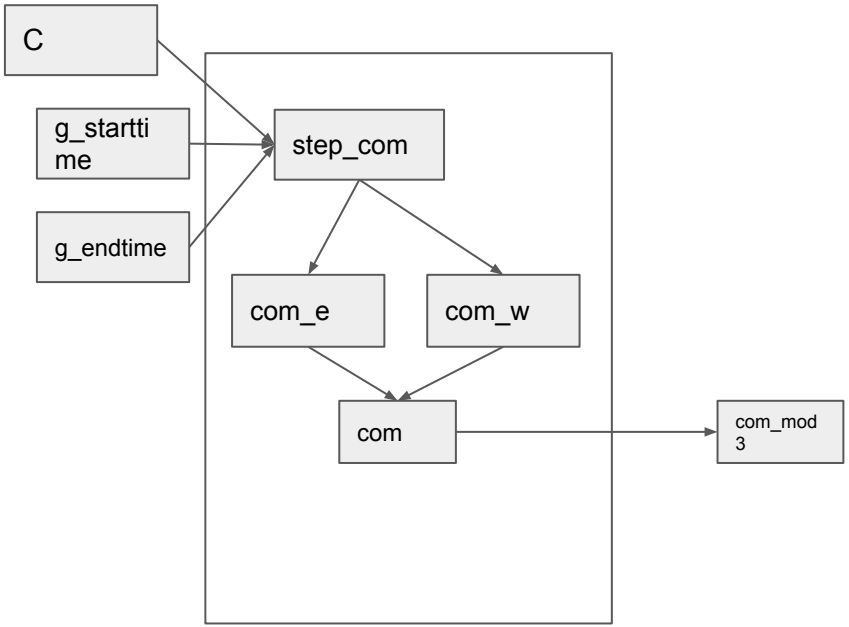
Modules and their relationship

Bilal

datawiz

Variable definitions

2.combinations



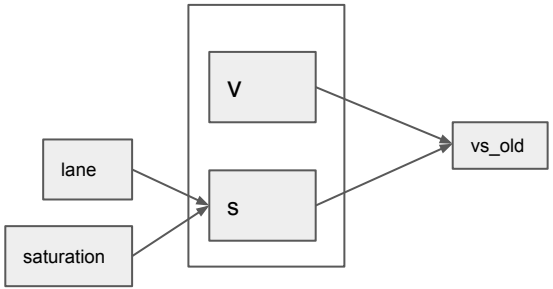

```r
#start time of the green time
g_starttime<- c(0,0,0,0)#random choosing to match the combinations using int1 to 4
#end time of green time of east bound # for overlapping phases we will add the phase for that movement
g_endtime1<- g_starttime[1]+phasing$phase2[1]
g_endtime2<- g_starttime[2]+phasing$phase2[2]
g_endtime3<- g_starttime[3]+phasing$phase1[3]+phasing$phase2[3]
g_endtime33<-g_starttime[3]+phasing$phase2[3]
g_endtime4<- g_starttime[4]+phasing$phase2[4]
g_endtime44<-g_starttime[4]+phasing$phase1[4]+phasing$phase2[4]
g_endtime_e<- c(g_endtime1,g_endtime2,g_endtime3,g_endtime4)
g_endtime_w<- c(g_endtime1,g_endtime2,g_endtime33,g_endtime44)

#combinations of all green split movements over the cycle length.
step_com<- function(C, g_starttime, g_endtime){

  step<- C-(g_endtime-g_starttime)
  step1<- as.vector(1:step[1])
  step2<- as.vector(1:step[2])
  step3<- as.vector(1:step[3])
  step4<- as.vector(1:step[4])
  step_com<- expand.grid(step1,step2,step3,step4)
  return(step_com)
}

com_e<- step_com(C, g_starttime, g_endtime_e)
com_w<- step_com(C, g_starttime, g_endtime_w)
#to minimize the total number of iterations
com<- com_e[!(com_e$Var4>max(com_w$Var4)),]
com<- com[1:10,]
# trying to reduce the number of combhinations effort
com_mod<- com
com_mod$diff12<- com_mod$Var1- com_mod$Var2
com_mod$diff13<- com_mod$Var1- com_mod$Var3
com_mod$diff14<- com_mod$Var1- com_mod$Var4
com_mod2<- com_mod[!duplicated(com_mod[,5:7]),]
com_mod3<- com_mod2[,c(1:4)]
com_mod3<- com_mod3[c(1:10),]
```

Variable definitions

3.Pedestrian time

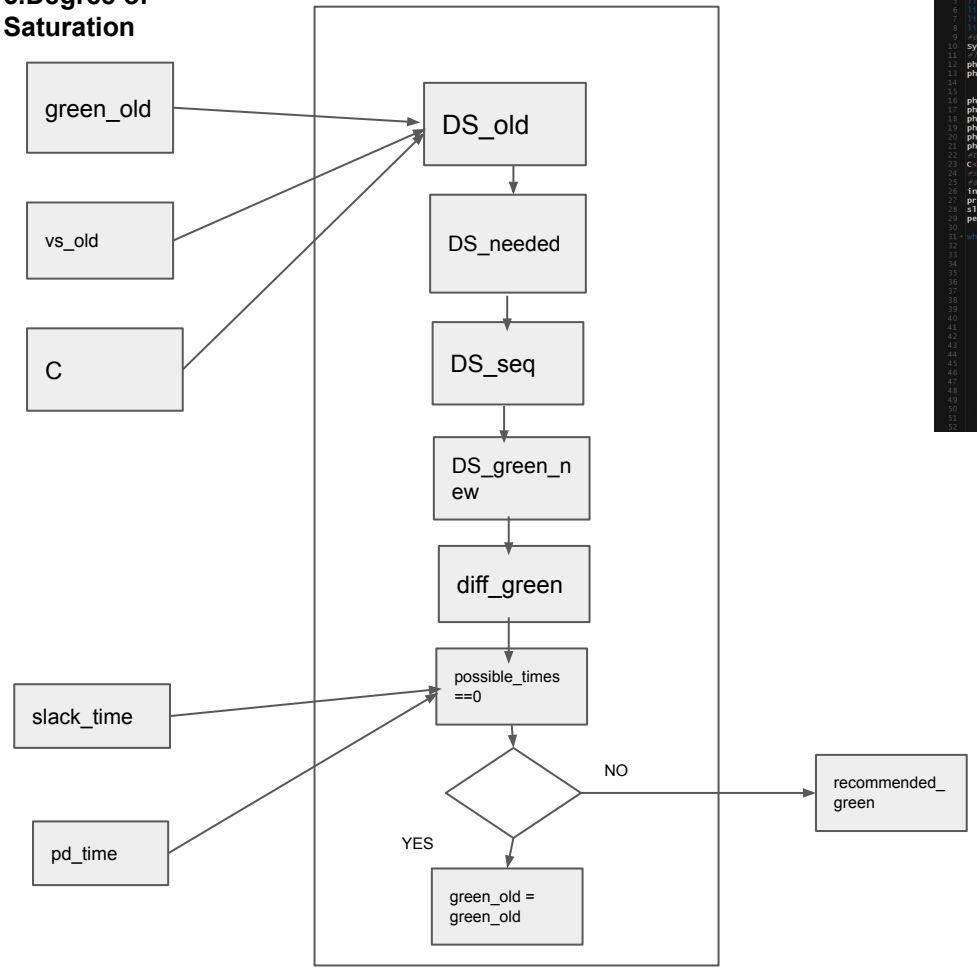
Variable definitions

1. V/S ratio




```
#selected phasing scheme and its corresponding slack time
#check if the difference of new and old green lies inside the slack time obtained
#if not, take another phasing scheme whose difference lies inside the slack time selected
#if yes, change the green time of the suggested phasing to new green and calculate again to find another optimized phasing sche
intersection_num<- 1
DS_phasing_standard<- 2


#existing phasing
phasing$phase1m<- c("3","7","8,3","4,7")
phasing$phase2m<- c("8,4","8,4","8,4","8,4")
phasing$phase3m<- c("5","1","7","1")
phasing$phase4m<- c("0","0","0","2,5")
phasing$phase5m<- c("0","0","0","1,6")

#volume
phasing$ET<- c(2820,2616,1730,2495)
phasing$EL<- c(399,0,560,0)
phasing$WT<- c(1383,1372,1006,1878)
phasing$WL<- c(0,64,91,156)
phasing$NT<- c(0,0,333,0)
phasing$NL<- c(0,193,130,126)
phasing$ST<- c(0,0,660,0)
phasing$SL<- c(137,0,979,0)

#saturation per lane and total lanes
saturation<- data.frame("ET"=rep(2000,4),"WT"=rep(2000,4),"NT"=rep(2000,4),"ST"=rep(2000,4),
                        "EL"=rep(1800,4),"WL"=rep(1800,4),"NL"=rep(1800,4),"SL"=rep(1800,4),
                        "lane_ET" = rep(4,4),"lane_WT" = rep(4,4),"lane_NT" = c(1,2,1,1),"lane_ST" = c(1,0,4,0),
                        "lane_EL" = rep(1,4),"lane_WL" = rep(1,4),"lane_NL" = rep(1,4),"lane_SL" = rep(1,4))

saturation$ET_s<- (saturation$ET*(saturation$lane_ET))
saturation$WT_s<- (saturation$WT*(saturation$lane_WT))
saturation$NT_s<- (saturation$NT*(saturation$lane_NT))
saturation$ST_s<- (saturation$ST*(saturation$lane_ST))
saturation$EL_s<- (saturation$EL*(saturation$lane_EL))
saturation$WL_s<- (saturation$WL*(saturation$lane_WL))
saturation$NL_s<- (saturation$NL*(saturation$lane_NL))
saturation$SL_s<- (saturation$SL*(saturation$lane_SL))

#v/s
vs_old<- data.frame("ET_v/s"=(phasing$ET/saturation$ET_s),"WT_v/s"=(phasing$WT/saturation$WT_s),
                    "NT_v/s"=(phasing$NT/saturation$NT_s),"ST_v/s"=(phasing$ST/saturation$ST_s),
                    "EL_v/s"=(phasing$EL/saturation$EL_s),"WL_v/s"=(phasing$WL/saturation$WL_s),
                    "NL_v/s"=(phasing$NL/saturation$NL_s),"SL_v/s"=(phasing$SL/saturation$SL_s))
```

5.Degree of Saturation




```
#Check the memory size
memory.limit()
memory.limit(size=500000000)
#loading libraries
library(schoolmath)
library(readxl)
library(dplyr)
library(grools)
#error is avoided
Sys.setenv(LANGUAGE='en')
#reading the excel file for data
phasing  <- read_excel("D:/BILAL/4. smart traffic signal system/신호패턴_TOD_최종.xlsx")
phasing<- phasing[phasing[,"교차로명 %in% c("서대전시청네거리","서문고교삼거리","진잠네거리","구봉중삼거리")
                  & phasing[,8]== %in% "주중" & phasing[,"시간 %in% "17:00-21:00",
                  ]
phasing[phase1<- phasing[,"1현시-녹색"]+phasing[,"1현시-황색"]
phasing[phase2<- phasing[,"2현시-녹색"]+phasing[,"2현시-황색"]
phasing[phase3<- phasing[,"3현시-녹색"]+phasing[,"3현시-황색"]
phasing[phase4<- phasing[,"4현시-녹색"]+phasing[,"4현시-황색"]
phasing[phase5<- phasing[,"5현시-녹색"]+phasing[,"5현시-황색"]
phasing<- phasing[, -c(colnames(phasing) %in% c("교차로명","요일","시간","phase1", "phase2", "phase3","phase4","phase5"))]
#first row as header
C<-  rowSums(phasing[1,4:8], na.rm = TRUE)
#select the intersection number and standard phasing to be used (main: through movements)
#ask from user
intersection_num <- 0
proposed_g<- vector(mode = "numeric")
slacktime<- c(10,10,10,10) #cars stack up and down
ped_time<- c(10,10,10,10)

while (intersection_num<5) {
 total_intersections<- 4
 intersection_num<- intersection_num+1
 DS_phasing_standard<- 2

 actual_phase_count_nas<- rowSums(is.na(phasing))
 actual_phase_count<-ncol(phasing[,-c(1,2,3)])-actual_phase_count_nas

 #green as_IS
 green_old<- phasing[,c(4:8)]

 #v/c for as_is situation...hard coded: find a way to not do it this way
 vs_old1<- c(0.222,0.3525,0.076)
 vs_old2<- c(0.0355,0.3270,0.1072)
 vs_old3<- c(0.311,0.2162,0.050,0.544,0.1665)
 vs_old4<- c(0.2347,0.312,0.070)
 #green times
 green_old1<- phasing[1,4:6]
 green_old2<- phasing[2,4:6]
 green_old3<- phasing[3,4:8]
 green_old4<- phasing[4,4:6]
 #DS OF AS_IS
```

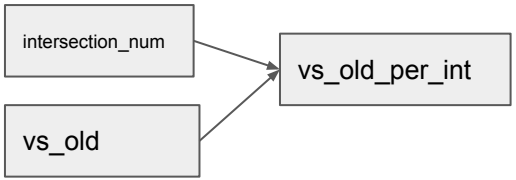
```
 DS_old1<- (vs_old1*C)/green_old1
 DS_old2<- (vs_old2*C)/green_old2
 DS_old3<- (vs_old3*C)/green_old3
 DS_old4<- (vs_old4*C)/green_old4
 DS_old<- smartbind(DS_old1,DS_old2,DS_old3,DS_old4)
 green_old<- smartbind(green_old1,green_old2,green_old3,green_old4)


 #DS needed
 DS_needed<- rowSums(DS_old, na.rm = TRUE)
 na_perrow<- rowSums(is.na(DS_old))
 divider_perphase<- ncol(DS_old)-na_perrow
 DS_needed<- DS_needed/divider_perphase

 #DS_seq<- seq(min(DS_old[intersection_num,DS_phasing_standard],DS_needed[intersection_num]),max(DS_old[intersection_num,DS_phasing_standard],DS_needed[intersection_num]), by = 0.01)
 #green times needed for DS_seq
 my_vector_e1 <- vector(mode="numeric")
 my_vector_e2 <- vector(mode="numeric")

 for (i in 1:length(DS_seq)){
  green2<- ((DS_old[intersection_num,DS_phasing_standard]/DS_seq[i])*(as.numeric(green_old[intersection_num,DS_phasing_standard]))
  diff_green<- as.numeric(green_old[intersection_num,DS_phasing_standard])- green2
  #storing values
  my_vector_e1<- append(my_vector_e1,diff_green)
  my_vector_e2<- append(my_vector_e2,green2)

 }
 DS_green_new<- data.frame("DS"=DS_seq,"new_green"=my_vector_e2, "diff_green" =my_vector_e1,"green_old" =green_old[intersection_num,DS_phasing_standard])
 DS_green_new[diff_green<- DS_green_new[diff_green]*-1]
 #checking if the difference calculated is lesser than the slack time calculated
 #assumed slack time
 if(nrow(DS_green_new[(abs(DS_green_new[diff_green)>slacktime[intersection_num])&(abs(DS_green_new[diff_green)>ped_time[intersection_num]),])==0){
  print("Calculated green time lies outside constraint")
  update<- FALSE
 }else{
  update<- TRUE
  possible_times<- DS_green_new[(abs(DS_green_new[diff_green)<slacktime[intersection_num])&(abs(DS_green_new[diff_green)>ped_time[intersection_num]),]
  print(possible_times)
 }
 recommended_green<- possible_times[which.max(abs(possible_times[diff_green)),]
 print(recommended_green)
 print(intersection_num)
 #storing the values in proposed_g
 proposed_g<- append(proposed_g,ceiling(recommended_green[new_green))
 #updating the green times
 if(update==TRUE){
  phasing[phase2[intersection_num]<- ceiling(recommended_green[new_green)
  #if(intersection_num==1){
   #  phasing[phase1[intersection_num]<- ceiling(recommended_green[new_green)
   #  }
 }
```

Variable definitions

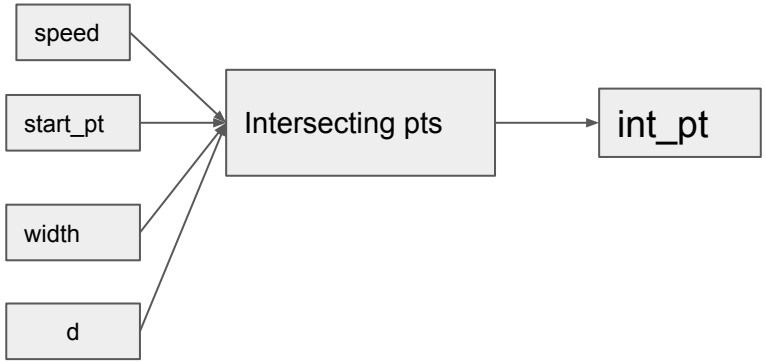
Variable definitions

```
#V/S
vs_old<- data.frame("ET_v/s"=(phasing$ET/saturation$ET_s),"WT_v/s"=(phasing$WT/saturation$WT_s),
                    "NT_v/s"=(phasing$NT/saturation$NT_s),"ST_v/s"=(phasing$ST/saturation$ST_s),
                    "EL_v/s"=(phasing$EL/saturation$EL_s),"WL_v/s"=(phasing$WL/saturation$WL_s),
                    "NL_v/s"=(phasing$NL/saturation$NL_s),"SL_v/s"=(phasing$SL/saturation$SL_s))
vs_old[is.na(vs_old)] = 0
vs_old_vector<- as.matrix(vs_old[intersection_num,])#change to whichever intersection you want
vs_old_vector<- as.vector(vs_old_vector)
vs_old_movs<- c(8,4,6,2,3,7,1,5)
vs_old_per_int<- cbind.data.frame(vs_old_vector,vs_old_movs)
```

4.V/s for each intersection



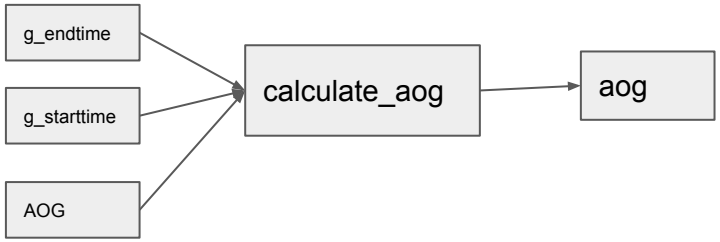
6.Point of intersection function



Variable definitions

```
92
93  #function to find the intersection pts.....
94  intersect_pts <- function(speed,start_pt,width,dist) {
95    A<- matrix(c(speed,1,1,0), nrow = 2, ncol = 2, byrow = TRUE)
96    B<- matrix(c(start_pt+width,dist), nrow = 2, ncol = 1, byrow = TRUE)
97    invA<- solve(A)
98    int_pt<- invA %*% B
99    int_pt<- as.vector(int_pt)
100   return(int_pt)
101 }
```

7.AOG calculation function



Variable definitions

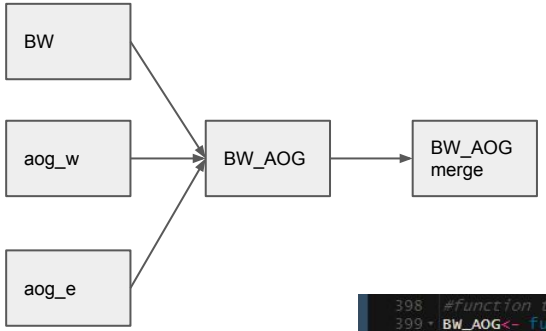
```
#function to find the location of max AOG position and number of veh passing.....................
calculate_aog<- function(g_endtime,g_starttime,AOG){
  vector10<-vector(mode = "numeric")
  # c=1
  while(!g_endtime>C-1){
    g_starttime<- g_starttime+1
    g_endtime<- g_endtime+1
    count_before<- sum(AOG > g_starttime & AOG < g_endtime)
    vector10<- append(vector10,count_before)

  }
  # b_count<- list("place" = which.max(vector10), "Arrival_num" = max(vector10))
  return(vector10)
}
```

Variable definitions

9.Start and end time of green


```
#start time of the green time
g_starttime<- c(0,0,0,0)#random choosing to match the combinations using int1 to 4
#end time of green time of east bound # for overlapping phases we will add the phase for that movement
g_endtime1<- g_starttime[1]+phasing$phase2[1]
g_endtime2<- g_starttime[2]+phasing$phase2[2]
g_endtime3<- g_starttime[3]+phasing$phase1[3]+phasing$phase2[3]
g_endtime33<-g_starttime[3]+phasing$phase2[3]
g_endtime4<- g_starttime[4]+phasing$phase2[4]
g_endtime44<-g_starttime[4]+phasing$phase1[4]+phasing$phase2[4]
g_endtime_e<- c(g_endtime1,g_endtime2,g_endtime3,g_endtime4)
g_endtime_w<- c(g_endtime1,g_endtime2,g_endtime33,g_endtime44)
```

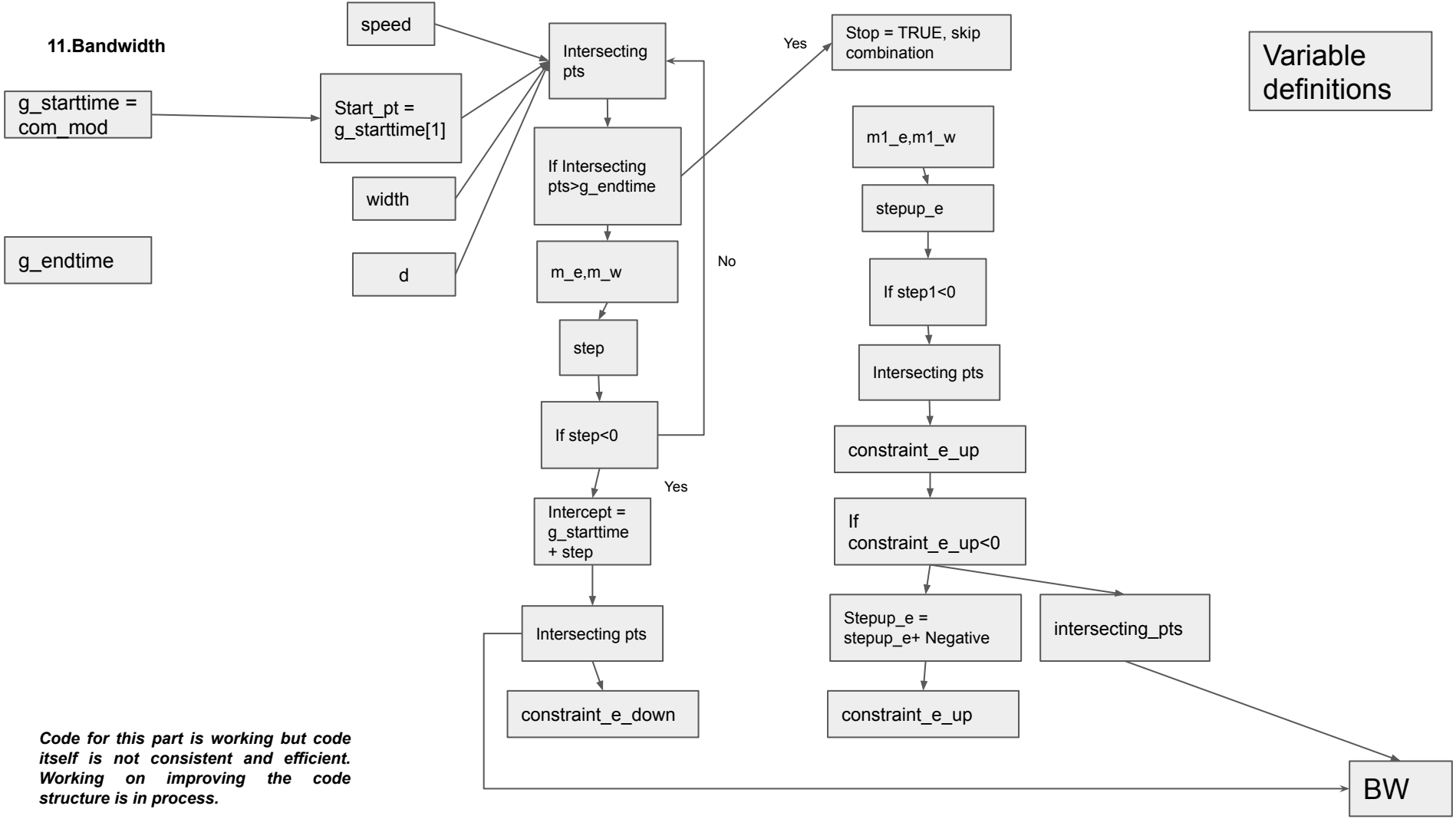
10.Merging AOG and BW



Variable definitions

```r
#function to combine Bw and AOG calculated
BW_AOG<- function(BW,aog_e, aog_w){
  aog_e1<- as.data.frame(unlist(aog_e[1]))
  aog_e2<- as.data.frame(unlist(aog_e[[2]]))
  aog_e3<- as.data.frame(unlist(aog_e[[3]]))
  aog_e4<- as.data.frame(unlist(aog_e[[4]]))

  aog_e1$com1<- 1:length(aog_e[[1]])
  aog_e2$com2<- 1:length(aog_e[[2]])
  aog_e3$com3<- 1:length(aog_e[[3]])
  aog_e4$com4<- 1:length(aog_e[[4]])

  BW_AOGmerge<- merge(x = BW, y = aog_e1, by = "com1", all.x = TRUE)
  BW_AOGmerge<- merge(x = BW_AOGmerge, y = aog_e2, by = "com2", all.x = TRUE)
  BW_AOGmerge<- merge(x = BW_AOGmerge, y = aog_e3, by = "com3", all.x = TRUE)
  BW_AOGmerge<- merge(x = BW_AOGmerge, y = aog_e4, by = "com4", all.x = TRUE)

  aog_w1<- as.data.frame(unlist(aog_w[[1]]))
  aog_w2<- as.data.frame(unlist(aog_w[[2]]))
  aog_w3<- as.data.frame(unlist(aog_w[[3]]))
  aog_w4<- as.data.frame(unlist(aog_w[[4]]))

  aog_w1$com1<- 1:length(aog_w[[1]])
  aog_w2$com2<- 1:length(aog_w[[2]])
  aog_w3$com3<- 1:length(aog_w[[3]])
  aog_w4$com4<- 1:length(aog_w[[4]])

  BW_AOGmerge<- merge(x = BW_AOGmerge, y = aog_w1, by = "com1", all.x = TRUE)
  BW_AOGmerge<- merge(x = BW_AOGmerge, y = aog_w2, by = "com2", all.x = TRUE)
  BW_AOGmerge<- merge(x = BW_AOGmerge, y = aog_w3, by = "com3", all.x = TRUE)
  BW_AOGmerge<- merge(x = BW_AOGmerge, y = aog_w4, by = "com4", all.x = TRUE)
  BW_AOGmerge[, 5:14] <- sapply(BW_AOGmerge[, 5:14], as.numeric)
  return(BW_AOGmerge)
}
```

11.Bandwidth

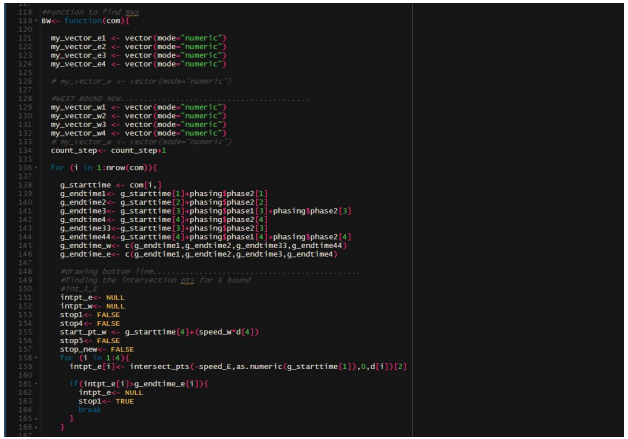


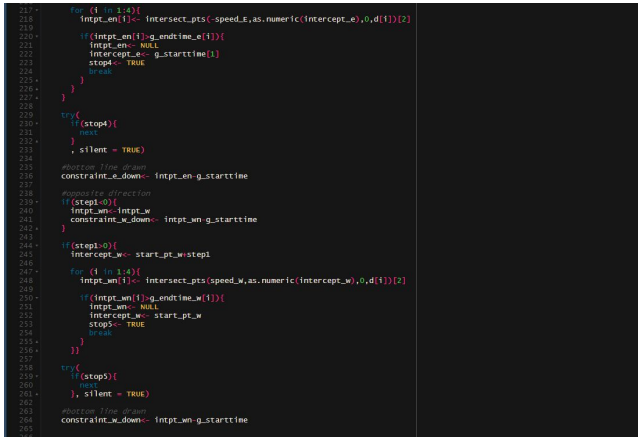
Code for this part is working but code itself is not consistent and efficient. Working on improving the code structure is in process.

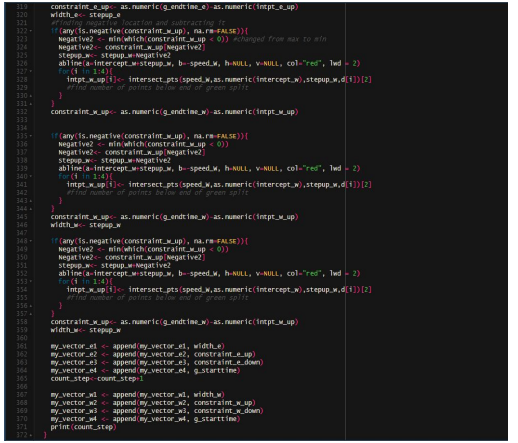
8.Band Width function

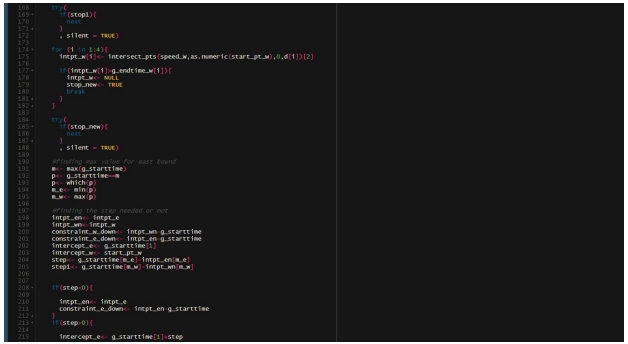
com_mod 3 → BW → BW

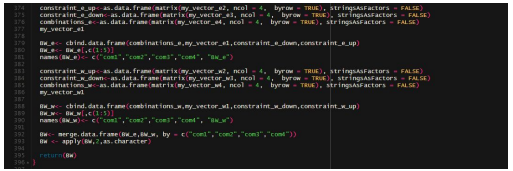
Variable definitions





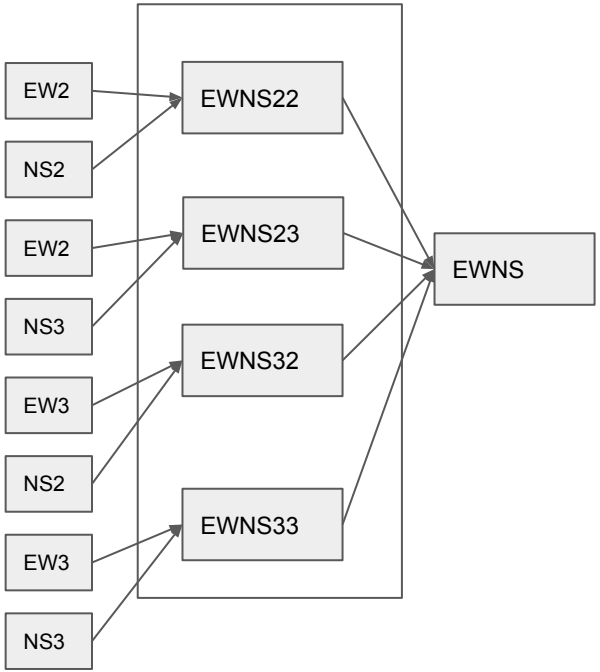






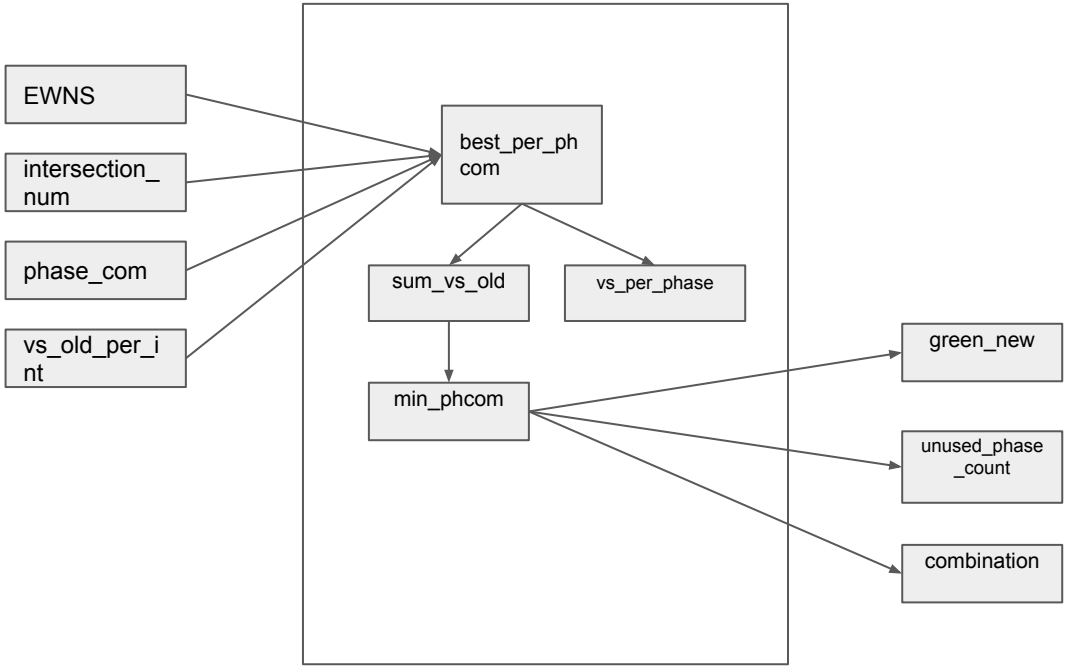
12.Phasing

Variable definitions



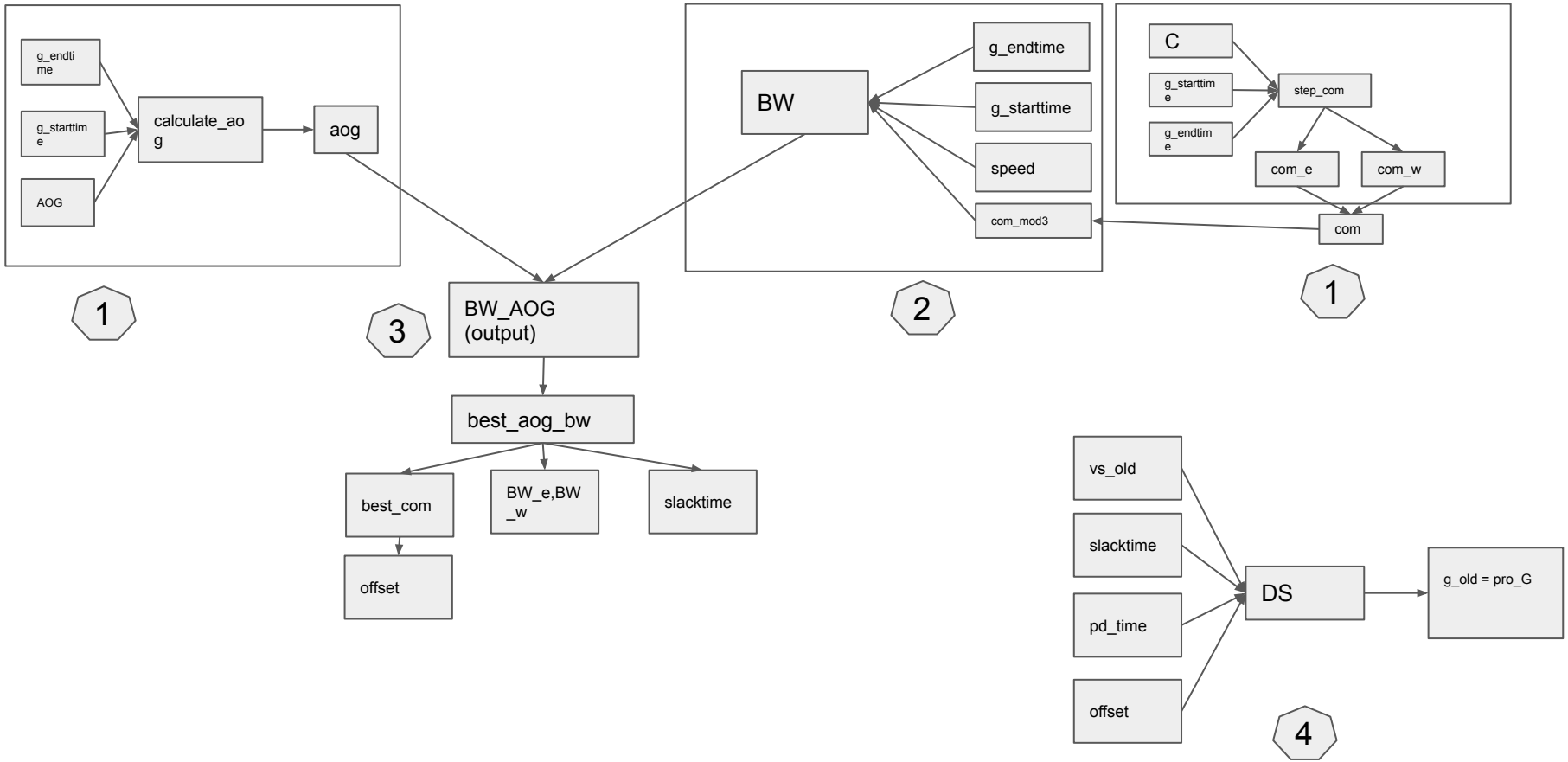

```
#more than two movements not possible per phase as they are conflicting
# m_2ph<- as.data.frame(t(combn(int1,2)))
# m_2ph<- m_2ph[-c(1,6),]
# m1<- paste(m_2ph$V1,"-",m_2ph$V2)
# m2<- as.data.frame(t(combn(m1,2)))
# m3<- as.data.frame(t(combn(m1,3)))

EW2<- data.frame("v1"=c(3),"v2"=c(7),"v3"=c(4),"v4"=c(8))
NS2<- data.frame("v1"=c(1,1),"v2"=c(5,6),"v3"=c(2,2),"v4"=c(6,5))
EW3<- data.frame("v1"=c(3,3),"v2"=c(7,7),"v3"=c(4,3),"v4"=c(7,8),"v5"=c(4,4),"v6"=c(8,8))
NS3<- data.frame("v1"=c(1,1),"v2"=c(5,5),"v3"=c(2,1),"v4"=c(5,6),"v5"=c(2,2),"v6"=c(6,6))

EW2_NS2<-cbind(EW2,NS2)
EWNS_22<- EW2_NS2

EW2_NS3<- cbind(EW2,NS3)
EWNS_23<- EW2_NS3

EW3_NS2<- cbind(rbind(EW3,EW3[nrow(EW3):1,]),NS2)
EWNS_32<- EW3_NS2

EW3_NS3<- cbind(rbind(EW3,EW3[nrow(EW3):1,]),NS3)
EWNS_33<- EW3_NS3

#take values of vs and put them accordingly
sum_vs_old <- vector(mode="numeric")
vs_per_phase <- vector(mode="numeric")
```

12.Phasing



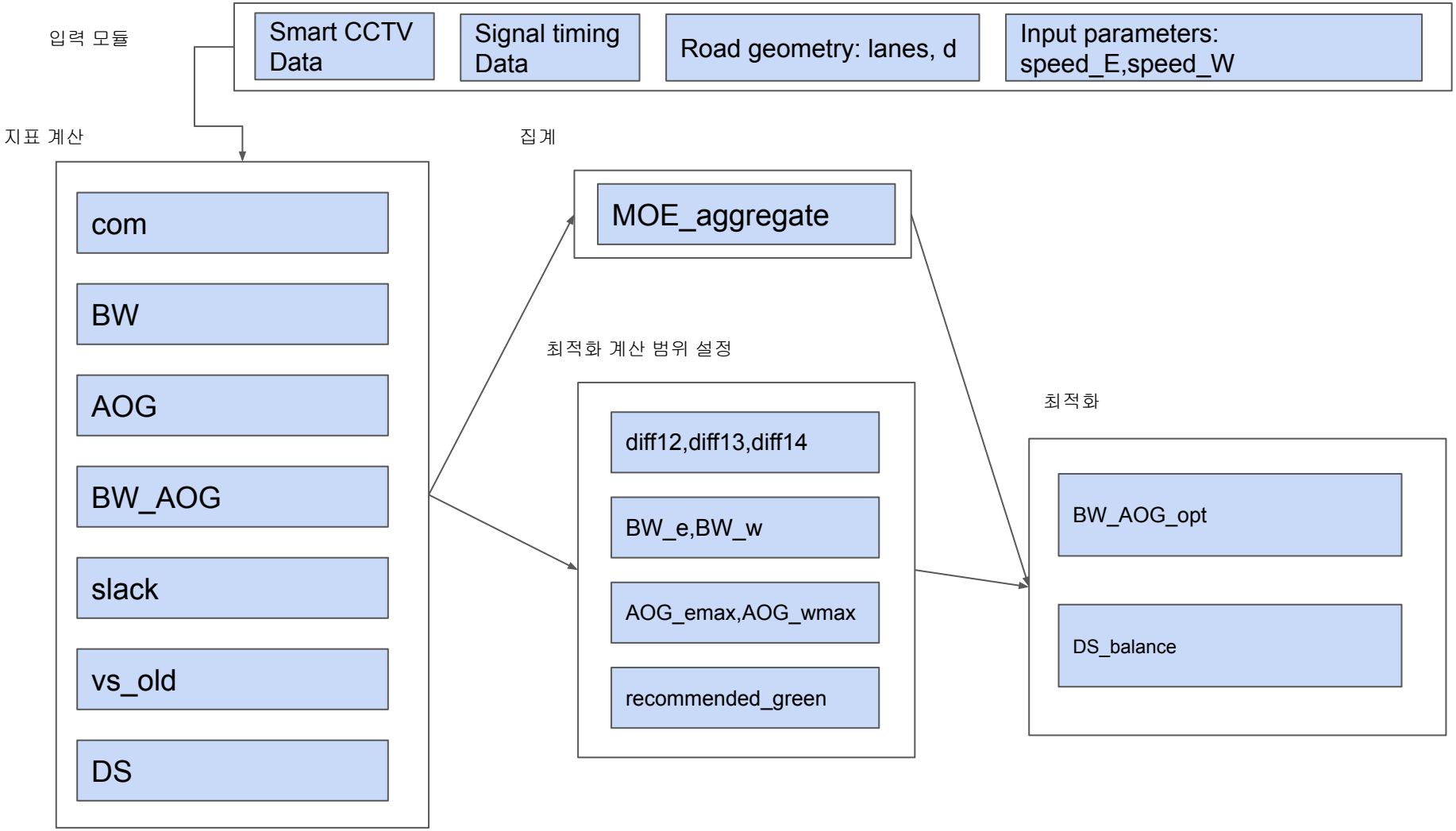
Variable definitions

```r
best_per_phcom<- function(EWNS,vs_old_per_int,phase_com,intersection_num){


  for (i in 1:nrow(EWNS)){

    xxxx<- EWNS[i,]
    xxxx<- as.data.frame(as.vector(as.matrix(xxxx)))
    xxxx$id<- 1:nrow(xxxx)
    yyy<- merge(xxxx,vs_old_per_int, all = TRUE, by.x = "as.vector(as.matrix(xxxx))", by.y = "vs_old_movs")
    yyy<- yyy[order(yyy$id),]

    if(phase_com == 22){
      original_phase_count<- 4 #needs to be calculated
      ph1<- max(yyy[c(T,F,F,F,F,F,F,F),3],yyy[c(F,T,F,F,F,F,F,F),3])
      ph2<- max(yyy[c(F,F,T,F,F,F,F,F),3],yyy[c(F,F,F,T,F,F,F,F),3])
      ph3<- max(yyy[c(F,F,F,F,T,F,F,F),3],yyy[c(F,F,F,F,F,T,F,F),3])
      ph4<- max(yyy[c(F,F,F,F,F,F,T,F),3],yyy[c(F,F,F,F,F,F,F,T),3])
      ph<- c(ph1,ph2,ph3,ph4)
      vs_sum<- sum(ph)
    }else if(phase_com ==23){
      original_phase_count<- 5
      ph1<- max(yyy[c(T,F,F,F,F,F,F,F),3],yyy[c(F,T,F,F,F,F,F,F),3])
      ph2<- max(yyy[c(F,F,T,F,F,F,F,F),3],yyy[c(F,F,F,T,F,F,F,F),3])
      ph3<- max(yyy[c(F,F,F,F,T,F,F,F),3],yyy[c(F,F,F,F,F,T,F,F),3])
      ph4<- max(yyy[c(F,F,F,F,F,F,T,F),3],yyy[c(F,F,F,F,F,F,F,T),3])
      ph5<- max(yyy[c(F,F,F,F,F,F,F,T),3],yyy[c(F,F,F,F,F,F,F,T),3])
      ph<- c(ph1,ph2,ph3,ph4,ph5)
      vs_sum<- sum(ph)
    }else if(phase_com ==32){
      original_phase_count<- 5
      ph1<- max(yyy[c(T,F,F,F,F,F,F,F),3],yyy[c(F,T,F,F,F,F,F,F),3])
      ph2<- max(yyy[c(F,F,T,F,F,F,F,F),3],yyy[c(F,F,F,T,F,F,F,F),3])
      ph3<- max(yyy[c(F,F,F,F,T,F,F,F),3],yyy[c(F,F,F,F,F,T,F,F),3])
      ph4<- max(yyy[c(F,F,F,F,F,F,T,F),3],yyy[c(F,F,F,F,F,F,F,T),3])
      ph5<- max(yyy[c(F,F,F,F,F,F,F,T),3],yyy[c(F,F,F,F,F,F,T,F),3])
      ph<- c(ph1,ph2,ph3,ph4,ph5)
      vs_sum<- sum(ph)
    }else if(phase_com==33){
      original_phase_count<- 6
      ph1<- max(yyy[c(T,F,F,F,F,F,F,F,F,F,F,F),3],yyy[c(F,T,F,F,F,F,F,F,F,F,F,F),3])
      ph2<- max(yyy[c(F,F,T,F,F,F,F,F,F,F,F,F),3],yyy[c(F,F,F,T,F,F,F,F,F,F,F,F),3])
      ph3<- max(yyy[c(F,F,F,F,T,F,F,F,F,F,F,F),3],yyy[c(F,F,F,F,F,T,F,F,F,F,F,F),3])
      ph4<- max(yyy[c(F,F,F,F,F,F,T,F,F,F,F,F),3],yyy[c(F,F,F,F,F,F,F,T,F,F,F,F),3])
      ph5<- max(yyy[c(F,F,F,F,F,F,F,F,T,F,F,F),3],yyy[c(F,F,F,F,F,F,F,F,F,T,F,F),3])
      ph6<- max(yyy[c(F,F,F,F,F,F,F,F,F,F,T,F),3],yyy[c(F,F,F,F,F,F,F,F,F,F,F,T),3])
      ph<- c(ph1,ph2,ph3,ph4,ph5,ph6)
      vs_sum<- sum(ph)
    }
    print(vs_sum)
    sum_vs_old <- append(sum_vs_old, vs_sum)

  }
  vs_per_phase<- as.data.frame(matrix(ph,ncol = original_phase_count, byrow = TRUE), stringsAsFactors = FALSE)
  com_phasing<- cbind.data.frame(EWNS,sum_vs_old,vs_per_phase)
  return(com_phasing)
}
```

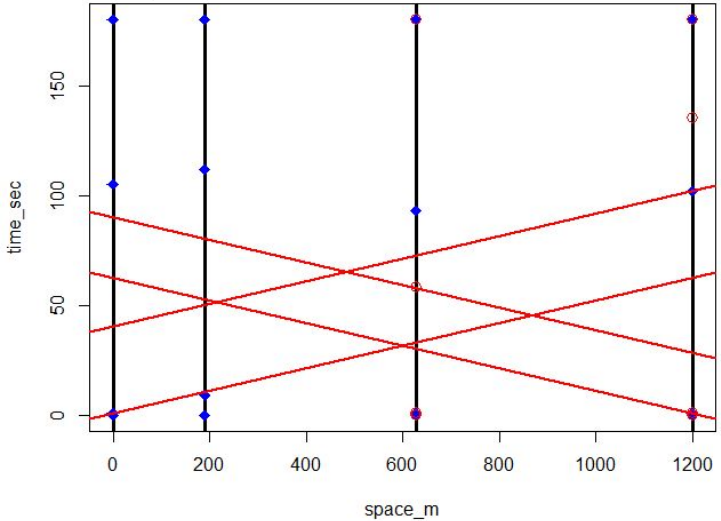


Process 1: 1-(1,2)>3>4>3> end

Process 2: 1-(1,2)>3>4>1-(1,2)>3>end

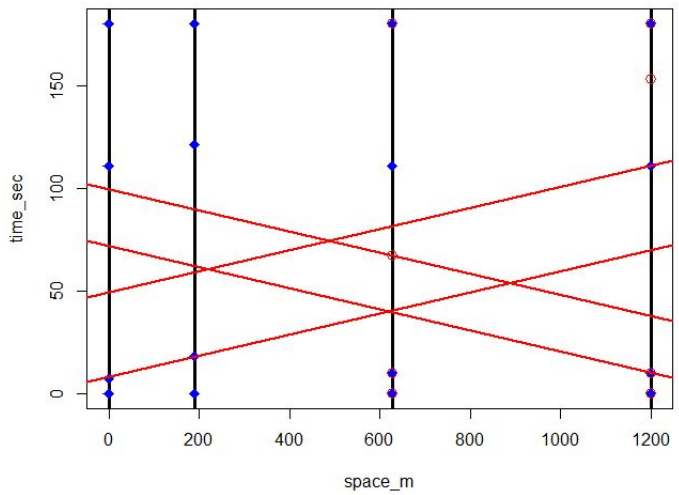

입력 모듈
Smart CCTV Data
Signal timing Data
Road geometry: lanes, d
Input parameters: speed_E,speed_W
지표 계산
com
BW
AOG
BW_AOG
slack
vs_old
DS
집계
MOE_aggregate
최적화 계산 범위 설정
diff12,diff13,diff14
BW_e,BW_w
AOG_emax,AOG_wmax
recommended_green
최적화
BW_AOG_opt
DS_balance

Before updating the green_split for
1000 coms only




```
> print(width_e)
[1] 39.28571
> print(width_w)
[1] 27.58286
> print(slag_down)
[1]  0.000000  1.771429 32.297143 61.714286
> print(slag_up)
[1] 64.71429 61.94286 20.41714  0.00000
> print(slag_down_w)
[1] 61.71429 43.94286 29.41714  0.00000
> print(slag_up_w)
[1]  14.70286  31.47429   0.00000 106.41714
```

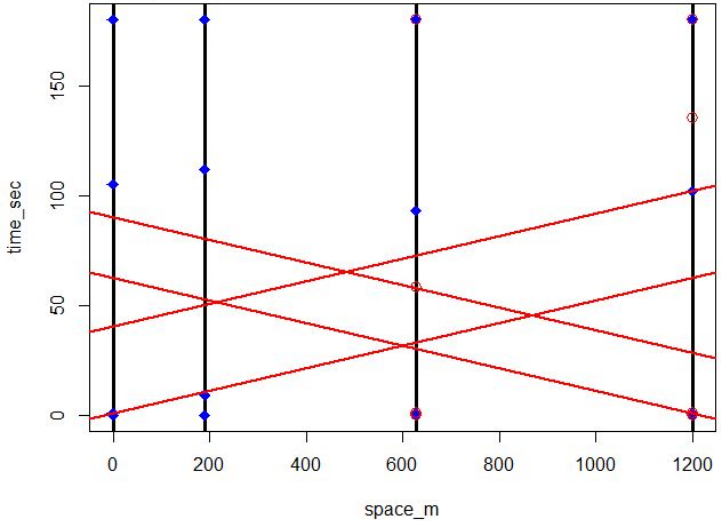
After updating the green_split




```
>   print(width_e)
[1] 41.05714
>   print(width_w)
[1] 27.58286
>   print(slag_down)
[1]  1.228571  0.000000 30.525714 59.942857
>   print(slag_up)
[1] 61.71429 61.94286 29.41714  0.00000
>   print(slag_down_w)
[1] 6.471429e+01 4.394286e+01 2.941714e+01 7.105427e-15
>   print(slag_up_w)
[1]  11.70286  31.47429   0.00000 115.41714
```

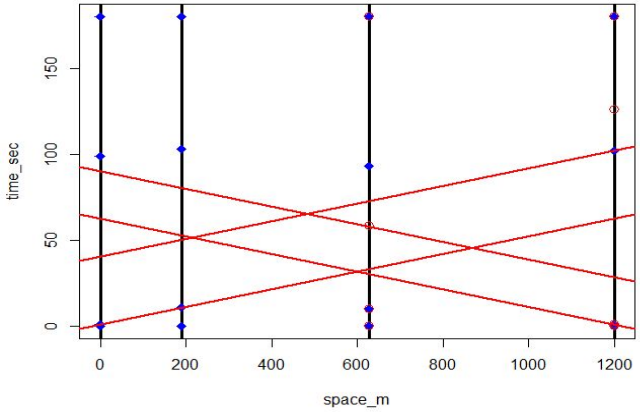
Updated

Before updating the green_split for 1000 coms only




```
> print(width_e)
[1] 39.28571
> print(width_w)
[1] 27.58286
> print(slag_down)
[1]  0.000000  1.771429 32.297143 61.714286
> print(slag_up)
[1] 64.71429 61.94286 20.41714  0.00000
> print(slag_down_w)
[1] 61.71429 43.94286 29.41714  0.00000
> print(slag_up_w)
[1]  14.70286  31.47429   0.00000 106.41714
```

After updating the green_split




```
>   print(width_e)
[1] 39.28571
>   print(width_w)
[1] 27.58286
>   print(slag_down)
[1]  0.00000  0.00000 23.29714 61.71429
>   print(slag_up)
[1] 58.71429 52.94286 20.41714  0.00000
>   print(slag_down_w)
[1] 61.71429 42.17143 20.41714  0.00000
>   print(slag_up_w)
[1]  8.702857 22.474286  0.000000 97.417143
>
```

Filter

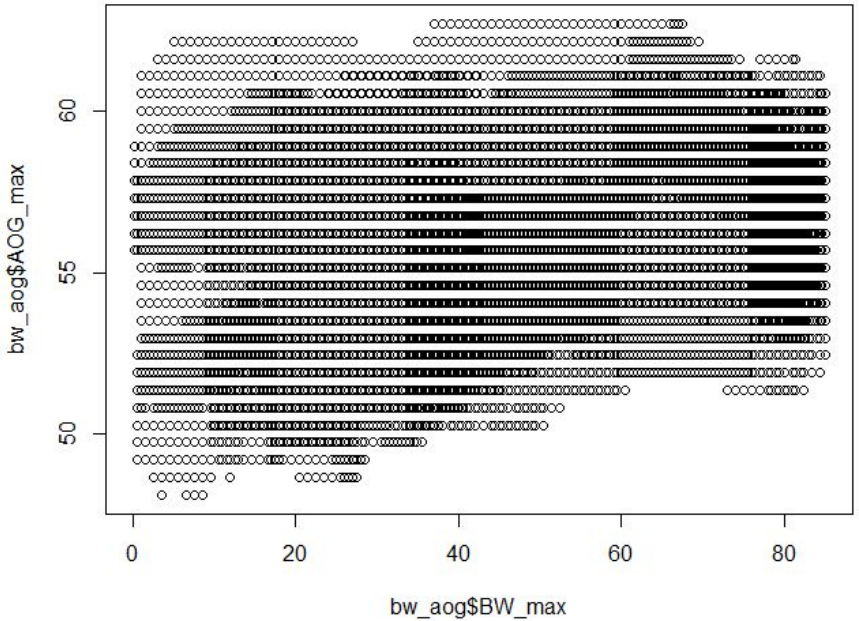
	교차로명	요일	시간	phase1	phase2	phase3	phase4	phase5
1	서대전나들목삼거리	주중	17:00~21:00	53	104	23	NA	NA
2	서일고교삼거리	주중	17:00~21:00	47	103	30	NA	NA
3	진잠네거리	주중	17:00~21:00	35	57	18	50	20
4	구봉중삼거리	주중	17:00~21:00	33	101	46	NA	NA

rearrange_total.R | phasing

Filter

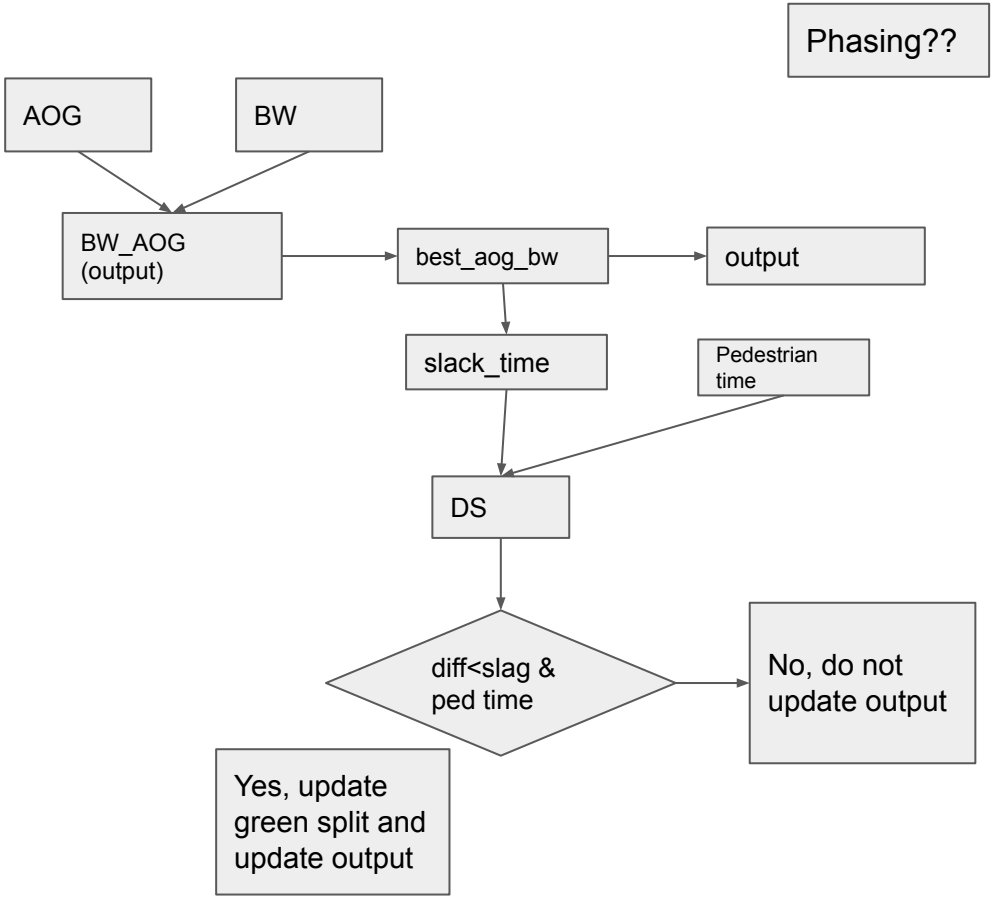
	교차로명	요일	시간	phase1	phase2	phase3	phase4	phase5
1	서대전나들목삼거리	주중	17:00~21:00	53	98	23	NA	NA
2	서일고교삼거리	주중	17:00~21:00	48	94	30	NA	NA
3	진잠네거리	주중	17:00~21:00	44	48	18	50	20
4	구봉중삼거리	주중	17:00~21:00	33	92	46	NA	NA

Final plot




```
> print(rec_com)
       com1 com2 com3 com4
169494    1    9   30    3
> |
```

	off1	off2	off3
1	8	29	2



BW_Input:

1. **Green time**
2. Distance between intersections
3. Phasing scheme
4. Speed for east and west bound

AOG_Input:

1. **Green time**
2. Phasing scheme
3. AOG arrivals on B.S during Cycle time

DS_Input:

1. Green time
2. Volumes
3. Lanes of road
4. Width of road
5. Phasing scheme
6. **Slack time**
7. Standard phasing number