsign.y[j]==vec$vec_callsign] <- tram2$id[j]
        vec$vec_stand[tram2$callsign.y[j]==vec$vec_callsign] <- tram2$stand.y[j]
        vec$vec_callsign[tram2$callsign.y[j]==vec$vec_callsign]<-
tram2$callsign.y[j]
        vec$vec_op[tram2$callsign.y[j]==vec$vec_callsign]<-tram2$op.y[j]
        # Escribimos operacion 3
        vec$vec_hora_inicio[tram2$callsign[j]==vec$vec_callsign]
<-tram2$op_time1[j]
```

```
        vec$vec_hora_final[tram2$callsign[j]==vec$vec_callsign] <-
tram2$op_time2[j]
        vec$vec_utility[tram2$callsign[j]==vec$vec_callsign] <- tram2$util[j]
        vec$vec_id[tram2$callsign[j]==vec$vec_callsign] <- tram2$id[j]
        vec$vec_stand[tram2$callsign[j]==vec$vec_callsign] <- tram2$stand[j]
        vec$vec_callsign[tram2$callsign[j]==vec$vec_callsign]<- tram2$callsign[j]
        vec$vec_op[tram2$callsign[j]==vec$vec_callsign]<-tram2$op[j]
        # Dejamos de iterar dentro de un ID
        j = jmax +1
      }
    }
  }

  setnames(vec,c("vec_callsign", "vec_op", "vec_stand"), c("callsign", "op",
"stand"))

  sol1 <- data.table(callsign=vec$callsign, op=vec$op, stand=vec$stand)
  sol1<-sol1[stand!="UNO",]
  solution<-sol1
  #fwrite(sol1, file="SA_10.csv")

  return(solution)
}
```

BB - 8

```
sa_heur <- function(schedule, stand_info){
  s_util <- stand_util(schedule, stand_info)
  #s_util <- fread("stand_util_09.csv")
  stands <- fread(stand_info)

  ############# FUNCTIONS #############
  ######### FUNCTION TO ALLOCATE IN Stands_Times

  allocate <- function(Stands_Times, op_time1, op_time2, callsign, id_pow, stand,
stands_all, times_index) {

    op_time_inicio <- as.ITime(op_time1)
    op_time_final <- as.ITime(op_time2)
    op_callsign <- callsign
    op_callsign_id <- id_pow
    op_stand <- stand

    op_stand_id <- match(op_stand, stands_all$stand)

    time1 <- match(as.ITime(op_time_inicio), as.ITime(times_index$times_str))
    time2 <- match(as.ITime(op_time_final), as.ITime(times_index$times_str))
    Stands_Times[time1:time2-1,op_stand_id] <- op_callsign_id

    # Allocate in constraints too!

    S1x <- stands_all[op_stand_id, S1]
    S2x <- stands_all[op_stand_id, S2]
    S3x <- stands_all[op_stand_id, S3]

    if (S1x != "") {
      op_S_id <- match(S1x, stands_all$stand)
      if (is.na(op_S_id) == FALSE) {
        Stands_Times[time1:time2-1, op_S_id] <- op_callsign_id
      }
    }


    if (S2x != ""){
      op_S_id <- match(S2x, stands_all$stand)
      if (is.na(op_S_id) == FALSE) {
        Stands_Times[time1:time2-1, op_S_id] <- op_callsign_id
      }
    }

    if (S3x != "") {
      op_S_id <- match(S3x, stands_all$stand)
      if (is.na(op_S_id) == FALSE) {
        Stands_Times[time1:time2-1, op_S_id] <- op_callsign_id
      }
```

```
  }

  return(Stands_Times)
}

######### FUNCTION TO SEE IF THE STAND IS EMPTY

is_empty_group <- function(stand_, Stands_Times, op_time1, op_time2,
                           callsign_unique, stands_all, times_index) {

  op_time_inicio <- as.ITime(op_time1[1])
  op_time_final <- as.ITime(op_time2[1])
  op_stand <- stand_

  op_stand_id <- match(op_stand, stands_all$stand)

  time1 <- match(as.ITime(op_time_inicio), as.ITime(times_index$times_str))
  time2 <- match(as.ITime(op_time_final), as.ITime(times_index$times_str))
  occupied <- Stands_Times[time1:time2-1, op_stand_id]

  uniques <- unique(c(occupied))
  uniques

  # Look for in constraints too!

  S1 <- stands_all[op_stand_id, S1]
  S2 <- stands_all[op_stand_id, S2]
  S3 <- stands_all[op_stand_id, S3]

  uniques_S1 <- c(0)

  if (S1 != "" ) {
    op_S_id <- match(S1, stands_all$stand)
    if (is.na(op_S_id) == FALSE) {
      occupied <- Stands_Times[time1:time2-1, op_S_id]
      uniques_S1 <- unique(c(occupied))
    }
  }

  uniques_S2 <- c(0)
  if (S2 != "") {
    op_S_id <- match(S2, stands_all$stand)
    if (is.na(op_S_id) == FALSE) {
      occupied <- Stands_Times[time1:time2-1, op_S_id]
      uniques_S2 <- unique(c(occupied))
    }
  }

  uniques_S3 <- c(0)
  if (S3 != "") {
    op_S_id <- match(S3, stands_all$stand)
```

```
      if (is.na(op_S_id) == FALSE) {
        occupied <- Stands_Times[time1:time2-1, op_S_id]
        uniques_S3 <- unique(c(occupied))
      }
    }

    bool_A <- FALSE
    if (length(uniques) == 1)  {
      if (uniques[1] == 0) {
        bool_A <- TRUE
      }
    }

    bool_B <- FALSE
    if (length(uniques_S1) == 1)  {
      if (uniques_S1[1] == 0) {
        bool_B <- TRUE
      }
    }

    bool_C <- FALSE
    if (length(uniques_S2) == 1)  {
      if (uniques_S2[1] == 0) {
        bool_C <- TRUE
      }
    }

    bool_D <- FALSE
    if (length(uniques_S3) == 1)  {
      if (uniques_S3[1] == 0) {
        bool_D <- TRUE
      }
    }

    if ( bool_A == TRUE & bool_B == TRUE & bool_C == TRUE & bool_D == TRUE ) {
      bool <- TRUE
    } else {
      bool <- FALSE
    }

    return(bool)
  }

  ##### FUNCTION TO LOOK FOR EMPTY STANDS (2nd for)
  # Input: stands
  # Output: Retorna True (si est· vacio) or False (si esta ocupado) para cada stand
del input

  obtain_possible_stand <- function (sch_, Stands_Times, stands_all) {

    sch_ <- sch_[order(stand)]
```

```
    sch_ <- sch_[order(util, decreasing = TRUE)]#[1:25,]

    op_time1 <- sch_$op_time1[1]
    op_time2 <- sch_$op_time2[1]
    stands <- sch_$stand

    i<-1
    bools <- rep(FALSE, length(stands))
    ## SECOND FOR
    for (i in seq(1,length(stands))) {
      stand_ <- stands[i]
      bools[i] <- is_empty_group(stand_, Stands_Times, op_time1, op_time2,
                                 callsign_unique, stands_all, times_index)
    }

    stands <- stands[bools]
    return(stands)
  }

  ###### PRE PROCESS
  ### POSSIBILITIES

    ids <- unique(s_util$id)
    ids_all <- s_util$id
    stands_all <- s_util$stand

    determine_possibilities <- function (x, ids_all, stands_all) {
      b <- ids_all==x
      c <- stands_all[b]
      d <- unique(c)
      #print(x)
      #print(ids_all)
      return(length(d))
    }

    poss1 <- lapply(ids, determine_possibilities, ids_all, stands_all)

    possibilities <- data.table(poss1)
    possibilities$id <- seq(from=1, to=length(ids))

    s_util <- merge(s_util, possibilities, by='id')

    rm(poss1, possibilities, ids, ids_all)

  ### COMPUTE TOTAL TIME FOR ID

    s_util_unique <- unique(s_util[, .(id, tail, time1, time2)])
    s_util_unique$total_time <- as.ITime(s_util_unique$time2) -
as.ITime(s_util_unique$time1)
    s_util_unique$total_time_s <- as.integer(as.ITime(s_util_unique$total_time))
    s_util_unique <- s_util_unique[, .(id, total_time, total_time_s)]
```

```
  s_util <- merge(s_util,
                  s_util_unique,
                  by='id')

  s_util <- s_util[order(total_time_s, decreasing=TRUE)]

### STANDS AND TIMES MATRIX

    times1 <- unique(s_util$time1)
    times2 <- unique(s_util$time2)

    max_time_s <- as.integer(as.ITime(max(as.ITime(times2))))
    min_time_s <- as.integer(as.ITime(min(as.ITime(times1))))

    increment <- as.integer(as.ITime("0:05"))

    times_s <- seq(from=min_time_s, to=max_time_s, by=increment)

    times <- as.ITime(times_s)

    stand <- unique(stands$stand)
    stands_all <- data.table(stand)

    stands_all <- merge(stands_all,
                        stands[,.(stand,S1,S2,S3)
                               ], by='stand')

    a <- rep(0,length(stands_all$stand))
    b <- t(rep(0,length(times)))

    c <- t(a%*%b)

    d <- data.table(c)
    Stands_Times <- c

    #Stands_Times <- setnames(d,stands)
    # i = time ; j = stand

    rm(a,b,c,d)

    times_str <- as.character(as.ITime(times_s))

    index <- seq(1,length(times))
    times_index <- data.table(t(rbind( index, times_str)))

### Create id for each callsign
    callsign <- unique(s_util$callsign)
    callsign_unique <- data.table(callsign)
    callsign_unique$id_pow <- seq(from=1,to=length(callsign_unique$callsign))
```

```
      s_util <- merge(s_util,
                      callsign_unique,
                      by='callsign')

      s_util <- s_util[order(total_time_s, decreasing=TRUE)]

    # ORDER
    s_util <- s_util[order(poss, -total_time_s)]

    # Quiero colocar primero los que tiene pocas possibildades
    # Y despues los que tienen parking corto
    poss <- s_util[poss <= 15]
    parking_corto <- s_util[poss > 15]
    parking_corto <- parking_corto[order(total_time_s)]
    s_util <- rbind(poss,parking_corto)

    rm(poss,parking_corto)

    min_total_time <- min(s_util$total_time_s)
    max_total_time <- max(s_util$total_time_s)
    x <- 20000
    ponderado  <- (x - min_total_time) / (max_total_time + min_total_time)

    ##### Incompatibilities computation

    stands_all$S1_inc <- as.integer(substr(stands_all$S1,1,1))
    stands_all$S2_inc <- as.integer(substr(stands_all$S2,1,1))
    stands_all$S3_inc <- as.integer(substr(stands_all$S3,2,2))
    stands_all[is.na(stands_all)] <- 0
    stands_all[S1_inc!=0]$S1_inc <- 1
    stands_all[S2_inc!=0]$S2_inc  <- 1
    stands_all[S3_inc!=0]$S3_inc  <- 1

    stands_all$incompat <- stands_all$S1_inc + stands_all$S2_inc +
stands_all$S3_inc

    stands_all <- stands_all[, .(stand, S1, S2, S3, incompat)]

    ##### Preallocation of solution
    callsign <- callsign_unique$callsign
    solution <- data.table(callsign)
    solution$op <- ""
    solution$stand <- ""

    ##### SOLVER
    num_towings <- 0
    ids_unique <-  unique(s_util$id)
    i <- 1
    # FIRST FOR
    for (i in seq(1,length(ids_unique))) {
```

```
print(paste("i:", i, "=> ID:",ids_unique[i]))
sch_partial <- s_util[s_util$id == ids_unique[i]]

sch_A <- sch_partial[op=="A"]
sch_P <- sch_partial[op=="P"]
sch_D <- sch_partial[op=="D"]

#op_time_inicio <- as.ITime(sch_A$op_time1[1])
#op_time_final <- as.ITime(sch_A$op_time2[1])
#op_stand <- "151A"

if (length(sch_A$id) != 0) {
  stands_poss_A <- obtain_possible_stand(sch_A, Stands_Times, stands_all)
    sch_A <- sch_A[ stand %in% stands_poss_A]
    stands_A <- sch_A$stand
} else { stands_A <- c('') }




stands_poss_P <- obtain_possible_stand(sch_P, Stands_Times, stands_all)
    sch_P <- sch_P[ stand %in% stands_poss_P]
    stands_P <- sch_P$stand

if (length(sch_D$id) != 0) {
  stands_poss_D <- obtain_possible_stand(sch_D, Stands_Times, stands_all)
    sch_D <- sch_D[ stand %in% stands_poss_D]
    stands_D <- sch_D$stand
} else { stands_D <- c('') }

stands_AP <- stands_A[ stands_A %in% stands_P ]
stands_PD <- stands_P[ stands_P %in% stands_D ]

stands_APD <- stands_AP[ stands_AP %in% stands_PD ]

# We punish a parkings with long stays in stands that
# have incompatibilities and reward long parkings that stay in a single stand

# Combinaciones 1

sch_P <- merge(sch_P, stands_all[, .(stand,incompat)], by='stand')

sch_A <- sch_A[order(util, decreasing=TRUE)]
sch_P <- sch_P[order(util, decreasing=TRUE)]
sch_D <- sch_D[order(util, decreasing=TRUE)]

## PUNISH-REWARD ALGORITHM
x <- sch_P$total_time_s[1]
ponderado  <- (x - min_total_time) / (max_total_time + min_total_time)
```

```
factor <- rep(1.3, length(sch_P$incompat))
factor[sch_P$incompat == 0] <- 1.5
factor[sch_P$incompat == 1] <- 0.7
factor[sch_P$incompat == 2] <- 0.2
factor[sch_P$incompat == 3] <- 0.2

#0.1
if (ponderado < 0.5) {
    factor <- 1
}

max(factor)
min(factor)

sch_P$util <- sch_P$util * factor

# Combinaciones 2
sch_A_AP <- sch_A [ stand %in% stands_AP ]
sch_P_AP <- sch_P [ stand %in% stands_AP ]
sch_D_PD <- sch_D [ stand %in% stands_PD ]
sch_P_PD <- sch_P [ stand %in% stands_PD ]

# Combinaciones 3
sch_A_APD <- sch_A [ stand %in% stands_APD ]
sch_P_APD <- sch_D [ stand %in% stands_APD ]
sch_D_APD <- sch_P [ stand %in% stands_APD ]

# AP
sch_AP <- merge( sch_A_AP[,.(stand, id_A=id_pow, util_A=util)],
                 sch_P_AP[,.(stand, id_P=id_pow, util_P=util)],
                 by='stand')
    sch_AP$s_util <- sch_AP$util_A + sch_AP$util_P
    sch_AP <- sch_AP[order(s_util, decreasing=TRUE)]
# PD
sch_PD <- merge( sch_P_PD[,.(stand, id_P=id_pow, util_P=util)],
                 sch_D_PD[,.(stand, id_D=id_pow, util_D=util)],
                 by='stand')
    sch_PD$s_util <- sch_PD$util_P + sch_PD$util_D
    sch_PD <- sch_PD[order(s_util, decreasing=TRUE)]

# APD
sch_APD <- merge( sch_A_APD[,.(stand, id_A=id_pow, util_A=util)],
                  sch_P_APD[,.(stand, id_P=id_pow, util_P=util)],
                  by='stand')
sch_APD <- merge( sch_APD,
                  sch_D_APD[,.(stand, id_D=id_pow, util_D=util)],
                  by='stand')
    sch_APD$s_util <- sch_APD$util_A + sch_APD$util_P + sch_APD$util_D
    sch_APD <- sch_APD[order(s_util, decreasing=TRUE)]

#### Calculate utility
```

```
        towing <- 25
        #towing <- 7

      #### CASES WITH A, P and D
      if ( length(sch_A$id) != 0 & length(sch_D$id) != 0 ) {
        ## CASE 1: APD
          s_util_APD <- sch_APD$s_util[1]
          selected_stands_APD <- c(sch_APD$stand[1], sch_APD$stand[1],
sch_APD$stand[1])

        ## CASE 2: AP + towing + D
          s_util_AP_D <- sch_AP$s_util[1] + sch_D$util[1] - towing
          selected_stands_AP_D <- c(sch_AP$stand[1], sch_AP$stand[1],
sch_D$stand[1])

        ## CASE 3: A + towing + PD
          s_util_A_PD <- sch_PD$s_util[1] + sch_A$util[1] - towing
          selected_stands_A_PD <- c(sch_A$stand[1], sch_PD$stand[1],
sch_PD$stand[1])

        ## CASE 4: A + towing + P + towing + D
          s_util_A_P_D <- sch_A$util[1] + sch_P$util[1] +  sch_D$util[1] - 2*towing
          selected_stands_A_P_D <- c(sch_A$stand[1], sch_P$stand[1],
sch_D$stand[1])

          s_util_APD
          s_util_AP_D
          s_util_A_PD
          s_util_A_P_D

          s_util_summary <- c( s_util_APD,  s_util_AP_D,  s_util_A_PD,
s_util_A_P_D)

          maximum <- max(s_util_summary)

          ID <- match(maximum, s_util_summary)

          if (ID == 1) {
            selected_stands <- selected_stands_APD
          } else if (ID == 2) {
            selected_stands <- selected_stands_AP_D
            num_towings <- num_towings + 1
          } else if (ID == 3) {
            selected_stands <- selected_stands_A_PD
            num_towings <- num_towings + 1
          } else if (ID == 4) {
            selected_stands <- selected_stands_A_P_D
            num_towings <- num_towings + 1
          }
      }
```

```
#### CASES WITH A, P no D

if ( length(sch_A$id) != 0 & length(sch_D$id) == 0 ) {
    ## CASE 1: AP
      s_util_AP <- sch_AP$s_util[1]
      selected_stands_AP <- c(sch_AP$stand[1], sch_AP$stand[1], '')
    ## CASE 2: A + towing + P
      s_util_A_P <- sch_A$util[1] + sch_P$util[1] - towing
      selected_stands_A_P <- c(sch_A$stand[1], sch_P$stand[1], '')

      s_util_summary <- c( s_util_AP,  s_util_A_P)

      maximum <- max(s_util_summary)

      ID <- match(maximum, s_util_summary)

      if (ID == 1) {
        selected_stands <- selected_stands_AP
      } else if (ID == 2) {
        selected_stands <- selected_stands_A_P
        num_towings <- num_towings + 1
      }
}


#### CASES WITH no A, yes P and D
if ( length(sch_A$id) == 0 & length(sch_D$id) != 0 ) {
    ## CASE 1: PD
    s_util_PD <- sch_PD$s_util[1]
    selected_stands_PD <- c('', sch_PD$stand[1], sch_PD$stand[1])
    ## CASE 2: P + towing + D
    s_util_P_D <- sch_P$util[1] + sch_D$util[1] - towing
    selected_stands_P_D <- c('', sch_P$stand[1], sch_D$stand[1])

    s_util_summary <- c( s_util_PD,  s_util_P_D)

    maximum <- max(s_util_summary)

    ID <- match(maximum, s_util_summary)
    ID
    if (ID == 1) {
      selected_stands <- selected_stands_PD
    } else if (ID == 2) {
      selected_stands <- selected_stands_P_D
      num_towings <- num_towings + 1
    }
}
```

```
      final_stand_A <- selected_stands[1]
      final_stand_P <- selected_stands[2]
      final_stand_D <- selected_stands[3]

      print(paste("    A:",final_stand_A,"P:",final_stand_P,"D:",final_stand_D))

      # ALLOCATION
      if (final_stand_A != "") {
          sch_ <- sch_A[stand==final_stand_A]
          Stands_Times <- allocate(Stands_Times, sch_$op_time1, sch_$op_time2,
                                   sch_$callsign,  sch_$id_pow, final_stand_A,
stands_all, times_index)

          solution [callsign==sch_$callsign]$op <- "A"
          solution [callsign==sch_$callsign]$stand <- final_stand_A

      }
      if (final_stand_P != "") {
        sch_ <- sch_P[stand==final_stand_P]
        Stands_Times <- allocate(Stands_Times, sch_$op_time1, sch_$op_time2,
                                 sch_$callsign,  sch_$id_pow, final_stand_P,
stands_all, times_index)

        solution [callsign==sch_$callsign]$op <- "P"
        solution [callsign==sch_$callsign]$stand <- final_stand_P

      }
      if (final_stand_D != "") {
        sch_ <- sch_D[stand==final_stand_D]
        Stands_Times <- allocate(Stands_Times, sch_$op_time1, sch_$op_time2,
                                 sch_$callsign,  sch_$id_pow, final_stand_D,
stands_all, times_index)

        solution [callsign==sch_$callsign]$op <- "D"
        solution [callsign==sch_$callsign]$stand <- final_stand_D
      }

    }

    check <- solution[stand==''']$callsign
    check

    return(solution)

}
```

AD - 10

```
 stand_util <- function(schedule, stand_info){
    sch <- fread(paste0("schedules/", schedule))
    stands <- fread(stand_info)
    airl_term <- fread("data/airl_term.csv")
    airp_info <- fread("data/airp_info.csv")
    faa <- fread("data/FAA-AC.csv")
    # your code
    faa[,"area":=ac_span*ac_length]
    names(airl_term)[1]<-"airl"



    schengen_area<-c("Austria" ,"Belgium" ,"Czech Republic","Denmark" ,"Estonia"
,"Finland","France",

"Germany","Greece","Hungary","Iceland","Italy","Latvia","Liechtenstein",

"Lithuania","Luxembourg","Malta","Netherlands","Norway","Poland","Portugal",

"Slovakia","Slovenia","Spain","Sweden","Switzerland","Czechia","Faroe Islands")
    no_schengen<-c("United Kingdom","Ireland")




    airp_info[country %in% schengen_area, border := "sch"]
    airp_info[!country %in% schengen_area, border := "no_ue"]
    airp_info[country %in% no_schengen, border := "no_sch"]




    names(airp_info)[1]<-"airp"
    names(faa)[1]<-"ac_model"
    sch<-merge(sch,airl_term,"airl")
    sch<-merge(sch,airp_info,"airp")
    sch<-merge(sch,faa,"ac_model")

    sch$border[(sch$city=="Madrid" & (sch$airl=="VLG" | sch$airl=="IBE" ))]<-"shut"

    sch[,ac_model:=NULL]
    sch[,airp:=NULL]
    sch[,airl:=NULL]
    sch[,name.x:=NULL]
    sch[,name.y:=NULL]
    sch[,city:=NULL]
    sch[,country:=NULL]
    sch[,tz:=NULL]
    sch[,lat:=NULL]
```

```
sch[,lon:=NULL]
sch[,model:=NULL]
sch[,ac_span:=NULL]
sch[,ac_length:=NULL]
sch[,mtow:=NULL]
sch[,wct:=NULL]
stands$count[stands$S1!=""]<-1
stands$count[stands$S1==""]<-0
stands$count2[stands$S2!=""]<-1
stands$count2[stands$S2==""]<-0
stands$count3[stands$S3!=""]<-1
stands$count3[stands$S3==""]<-0
stands[,incomp:=count3+count2+count]
stands[,count:=NULL]
stands[,count2:=NULL]
stands[,count3:=NULL]
stands<-merge(stands,faa,by.x="ac_max",by.y="ac_model")
setnames(stands,"area","area_max")
stands[,model:=NULL]
stands[,ac_span:=NULL]
stands[,ac_length:=NULL]
stands[,mtow:=NULL]
stands[,wct:=NULL]

stands$terminal[stands$T1==1]<-1
stands$terminal[stands$T2==1]<-2
stands$terminal[stands$T1==0 & stands$T2==0]<-0


a<-copy(sch)
a[,"first_time":=min(time_sch),by="tail"]
a[,"last_time":=max(time_sch),by="tail"]

sleep_first<-a[(time_sch==first_time & a_d=="D")]
sleep_first$callsign<-paste("SLEEP_",sleep_first$callsign,sep="")
sleep_first[,"op_time1":=as.ITime("06:00")]
sleep_first[,"op_time2":=as.ITime(time_sch)-as.ITime("00:15")]
sleep_first$a_d<-"P"
sleep_first$border<-"park"
sleep_first[,first_time:=NULL]
sleep_first[,last_time:=NULL]


sleep_last<-a[(time_sch==last_time & a_d=="A")]
sleep_last$callsign<-paste(sleep_last$callsign,"_SLEEP",sep="")
sleep_last[,"op_time1":=as.ITime(time_sch)+as.ITime("00:15")]
sleep_last[,"op_time2":=as.ITime("23:00")]
sleep_last$a_d<-"P"
sleep_last$border<-"park"
sleep_last[,first_time:=NULL]
sleep_last[,last_time:=NULL]
```

```
    sch[sch$a_d %in% "D",
c("op_time1","op_time2"):=list((as.ITime(time_sch)-as.ITime("00:15")),(as.ITime(tim
e_sch)))]

    sch[sch$a_d %in% "A",
c("op_time1","op_time2"):=list((as.ITime(time_sch)),(as.ITime(time_sch)+as.ITime("0
0:15")))]


    new_sch<-rbind(sch,sleep_first,sleep_last)[order(tail,op_time1)]

    taula_id<-data.table(id=c(1:(new_sch[,.N]/2)))
    taula_id<-taula_id[rep(seq(nrow(taula_id)),each=2)]
    new_sch[,id:=taula_id$id]

    ids_parking<-new_sch$id[new_sch$a_d=="P"]

    no_park<-new_sch[!id %in% ids_parking] #per saber els ids que no tenen parking


    no_park_arrivals <- no_park[,,on=.("a_d"="A")][order(id)]
    no_park_departures <- no_park[,,on=.("a_d"="D")][order(id)]

    no_park<-merge(no_park_arrivals,no_park_departures,"id")
    id_falten<-no_park$id[(as.ITime(no_park$time_sch.y) -
as.ITime(no_park$time_sch.x)) > as.ITime("00:30")]


    new_park<-new_sch[id %in% id_falten]
    no_park<-no_park[id %in% id_falten]
    new_park<-unique(new_park,by="id")
    new_park$op_time1<-as.ITime(no_park$op_time2.x)
    new_park$op_time2<-as.ITime(no_park$op_time1.y)
    new_park$callsign<-paste(no_park$callsign.x,no_park$callsign.y,sep="_")
    new_park$a_d<-"P"
    new_park$border<-"park"

    new_sch_2<-rbind(new_sch,new_park)[order(tail,op_time1)]

    stands_new<-copy(stands)

    stands_new[,ac_max:=NULL]
    stands_new[,S1:=NULL]
    stands_new[,S2:=NULL]
    stands_new[,S3:=NULL]
    stands_new[,contact:=NULL]
    stands_new[,T1:=NULL]
    stands_new[,T2:=NULL]
```

```
stands_new<-melt(stands_new,c("ramp","stand","incomp","area_max","terminal"))[order
(ramp,stand)]

    setnames(stands_new,"variable","border")
    setnames(stands_new,"value","weight")
    stands_new<-stands_new[weight!=0]


    new_sch_2<-merge(new_sch_2,stands_new,by=c("terminal","border"),allow.cartesian
= TRUE)[order(id)]





    new_sch_2$weight[new_sch_2$weight==1]<-90
    new_sch_2$weight[new_sch_2$weight==2]<-70
    new_sch_2$weight[new_sch_2$weight==3]<-50
    new_sch_2$weight[new_sch_2$weight==4]<-20
    new_sch_2$weight[new_sch_2$weight==5]<-0


    new_sch_2[,time_atm:=NULL]
    new_sch_2[,border:=NULL]

    new_sch_2[,terminal:=NULL]
    new_sch_2[,time_sch:=NULL]

    new_sch_2[,time1:=as.ITime(min(op_time1)),by="id"]
    new_sch_2[,time2:=as.ITime(max(op_time2)),by="id"]
    new_sch_2<-new_sch_2[area<=area_max]
    new_sch_2[,ac_fit:=((area-4000)/(area_max-4000))]
    new_sch_2[,area:=NULL]
    new_sch_2[,area_max:=NULL]
    new_sch_2[,util:=weight+10*ac_fit-(incomp/10)]


taula_final<-data.table("callsign"=new_sch_2$callsign,"id"=new_sch_2$id,"op"=new_sc
h_2$a_d,"tail"=new_sch_2$tail,"ramp"=new_sch_2$ramp,"stand"=new_sch_2$stand,"op_tim
e1"=new_sch_2$op_time1,"op_time2"=new_sch_2$op_time2,"time1"=new_sch_2$time1,"time2
"=new_sch_2$time2,"util"=new_sch_2$util)
    return(taula_final)
  }


sa_heur<-function(schedule,stand_info){

s_util<-stand_util(schedule,stand_info)
```

```
resposta<-s_util

stands <- fread(stand_info)









resposta<-resposta[order(as.ITime(time2)-as.ITime(time1),id,op_time1)]


f<-resposta[,.N,by="id"][order(N)]
f<-merge(f,resposta)[,.(id,N,time2,time1)][,.SD[1],by="id"]
f<-f[order(N,as.ITime(time2)-as.ITime(time1))]
f[,id2:=1:.N]
f[,N:=NULL]
f[,time2:=NULL]
f[,time1:=NULL]
resposta<-merge(resposta,f)
resposta$id<-as.integer(resposta$id2)
resposta[,id2:=NULL]



resposta<-resposta[order(id,op_time1,-util)]

incompatibilitats<-data.table("stand"=stands$stand,"S1"=stands$S1,"S2"=stands$S2,"S
3"=stands$S3)

incompatibilitats<-melt(incompatibilitats,c("stand"))
incompatibilitats[,variable:=NULL]
incompatibilitats<-incompatibilitats[value!=""][order(stand)]
setnames(incompatibilitats,"value","incomp")




stand_ocupat<-function
(ocupat,stand_buscat,op_time1_buscat,op_time2_buscat,incompatibilitats){

  d<-ocupat[ !((op_time1_buscat<op_time1 & op_time2_buscat<=op_time1) |
(op_time1_buscat>=op_time2 & op_time2_buscat>op_time2))]
  treure<-incompatibilitats[stand%in%d$stand]




  free_stands<-stand_buscat[!stand_buscat%in%d$stand]
```

```
  free_stands2<-free_stands[!free_stands%in%treure$incomp]


  return(free_stands2)
}



ocupat<-data.table(resposta[1])
ocupat<-ocupat[0,]




for (i in 1:max(resposta$id)){

  arrival<-resposta[id==i & op=="A"]
  parking<-resposta[id==i & op=="P"]
  departure<-resposta[id==i & op=="D"]
  if (parking[,.N]==0){#Només A i D



free_stand_arrival<-stand_ocupat(ocupat,arrival$stand,arrival$op_time1[1],arrival$o
p_time2[1],incompatibilitats)

free_stand_departure<-stand_ocupat(ocupat,departure$stand,departure$op_time1[1],dep
arture$op_time2[1],incompatibilitats)


    arrival<-arrival[stand %in% free_stand_arrival]
    setnames(arrival,"stand","stand_arrival")
    setnames(arrival,"util","util_arrival")

    departure<-departure[stand %in% free_stand_departure]
    setnames(departure,"stand","stand_departure")
    setnames(departure,"util","util_departure")


combinations<-data.table(arrival[rep(seq(nrow(arrival)),each=departure[,.N])],depar
ture[rep(seq(nrow(departure)),arrival[,.N])])


    combinations[,tows:=0]
    combinations$tows[combinations$stand_arrival!=combinations$stand_departure]<-25
```

```
combinations<-combinations[,util_total:=util_arrival+util_departure-tows][order(-ut
il_total)]

    arrival<-combinations[1,1:11]
    departure<-combinations[1,12:22]


    ocupat<-rbind(ocupat,arrival,departure,use.names=FALSE)



  }

  else if(departure[,.N]==0){#Només A i P_sleep


free_stand_arrival<-stand_ocupat(ocupat,arrival$stand,arrival$op_time1[1],arrival$o
p_time2[1],incompatibilitats)

free_stand_parking<-stand_ocupat(ocupat,parking$stand,parking$op_time1[1],parking$o
p_time2[1],incompatibilitats)


    arrival<-arrival[stand %in% free_stand_arrival]
    setnames(arrival,"stand","stand_arrival")
    setnames(arrival,"util","util_arrival")

    parking<-parking[stand %in% free_stand_parking]
    setnames(parking,"stand","stand_parking")
    setnames(parking,"util","util_parking")


combinations<-data.table(arrival[rep(seq(nrow(arrival)),each=parking[,.N])],parking
[rep(seq(nrow(parking)),arrival[,.N])])


    combinations[,tows:=0]
    combinations$tows[combinations$stand_arrival!=combinations$stand_parking]<-25


combinations<-combinations[,util_total:=util_arrival+util_parking-tows][order(-util
_total)]

    arrival<-combinations[1,1:11]
    parking<-combinations[1,12:22]
    ocupat<-rbind(ocupat,arrival,parking,use.names=FALSE)

  }

  else if (arrival[,.N]==0){#NOmes P_sleep i D
```

```
free_stand_parking<-stand_ocupat(ocupat,parking$stand,parking$op_time1[1],parking$o
p_time2[1],incompatibilitats)

free_stand_departure<-stand_ocupat(ocupat,departure$stand,departure$op_time1[1],dep
arture$op_time2[1],incompatibilitats)



    parking<-parking[stand %in% free_stand_parking]
    setnames(parking,"stand","stand_parking")
    setnames(parking,"util","util_parking")
    departure<-departure[stand %in% free_stand_departure]
    setnames(departure,"stand","stand_departure")
    setnames(departure,"util","util_departure")


combinations<-data.table(parking[rep(seq(nrow(parking)),each=departure[,.N])],depar
ture[rep(seq(nrow(departure)),parking[,.N])])


    combinations[,tows:=0]
    combinations$tows[combinations$stand_departure!=combinations$stand_parking]<-25


combinations<-combinations[,util_total:=util_departure+util_parking-tows][order(-ut
il_total)]

    parking<-combinations[1,1:11]
    departure<-combinations[1,12:22]


    ocupat<-rbind(ocupat,parking,departure,use.names=FALSE)



  }
  else{



free_stand_arrival<-stand_ocupat(ocupat,arrival$stand,arrival$op_time1[1],arrival$o
p_time2[1],incompatibilitats)

free_stand_parking<-stand_ocupat(ocupat,parking$stand,parking$op_time1[1],parking$o
p_time2[1],incompatibilitats)
```

```
free_stand_departure<-stand_ocupat(ocupat,departure$stand,departure$op_time1[1],dep
arture$op_time2[1],incompatibilitats)


    arrival<-arrival[stand %in% free_stand_arrival]
    setnames(arrival,"stand","stand_arrival")
    setnames(arrival,"util","util_arrival")
    parking<-parking[stand %in% free_stand_parking]
    setnames(parking,"stand","stand_parking")
    setnames(parking,"util","util_parking")
    departure<-departure[stand %in% free_stand_departure]
    setnames(departure,"stand","stand_departure")
    setnames(departure,"util","util_departure")


combinations<-data.table(arrival[rep(seq(nrow(arrival)),each=parking[,.N])],parking
[rep(seq(nrow(parking)),arrival[,.N])])


combinations<-data.table(combinations[rep(seq(nrow(combinations)),each=departure[,.
N])],departure[rep(seq(nrow(departure)),combinations[,.N])])

    combinations[,tows1:=0]
    combinations$tows1[combinations$stand_arrival!=combinations$stand_parking]<-25
    combinations[,tows2:=0]

combinations$tows2[combinations$stand_departure!=combinations$stand_parking]<-25
    combinations[,tows:=tows1+tows2]

combinations<-combinations[,util_total:=util_arrival+util_departure+util_parking-to
ws][order(-util_total)]

    arrival<-combinations[1,1:11]
    parking<-combinations[1,12:22]
    departure<-combinations[1,23:33]

    ocupat<-rbind(ocupat,arrival,parking,departure,use.names=FALSE)
  }

}

stand_heuristic<-data.table("callsign"=ocupat$callsign,"op"=ocupat$op,"stand"=ocupa
t$stand)

return (stand_heuristic)
}
```