1. 【决策树】：根据当前上海市规定的决策树进行初步决策，决策树如下所示。对于沥青路面、水泥路面，决策树有所不同。

**沥青路面维修决策树**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PCI | RQI | RDI | SRI | PSSI | 推荐对策 |
| 符合 | 符合 | 符合 | 符合 |  | 日常养护 |
| 符合 | 符合 | 符合 | 不符合 |  | 封层 |
| 符合 | 符合 | 不符合 |  |  | 高速公路：二层式罩面；其它：一层式罩面 |
| 符合 | 不符合 |  |  | 符合 | 高速公路：二层式罩面；一级、二级、三级公路：一层式罩面；其它：封层 |
| 符合 | 不符合 |  |  | 不符合 | 局部补强+二层式罩面 |
| 不符合 |  |  |  | 符合 | 对PCI为中等及以上的实施一层式罩面，其它实施二层式罩面 |
| 不符合 |  |  |  | 不符合 | PCI为中等及以上时实施局部补强+二层式罩面，否则实施全面补强+翻修 |

**水泥路面维修决策树**

|  |  |  |  |
| --- | --- | --- | --- |
| PCI | RQI | SRI | 推荐对策 |
| 符合 | 符合 | 符合 | 日常养护 |
| 符合 | 符合 | 不符合 | 刻槽 |
| 符合 | 不符合 | 符合 | 翻修破损板+板顶研磨 |
| 符合 | 不符合 | 不符合 | 翻修破损板+加铺沥青混凝土 |
| 不符合（断板率≤a（高速、一级，a=5%，其它：a=10%） |  |  | 整路段翻修+加铺沥青混凝土 |
| 不符合（断板率＞a（高速、一级，a=5%，其它：a=10%） |  |  | 整路段改造为沥青混凝土路面 |

符合的标准与路面类型，技术等级有关，如下所示。

养护规定值

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 项目 | 高速公路 | 一级公路 | 二级公路 | 三级公路 | 四级公路 |
| PCI | ≥92 | ≥90 | ≥90 | ≥70 | ≥70 |
| RQI | ≥90 | ≥90 | ≥90 | ≥70 | ≥70 |
| RDI | ≥90 | ≥85 | ≥75 | / | / |
| SRI | ≥80 | ≥80 | ≥70 | / | / |
| PSSI | ≥80 | ≥75 | ≥70 | ≥65 | ≥65 |
| DBL | 0.05 | 0.05 | 0.1 | 0.1 | 0.1 |

因此，决策树的运算逻辑如下：（1）缺失值处理，对于PCI,RQI,RDI,SRI,PSSI，缺失值为满分100，对于DBL，缺失值为0；（2）将上表（养护规定值表）记入矩阵或向量；（3）根据养护规定值表判断每个指标符合与否；（4）按维修决策树进行决策。上述过程的运算结果样例及伪代码如下所示。



#决策模块

##养护决策树

maintenance\_measure <- function(PCI,RQI,RDI,SRI,PSSI,DBL,Type,Grade){#6种指标,路面类型,技术等级

#缺失值处理

if(is.na(PCI)){PCI<-100};if(is.na(RQI)){RQI<-100};if(is.na(RDI)){RDI<-100};if(is.na(SRI)){SRI<-100};if(is.na(PSSI)){PSSI<-100};if(is.na(DBL)){DBL<-0}

#养护规定值(只与技术等级有关,分别对应高速、一级、二级、三级、四级)

pci\_threshold <- c(92,90,90,70,70)

rqi\_threshold <- c(90,90,90,70,70)

rdi\_threshold <- c(90,80,75,0,0)

sri\_threshold <- c(80,80,70,0,0)

pssi\_threshold <- c(80,75,70,65,65)

dbl\_threshold <- c(0.05,0.05,0.1,0.1,0.1)

#判断养护规定值是否达到

grade\_level <- switch (Grade,

"高速" = 1,

"一级" = 2,

"二级" = 3,

"三级" = 4,

"四级" = 5,

5

)

if(PCI>=pci\_threshold[grade\_level]){pci\_ok<-TRUE}else{pci\_ok<-FALSE}

if(RQI>=rqi\_threshold[grade\_level]){rqi\_ok<-TRUE}else{rqi\_ok<-FALSE}

if(RDI>=rdi\_threshold[grade\_level]){rdi\_ok<-TRUE}else{rdi\_ok<-FALSE}

if(SRI>=sri\_threshold[grade\_level]){sri\_ok<-TRUE}else{sri\_ok<-FALSE}

if(PSSI>=pssi\_threshold[grade\_level]){pssi\_ok<-TRUE}else{pssi\_ok<-FALSE}

if(DBL<=dbl\_threshold[grade\_level]){dbl\_ok<-TRUE}else{dbl\_ok<-FALSE}#注意此处是小于等于

#养护决策树(其结构仅与路面类型有关)

if(Type=="沥青"){#沥青路面:0代表日常养护,1代表封层,2代表一层式铣刨加罩,3代表二层式铣刨加罩,4代表局部补强+二层式铣刨加罩,5代表全面补强+翻修

if(pci\_ok){

if(rqi\_ok){

if(rdi\_ok){

if(sri\_ok){

0

}else{

1

}

}else{

if(Grade=="高速"){

3

}else{

2

}

}

}else{

if(pssi\_ok){

if(Grade=="高速"){

3

}else if(Grade %in% c("一级","二级","三级")){

2

}else{

1

}

}else{

4

}

}

}else{

if(pssi\_ok){

if(PCI>=70){

2

}else{

3

}

}else{

if(PCI>=70){

4

}else{

5

}

}

}

}else{#水泥路面:0代表日常养护,1代表刻槽,2代表翻修破损板+板顶研磨,3代表翻修破损板+加铺沥青混凝土,4代表整路段翻修+加铺沥青混凝土,5代表整路段改造为沥青混凝土路面

if(pci\_ok){

if(rqi\_ok){

if(sri\_ok){

0

}else{

1

}

}else{

if(sri\_ok){

2

}else{

3

}

}

}else{

if(dbl\_ok){

4

}else{

5

}

}

}

}

data$初始推荐养护对策 <- pmap\_dbl(.l=list(PCI=data$PCI,RQI=data$RQI,RDI=data$RDI,SRI=data$SRI,PSSI=data$PSSI,DBL=NA,Type=data$路面类型,Grade=data$技术等级),.f=maintenance\_measure)

1. 【养护单元动态划分】：

对各养护对策补充附加信息以方便后续运算，附加信息如下所示。

**沥青路面养护对策基本信息**

|  |  |  |  |
| --- | --- | --- | --- |
| **养护对策** | **单价(元/m2）** | **最小连续实施长度(100m的整数倍)** | **养护对策级别** |
| 日常养护 | 7 | 1 | 0 |
| 封层 | 42 | 2 | 1 |
| 一层式铣刨加罩 | 105.9 | 3 | 2 |
| 二层式铣刨加罩 | 207.86 | 3 | 3 |
| 局部补强+二层式铣刨加罩 | 268.5 | 3 | 4 |
| 全面补强+翻修 | 655.82 | 3 | 5 |

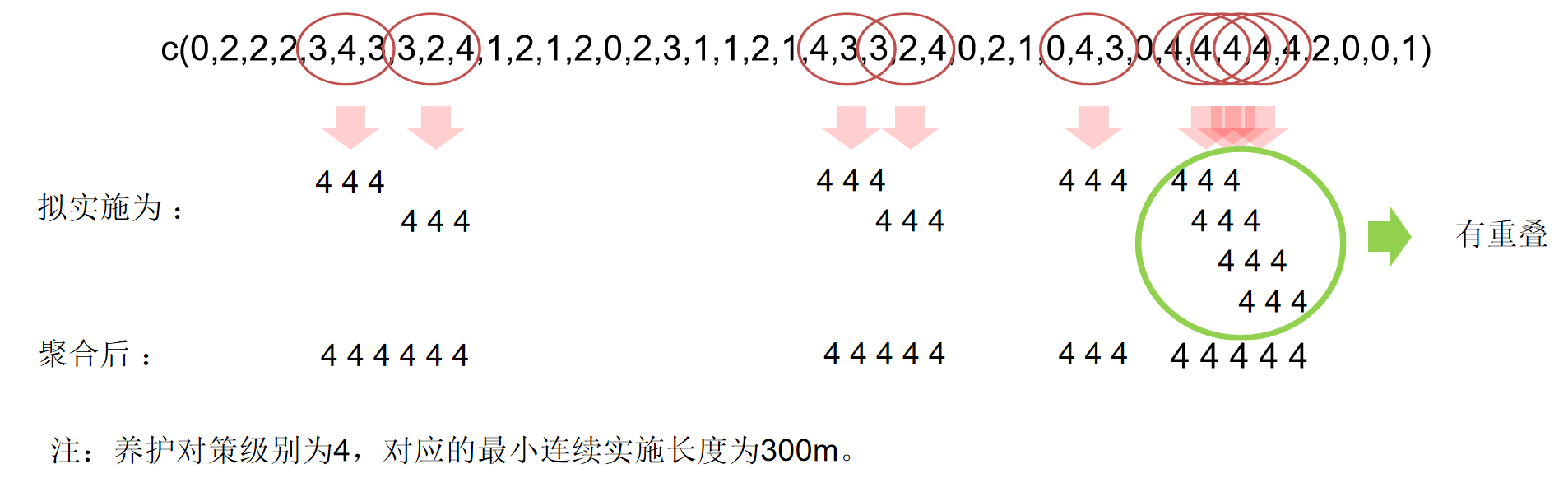
**水泥路面养护对策基本信息**

|  |  |  |  |
| --- | --- | --- | --- |
| **养护对策** | **单价(元/m2）** | **最小连续实施长度(100m的整数倍)** | **养护对策级别** |
| 日常养护 | 7 | 1 | 0 |
| 刻槽 | 9.5 | 2 | 1 |
| 翻修破损板+板顶研磨 | 150 | 3 | 2 |
| 翻修破损板+加铺沥青混凝土 | 240 | 3 | 3 |
| 整路段翻修+加铺沥青混凝土 | 420 | 3 | 4 |
| 整路段改造为沥青混凝土路面 | 430 | 3 | 5 |

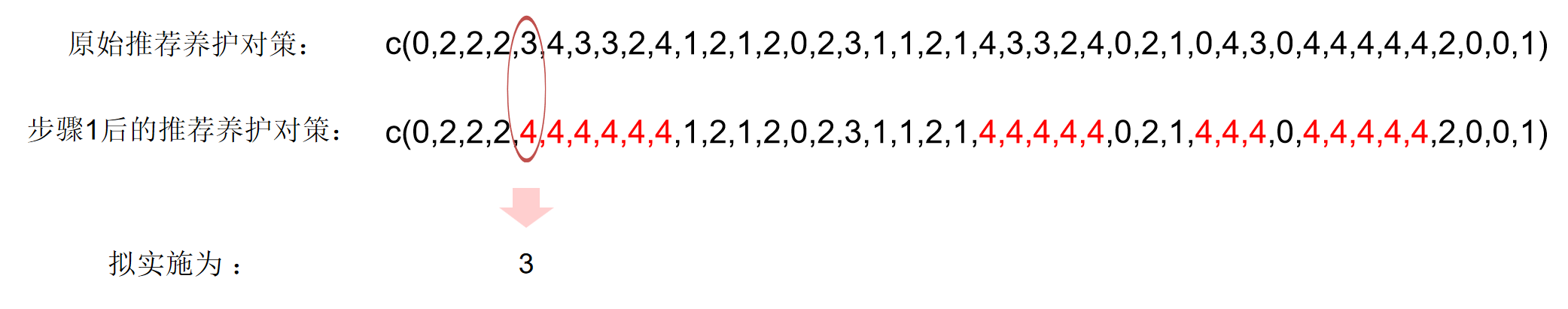
按照【决策树】进行决策后，某示例路段（4.2km，含42个100m单元，100m为最小单元）的推荐对策级别如下所示：

c(0,2,2,2,3,4,3,3,2,4,1,2,1,2,0,2,3,1,1,2,1,4,3,3,2,4,0,2,1,0,4,3,0,4,4,4,4,4,2,0,0,1)

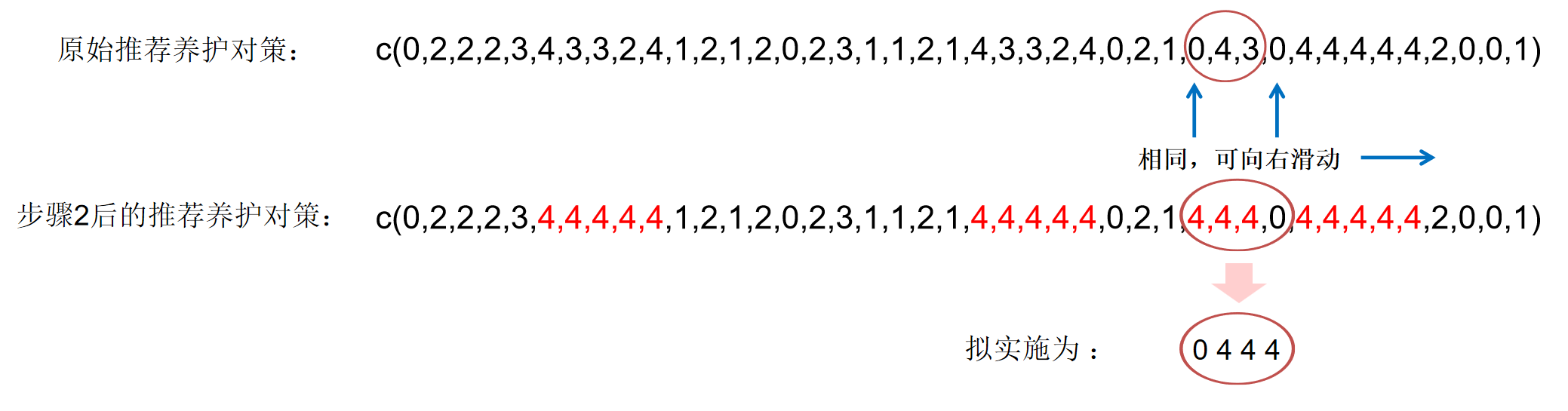
（1）步骤1：从最高推荐养护对策级别开始，对于每一个最小基本单元，根据其最小连续实施长度，确定其邻域范围内实施后增量代价最小的片段，然后自动聚合以消除重复片段。以上述向量第一个4出现的位置为例，其最小连续实施长度为3个最小单元的长度，则由于必须包含这个4出现的位置，可以选择的片段有2 3 4，3 4 3，4 3 3，拟将其替换为4 4 4。显然，根据各养护对策级别的单价，从3 4 3或4 3 3替换为4 4 4，增量代价更小（对策2 3 4或3 4 3 根据养护对策基本信息表，代价分别为105.9+207.86+268.5=582.26，207.86\*2+268.5=684.22，替换后对策为4 4 4，代价为268.5\*3=805.5，从后者替换为4 4 4，代价增量更小）。然后，对于增量代价一样的，比如3 4 3或4 3 3，默认选从左自右（或从上至下）的第一个，即此处选为3 4 3，拟实施为4 4 4。同理，第二个4所在的位置，拟将3 2 4实施为4 4 4。这些拟实施的为级别4的位置可能存在重叠，将位置去重即可完成自动聚合，得到拟实施为级别4的片段，如下所示。



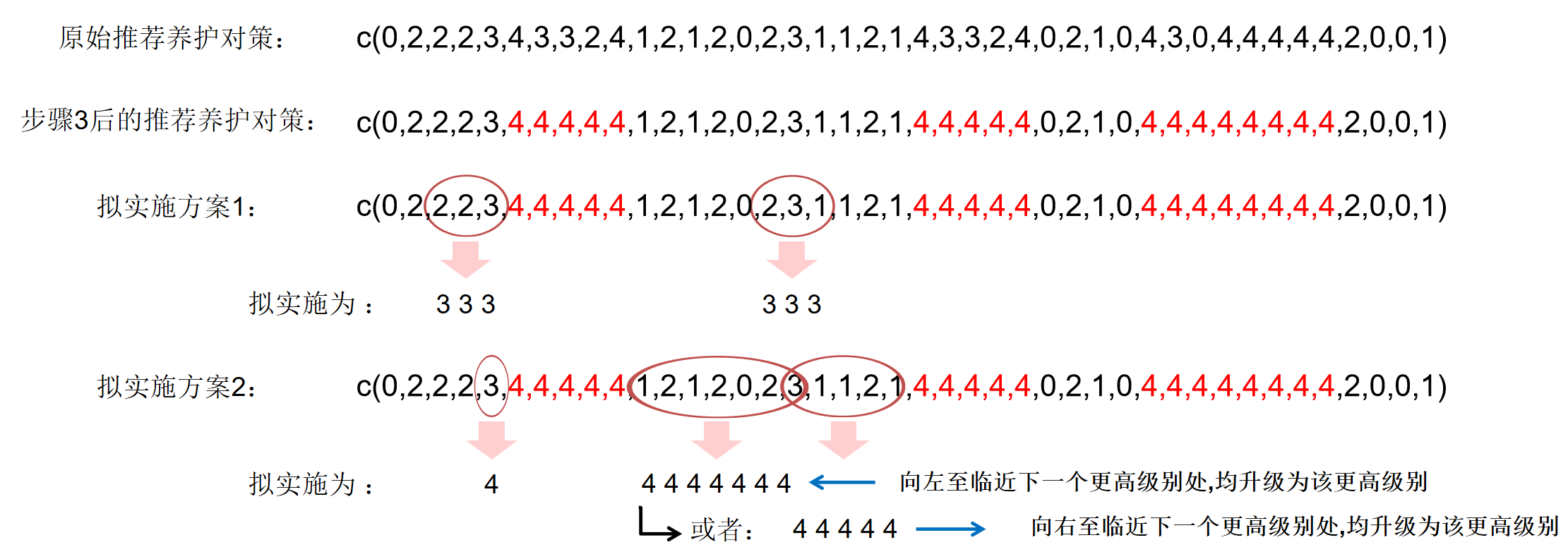
1. 步骤2：步骤1实施后，各片段自动聚合后新片段的总长可能超过最小连续实施长度。对于这些新片段，删除首尾两端原来非本级的推荐养护对策，至多删除至片段总长恰好满足最小连续实施长度。对于如下案例，经第一步后，3 4 3 3 2 4拟实施为4 4 4 4 4 4，级别为4的拟实施片段实现了自动聚合，长度超过了3为6。则对于总长超过最小连续实施长度的片段，判断其两端的原始推荐养护对策是否低于本级，如果低于，则还原，避免增加经费开支。对于左端的4，其原始推荐养护对策为3，拟还原为3，然后剩下的片段为4 4 4 4 4，长度为5，满足最小长度约束，因此可以还原，否则不可还原。默认从左端开始判断。



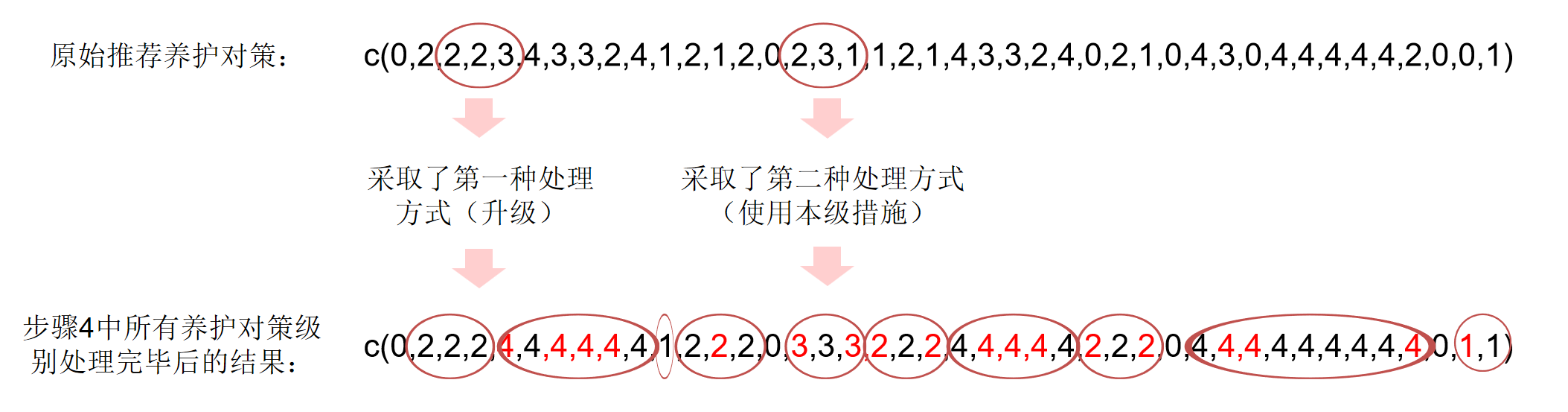
1. 步骤3：对于拟连续实施某级（记为本级措施，即当前判断到的养护对策级别）措施的片段，如果两端的原始推荐养护对策为非本级措施，则执行该片段的拟实施窗口滑动步骤，使该片段与最小连续实施长度内其它更高级别片段之间更紧凑。注意，该片段的的长度必然小于等于最小连续实施长度，因为已经经过了步骤2的处理。对于如下案例，原始推荐养护对策为0 4 3的片段，经步骤1、2，现拟实施为4 4 4。对于拟实施的窗口，可以进行滑动。可以向右滑动一步的原则是，该片段左端的原始推荐对策为非本级措施，该措施级别记为x，该片段右侧第一个相邻推荐养护对策级别也为x。向左滑动的原则同理。先判断能否向右滑动，如果不能，再判断能否向左滑动。如下案例中，原始推荐养护对策级别为0 4 3的片段右侧第一个推荐养护对策级别为0，等于该片段最左侧的推荐养护对策级别，则可以向右滑动。滑动后，由0 4 3的位置变到4 3 0的位置，4 3 0片段距离其它拟实施本级或更高级措施的片段有效地更近了，则确定滑动至4 3 0的位置。有效地更近是指，与其它拟实施本级或更高级措施的片段（本案例中，左侧的4 4 4 4 4距离本片段原片段0 4 3有4个距离，即为本片段原片段左端位置序号与左侧片段右端位置序号的差值，右侧4 4 4 4 4距离本片段原片段0 4 3有2个距离）距离小于等于最小连续实施长度3。拟实施4 4 4的位置由原来的0 4 3变为4 3 0后，与左边拟实施本级措施的片段距离变为5（从4到5无效，不计入，均大于3，至少需要一个小于等于3），与右边拟实施本级措施的片段距离变为1（从2到1有效，均小于3，至少需要一个小于等于3）。那么，对于本案例，有效距离减少了1（>0），则需要滑动。



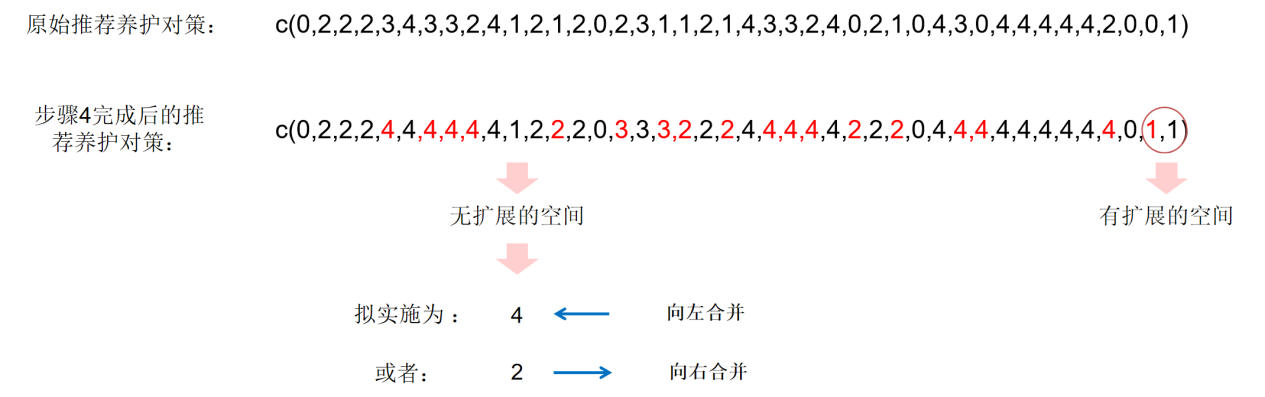
1. 步骤4，如果本级措施非原始推荐养护对策级别中的最高级别，则对于每个片段，基于增量代价最小原则，判断其完整实施本级措施，还是部分实施更高级别的措施。对于如下所示的案例，假设本级措施为3（级别为4的已经判断完毕），则如下所示的一片段2 2 3经步骤3拟实施为3 3 3（级别为3的措施最小连续实施长度为3）。在本步骤中，判断该片段确实执行为3 3 3还是将2 2 3中的两端的3至相邻更高级别片段的区域升级为该更高级别。比如，该片段2 2 3，其左侧无更高级别片段，右侧紧邻一更高级别片段4 4 4 4 4，可考虑将2 2 3中最左侧的3至右侧4 4 4 4 4片段的全部区域（本例中只有3这一小段）全部升级为4。则方案是由2 2 3变为3 3 3或2 2 4。两种方案的增量代价分别是(207.86-105.9)\*2=203.92和268.5-207.86=60.64，显然后者更低，因此拟实施方案变为2 2 4。下图所示的另一片段2 3 1，根据步骤3拟实施为3 3 3，根据步骤4其还可以实施的方案为将本片段中最右侧的3至左侧4 4 4 4 4片段中间的区域全部替换为4，或者本片段中最左侧（由于只有1个3，因此是同一个位置）的3至右侧4 4 4 4 4片段中间的区域全部替换为4，显然这两种方案的增量代价远大于将2 3 1替换为3 3 3的代价，因此仍然将2 3 1拟实施为3 3 3。



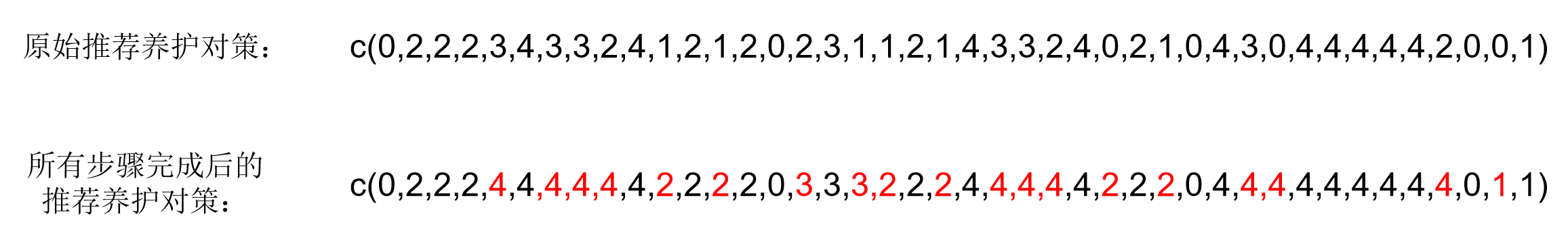
步骤4执行完毕后的效果如下所示：



1. 步骤5，在前四个步骤都执行完毕，且所有养护措施级别都判断完毕后执行。对于不满足最小连续实施长度的片段，将其合并到相邻更高级别的片段中。对于如下所示案例，第一个1的最小连续实施长度要求为2（200m），然而其只有100m（1段），不满足最小连续实施长度约束，因此需要合并到相邻更高级别的片段中。此片段左侧片段级别为4，右侧片段级别为2，因此选择更低级别的进行合并，即将1所在位置处的拟实施措施升级为2，结果如下所示。



1. 步骤1-5完整实施后结果如下所示，红色表示与原始推荐养护对策级别序列相比，有改变的地方。步骤1-5的伪代码如下所示。



#动态划分

info <- data.frame(养护对策=c("日常养护","封层","一层式铣刨加罩","二层式铣刨加罩","局部补强+二层式铣刨加罩","全面补强+翻修"),单价=c(7,42,105.9,207.86,268.5,655.82),最小连续实施长度=c(1,2,3,3,3,3),养护对策级别=c(0:5))

extract\_price\_info <- function(no,x,info){#no为需要提取信息的序号向量,x是各单元的养护对策级别序列,info为各养护对策的信息

return(info[x[no]+1,"单价"])

}

dynamic\_division <- function(x,info){#x是各单元的养护对策级别序列,info为各养护对策的信息(其中,最小连续实施长度是基本单元长度的整数倍)

y <- rep(NA,length(x))#最终推荐养护对策级别序列初始化

#第一步,从最高level开始,基于每个点,根据其min\_length要求,拟将相邻片段置为本level

highest\_level <- max(x)#最高级别

while(sum(!is.na(x))>0){

tmp <- sort(unique(x),decreasing = TRUE)#序列中唯一的养护对策级别按降序排列,先处理级别高的

level <- tmp[1]#当前考虑的养护对策级别

min\_length <- info[level+1,"最小连续实施长度"]

tmp2 <- which(x==level)#该级别最小单元所在位置

tmp3 <- foreach(i=tmp2,.combine = c) %do%

{

#从tmp2的位置起,往前往后分别考虑min\_length内的单元,且不超过x序列本身的范围

start <- max(i-min\_length+1,1)

end <- min(i+min\_length-1,length(x))

#在该范围内搜索

if(end-start<=min\_length-1){#总长度小于min\_length,直接全选

return(start:end)

}else{#范围够大,则取min\_length长度的序列,然后取全部替换为level级别后代价增加最小的

tmp4 <- lapply(seq(from=start,to=end-min\_length+1), seq,length.out=min\_length)#枚举start:end内所有长为min\_length的连续片段

tmp5 <- lapply(tmp4, extract\_price\_info,x=x,info=info)#通过序号提取相应的价格信息

tmp5 <- sapply(tmp5, sum,na.rm=TRUE)

tmp5 <- which.max(tmp5)#取第一个和最大的作为初始解

return(tmp4[[tmp5]])#范围该片段在x中的序号

}

}

#输出拟设置为本级level的序号

tmp3 <- unique(tmp3)#序号去重

tmp4 <- tmp3[!is.na(y[tmp3])];tmp3 <- setdiff(tmp3,tmp4)#已有上一级措施的不替换

#第二步,从第一步已经实现自动聚合的拟实施本level措施的片段中,筛选出连续实施长度超过min\_length的,然后将两端的非本级措施删除,以实现节省资金的目的.注意删除时如果要使本片段长度小于min\_length,则不再删除.参考示例片段为033433400

if(!identical(tmp3,numeric(0))){#如果tmp3非空,则进行处理

#存在多个拟实施本级措施的片段,则需要对每个片段分别处理.注意是!=1,因为是基于序号的差分进行的判断

start <- c(tmp3[1],tmp3[which(diff(tmp3)!=1)+1])#连续实施片段的起点序号

end <- c(tmp3[which(diff(tmp3)!=1)],tmp3[length(tmp3)])#连续实施片段的止点序号

tmp5 <- which(end-start>=min\_length)#筛选出连续实施长度超过min\_length的,注意没有-1,因为是要超过

if(!identical(tmp5,integer(0))){#如果存在连续实施长度超过min\_length的片段

tmp6 <- foreach(i=tmp5,.combine = c) %do% #对于每个片段,分别处理

{

tmp7 <- x[start[i]:end[i]]#取得该片段

tmp8 <- length(tmp7)-min\_length#允许删除的最大数量

tmp9 <- which(tmp7==level)#本级措施所在位置

tmp10 <- seq(from=1,to=length(tmp7))#tmp7中的局部序号

tmp11 <- c(which(tmp10<min(tmp9)),which(tmp10>max(tmp9)))#两端最后一个本级level点以外的位置(局部序号)

if(!identical(tmp11,integer(0))){#如果有数据点

if(length(tmp11)<=tmp8){

return((start[i]:end[i])[tmp11])#返回可以删除的全局序号

}else{

return((start[i]:end[i])[tmp11[1:tmp8]])#从前面开始删,因为初始解也是从前面开始取的

}

}else{

return(NULL)

}

}

tmp3 <- setdiff(tmp3,tmp6)#输出拟设置为本级level的序号,已根据第二步的原则删除了部分非必要实施的序号

}

#第三步,对于端头level低于本级level的片段(连续长度必然小于等于min\_length,因为经过了第二步的处理),尝试进行实施窗口的滑动,使其与min\_length范围内的其它片段更紧凑,从而实现经费节省的效果

#tmp3修改后,重新计算每个拟设置为本级level片段的起止点

start <- c(tmp3[1],tmp3[which(diff(tmp3)!=1)+1])

end <- c(tmp3[which(diff(tmp3)!=1)],tmp3[length(tmp3)])

tmp2 <- which(end-start<=min\_length-1)

while(!identical(tmp2,integer(0))){

tmp4 <- start[tmp2[1]]:end[tmp2[1]]#第一个片段的全局序号

tmp5 <- x[tmp4]#该片段对应的level序列

tmp6 <- which(tmp5==level)#本级措施所在位置(局部序号)

tmp7 <- 1:length(tmp5)#tmp5中的局部序号

tmp8 <- which(tmp7<min(tmp6))#左边端点以外的位置(局部序号)

tmp9 <- which(tmp7>max(tmp6))#右边端点以外的位置(局部序号)

slided <- FALSE#初始化,未曾滑动

while (!identical(c(tmp8),integer(0))){#左右端点中存在小于本级level的片段

tmp10 <- tmp4[tmp8[1]]#逐个全局序号判断是否需要移动

tmp8 <- tmp8[-1]#将当前序号从tmp8中移除

tmp11 <- tmp10+length(tmp4)#检查该片段右边第一个位置

#检查能否滑动到tmp11的位置

if(!any(is.na(x[tmp10:tmp11])) & x[tmp11]==x[tmp10]){#tmp10:tmp11之间不包含已经处理过的数据,且tmp10和tmp11位置处措施相同时,才可以滑动

tmp12 <- c(setdiff(tmp4,tmp10),tmp11)#向右移动一位后该片段的全局序号

tmp13 <- tmp3;tmp13[seq(from=which(tmp3==tmp10),length.out=length(tmp4))] <- tmp12;tmp13 <- sort(c(tmp13,which(!is.na(y))))#移动后拟实施本级及更高level措施的全局序号

tmp14 <- sort(c(tmp3,which(!is.na(y))))#移动前拟实施本级及更高level措施的全局序号

tmp15 <- which(diff(tmp14)>1 & diff(tmp14)<=min\_length)#移动前相邻距离小于等于min\_length且大于1的位置

tmp16 <- which(diff(tmp13)>1 & diff(tmp13)<=min\_length)#移动后相邻距离小于等于min\_length且大于1的位置

if(sum(diff(tmp13)[tmp16]) < sum(diff(tmp14)[tmp15])){#注意如果tmp15或tmp16为空sum后的值为0

tmp3 <- setdiff(tmp13,which(!is.na(y)))#需要向右移动,当且仅当存在更多相邻距离小于min\_length的片段时

slided <- TRUE#已经移动

}

}

}

if(slided==FALSE){#未曾向右移动,可以考虑向左移动

while (!identical(c(tmp9),integer(0))){

tmp10 <- tmp4[tmp9[1]]#逐个序号(全局序号)判断是否需要移动

tmp9 <- tmp9[-1]#将当前序号从tmp9中移除

tmp11 <- tmp10-length(tmp4)#检查该片段左边第一个位置

#检查能否移动到tmp11的位置

if(!any(is.na(x[tmp11:tmp10])) & x[tmp11]==x[tmp10]){#tmp11:tmp10之间不包含已经处理过的数据,且tmp11和tmp10位置处措施相同时,才可以移动

tmp12 <- c(tmp11,setdiff(tmp4,tmp10))#向左移动一位后该片段的全局序号

tmp13 <- tmp3;tmp13[seq(to=which(tmp3==tmp10),length.out=length(tmp4))] <- tmp12;tmp13 <- sort(c(tmp13,which(!is.na(y))))#移动后拟实施本级及更高level措施的全局序号

tmp14 <- sort(c(tmp3,which(!is.na(y))))#移动前拟实施本级及更高level措施的全局序号

tmp15 <- which(diff(tmp14)>1 &diff(tmp14)<=min\_length)#移动前相邻距离小于等于min\_length且大于1的位置

tmp16 <- which(diff(tmp13)>1 & diff(tmp13)<=min\_length)#移动后相邻距离小于等于min\_length且大于1的位置

if(sum(diff(tmp13)[tmp16]) < sum(diff(tmp14)[tmp15])){#注意如果tmp15或tmp16为空sum后的值为0

tmp3 <- setdiff(tmp13,which(!is.na(y)))#需要向左移动,当且仅当存在更多相邻距离小于min\_length的片段时

}

}

}

}

tmp2 <- tmp2[-1]#不再考虑该片段

}

}

#第四步,如果本级level非highest\_level,则判断tmp3中每个片段,是部分实施更高level，还是整体实施本级level

if(level!=highest\_level){

while(!(identical(tmp3,integer(0))|identical(tmp3,numeric(0)))){#tmp3逐个片段进行处理

start <- tmp3[c(1,which(diff(tmp3)!=1)+1)][1]#第一个片段的起点全局序号

end <- tmp3[c(which(diff(tmp3)!=1),length(tmp3))][1]#第一个片段的止点全局序号

tmp2 <- x[start:end]#取出第一个片段的等级序列

if(max(tmp2)<level){#跨片段设置导致本片段中本级level缺失

tmp3 <- setdiff(tmp3,c(start:end))#移除已经考虑过的全局序号

next

}else{

if(tmp2[1]==level & tmp2[length(tmp2)]==level){#如果该片段首尾的level均为本级level,则整体实施本级level

tmp3 <- setdiff(tmp3,c(start:end))#移除已经考虑过的全局序号

y[start:end] <- level#设置本级level序列

x[start:end] <- NA#已经考虑过的位置置为NA

next

}else{

delta\_cost\_default <- sum(info[level+1,"单价"]-info[tmp2+1,"单价"])#整体实施本级level的增量代价

location\_this <- (start:end)[which(tmp2==level)]#本级level的全局序号

location\_higher <- which(!is.na(y))#比本级level更高的位置

if(max(location\_this)<min(location\_higher)){#只能向右升级

no\_right <- min(location\_this):(min(location\_higher)-1)#拟升级的全局序号

upgraded\_level\_right <- y[min(location\_higher)]#拟升级为的级别

delta\_cost\_right <- info[upgraded\_level\_right+1,"单价"]\*length(no\_right)-sum(info[x[no\_right]+1,"单价"])#采用升级方法的增量代价

delta\_cost\_left <- Inf

}else if(min(location\_this)>max(location\_higher)){#只能向左升级

no\_left <- (max(location\_higher)+1):min(location\_this)#拟升级的全局序号

upgraded\_level\_left <- y[max(location\_higher)]#拟升级为的级别

delta\_cost\_left <- info[upgraded\_level\_left+1,"单价"]\*length(no\_left)-sum(info[x[no\_left]+1,"单价"])#采用升级方法的增量代价

delta\_cost\_right <- Inf

}else{

no\_left <- (location\_higher[which.max(location\_higher-min(location\_this)>0)-1]+1):min(location\_this)#往左侧升级拟采用的全局序号

no\_right <- max(location\_this):(location\_higher[which.max(location\_higher-min(location\_this)>0)]-1)#往右侧升级拟采用的全局序号

upgraded\_level\_left <- y[location\_higher[which.max(location\_higher-min(location\_this)>0)-1]]#往左侧升级拟升级为的级别

upgraded\_level\_right <- y[location\_higher[which.max(location\_higher-min(location\_this)>0)]]#往右侧升级拟升级为的级别

delta\_cost\_left <- info[upgraded\_level\_left+1,"单价"]\*length(no\_left)-sum(info[x[no\_left]+1,"单价"])#往左侧升级的增量代价

delta\_cost\_right <- info[upgraded\_level\_right+1,"单价"]\*length(no\_right)-sum(info[x[no\_right]+1,"单价"])#往右侧升级的增量代价

}

if(delta\_cost\_default <= min(delta\_cost\_left,delta\_cost\_right)){

tmp3 <- setdiff(tmp3,c(start:end))#移除已经考虑过的全局序号

y[start:end] <- level#设置本级level序列

x[start:end] <- NA#已经考虑过的位置置为NA

next

}else if(delta\_cost\_left<=delta\_cost\_right){#往左侧升级

y[no\_left] <- upgraded\_level\_left#设置左侧升级序列

x[no\_left] <- NA#已经考虑过的位置置为NA

tmp3 <- setdiff(tmp3,c(start:end,no\_left))#移除已经考虑过的全局序号

}else{#往右侧升级

y[no\_right] <- upgraded\_level\_right#设置右侧升级序列

x[no\_right] <- NA#已经考虑过的位置置为NA

tmp3 <- setdiff(tmp3,c(start:end,no\_right))#移除已经考虑过的全局序号

}

}

}

}

}else{

y[tmp3] <- level#设置本级level序列

x[tmp3] <- NA#已经考虑过的位置置为NA

}

}

# 第五步,对于不满足连续实施最小长度的片段,考虑两种处理方式,一种是合并到相邻更高level的片段中，一种是相邻更低level的(部分)片段替换为本level

repeat{

tmp <- c(1,which(diff(y)!=0)+1)#所有片段的起点序号

segment\_info <- data.frame(start=tmp,end=c(which(diff(y)!=0),length(y)),level=y[tmp])#每个片段基本信息

segment\_info$length <- segment\_info$end-segment\_info$start+1;segment\_info$min\_length <- info[segment\_info$level+1,"最小连续实施长度"]#补充每个片段的基本信息

tmp2 <- which(segment\_info$length<segment\_info$min\_length)#需要处理的片段在segment\_info中的局部序号

if(identical(tmp2,integer(0))){#没有需要处理的片段了

break

}

tmp2 <- tmp2[1]#逐个片段进行处理

if(nrow(segment\_info)==1){#只有一个片段,则无需处理

break

}else if(tmp2==1){#片段开头,只能向右处理

right\_level <- segment\_info[tmp2+1,"level"]

if(right\_level>segment\_info[tmp2,"level"]){

y[segment\_info[tmp2,"start"]:segment\_info[tmp2,"end"]] <- right\_level#右侧level更大,则本段的level置为右侧level

}else{

additional\_length <- segment\_info[tmp2,"min\_length"]-segment\_info[tmp2,"length"]#需要扩大的长度

right\_length <- min(additional\_length,segment\_info[tmp2+1,"length"])#最大扩展至下一个完整的相邻片段,如果后续还需要扩大可以通过反复迭代实现

y[segment\_info[tmp2,"start"]:(segment\_info[tmp2,"end"]+right\_length)] <- segment\_info[tmp2,"level"]#将相邻更低level的(部分)片段替换为本level

}

}else if(tmp2==nrow(segment\_info)){#片段末尾,只能向左处理

left\_level <- segment\_info[tmp2-1,"level"]

if(left\_level>segment\_info[tmp2,"level"]){

y[segment\_info[tmp2,"start"]:segment\_info[tmp2,"end"]] <- left\_level#左侧level更大,则本段的level置为右侧level

}else{

additional\_length <- segment\_info[tmp2,"min\_length"]-segment\_info[tmp2,"length"]#需要扩大的长度

left\_length <- min(additional\_length,segment\_info[tmp2-1,"length"])#最大扩展至下一个完整的相邻片段,如果后续还需要扩大可以通过反复迭代实现

y[(segment\_info[tmp2,"start"]-left\_length):(segment\_info[tmp2,"end"])] <- segment\_info[tmp2,"level"]#将相邻更低level的(部分)片段替换为本level

}

}else{#中间片段,对可以向两边处理

left\_level <- segment\_info[tmp2-1,"level"]

right\_level <- segment\_info[tmp2+1,"level"]

if(min(left\_level,right\_level)>segment\_info[tmp2,"level"]){#如果两边的level都更高,则本段的level设置为其中更低的

y[segment\_info[tmp2,"start"]:segment\_info[tmp2,"end"]] <- min(left\_level,right\_level)

}else{

additional\_length <- segment\_info[tmp2,"min\_length"]-segment\_info[tmp2,"length"]#需要扩大的长度

if(left\_level>segment\_info[tmp2,"level"]){#只能向右扩展

right\_length <- min(additional\_length,segment\_info[tmp2+1,"length"])#最大扩展至下一个完整的相邻片段,如果后续还需要扩大可以通过反复迭代实现

y[segment\_info[tmp2,"start"]:(segment\_info[tmp2,"end"]+right\_length)] <- segment\_info[tmp2,"level"]#将相邻更低level的(部分)片段替换为本level

}else if(right\_level>segment\_info[tmp2,"level"]){

left\_length <- min(additional\_length,segment\_info[tmp2-1,"length"])#最大扩展至下一个完整的相邻片段,如果后续还需要扩大可以通过反复迭代实现

y[(segment\_info[tmp2,"start"]-left\_length):(segment\_info[tmp2,"end"])] <- segment\_info[tmp2,"level"]#将相邻更低level的(部分)片段替换为本level

}else{

#可以向两边扩展的长度

left\_length <- min(additional\_length,segment\_info[tmp2-1,"length"])

right\_length <- min(additional\_length,segment\_info[tmp2+1,"length"])

if(segment\_info[tmp2,"length"]+left\_length+right\_length<segment\_info[tmp2,"min\_length"]){#向两边最大程度扩展后仍不满足最小连续实施长度要求,则整段置为本级level,待后续迭代处理

y[(segment\_info[tmp2,"start"]-left\_length):(segment\_info[tmp2,"end"]+right\_length)] <- segment\_info[tmp2,"level"]#将本片段扩展后的片段替换为本级level

}else{

tmp3 <- lapply(seq(from=segment\_info[tmp2,"start"]-left\_length,to=segment\_info[tmp2,"end"]+right\_length-segment\_info[tmp2,"min\_length"]+1), seq,length.out=segment\_info[tmp2,"min\_length"])#枚举可选全局序号范围中任意连续min\_length长度的序号

tmp4 <- lapply(tmp3, extract\_price\_info,x=y,info=info)#通过序号提取相应的价格信息

tmp5 <- sapply(tmp4, sum)

tmp6 <- which.max(tmp5)#第一个最大值所在位置

y[tmp3[[tmp6]]] <- segment\_info[tmp2,"level"]#将本片段扩展后的片段替换为本级level

}

}

}

}

}

return(y)

}

tmp <- c(0,2,2,2,3,4,3,3,2,4,1,2,1,2,0,2,3,1,1,2,1,4,3,3,2,4,0,2,1,0,4,3,0,4,4,4,4,4,2,0,0,1)

print(dynamic\_division(tmp,info = info))