CrRNA 筛选

实验目标：筛选出对 L1PA1-4 覆盖率高，且存在一定亚家族特异性的 crRNA。

主要思路：（1）根据 TE 的基因组位置的注释信息，使用 bedtools 提取 L1PA1-4 所有基因组位置的 fasta 序列（负链转为正链），所有序列保存为一个文件。（2）以 30bp 为单位，对每条 fasta 序列进行滑窗截取，截取完成之后，若某段序列在基因组的多个位置出现，则去重，仅保留一个序列；对所有的 fasta 序列重复上述步骤；（3）将得到的 30bp 的序列进行计数，并降序排列；取前 3000 为候选的 crRNA。（4）以所有的 TE 进行 blast 建库，将所有候选的 crRNA 与上述索引进行比对，获取 crRNA 所靶向的序列，并计算靶向的序列占该亚家族所有序列的比例，汇总成表格。（5）使用 excel 将文件，按照一定的组合条件（如：L1HS 6k >90% && L1PA5 6k＜40%）进行筛选。（6）若候选的序列间存在一定比例的 overlap，则将其进行标注。

一、 crRNA 遍历计数

1. 获取候选 crRNA

代码 1：使用 bedtools 提取 L1PA1-4 所有基因组位置的 fasta 序列（负链转为正链）。

细节说明：这里考虑到运算时间，只关注 L1PA1-4 ＞6kb 的序列。并且“-s”参数考虑了链的信息，将负链转换为正链。

```
cd /home/xxzhang/workplace/project/CRISPRa/sgRNA_30bp_window/fasta/6kb/
bedtools getfasta -fi "/home/xxzhang/data/Genome_reference/Genome/homo_sapiens/hg38/GRCh38.primary_assembly.genome.fa" -bed L1PA1_4_6k.bed -s -name -fo L1PA1_4_6k.fasta
## awk 'NR % 2 == 1 {print substr($0, 1, 50); next} {print}' exon.fasta >exon_cut50.fasta  #这一步骤可选，为防止名字过长报错。
```

代码 2：sgRNA_30window_v5.py：包含上述步骤（1），（2），（3）。

细节说明：因为最终设计出来的 crRNA 是要跟 L1 转录出来的 RNA（无论正负链，编码出来的 RNA 应该都是 5’-3’方向）互补配对的。因此我们最后在这个

代码中得到的序列是互补序列。

```python
import argparse
from Bio import SeqIO
from Bio.Seq import Seq
from collections import Counter
from concurrent.futures import ThreadPoolExecutor, as_completed

def slide_window(sequence, window_size):
    """滑窗生成器，按指定窗口大小切割序列"""
    for i in range(len(sequence) - window_size + 1):
        yield str(sequence[i:i + window_size])

def calculate_gc_content(sequence):
    """计算 GC 含量百分比（保留两位小数）"""
    gc = sequence.upper().count("G") + sequence.upper().count("C")
    return round(100 * gc / len(sequence), 2) if len(sequence) > 0 else 0.0

def get_reverse_complement(seq):
    """生成反向互补序列"""
    return str(Seq(seq).reverse_complement())

def process_single_sequence(sequence, window_size):
    """处理单条序列：去重后统计反向互补序列"""
    unique_subseqs = set()  # 使用集合自动去重
    for subseq in slide_window(sequence, window_size):
        rc_seq = get_reverse_complement(subseq)
        unique_subseqs.add(rc_seq)
    return Counter(unique_subseqs)  # 每个唯一序列计数为 1

def parallel_process_fasta(input_file, window_size, max_workers):
    """并行处理 FASTA 文件并汇总结果"""
    total_counter = Counter()
    with ThreadPoolExecutor(max_workers=max_workers) as executor:
        futures = []
        for record in SeqIO.parse(input_file, "fasta"):
            seq = str(record.seq)
            if len(seq) < window_size:
                continue  # 跳过短于窗口的序列
            futures.append(executor.submit(process_single_sequence, seq, window_size))

        for future in as_completed(futures):
            total_counter.update(future.result())
    return total_counter

def write_results(output_file, counter):
    """将结果写入文件，按频率降序排列"""
    with open(output_file, 'w') as f:
        f.write("Reverse_Complement_Sequence\tCount\tGC_Content(%)\n")
        for seq, count in counter.most_common():
            gc = calculate_gc_content(seq)
            f.write(f"{seq}\t{count}\t{gc:.2f}\n")

def main():
    parser = argparse.ArgumentParser(description="统计 FASTA 中 30bp 窗口反向互补序列的出现频率（同序列去重）")
    parser.add_argument("-i", "--input", required=True, help="输入 FASTA 文件路径")
    parser.add_argument("-o", "--output", required=True, help="输出文件路径")
    parser.add_argument("-w", "--window", type=int, default=30, help="滑窗大小（默认30）")
    parser.add_argument("-t", "--threads", type=int, default=4, help="并行线程数（默认4）")
    args = parser.parse_args()

    result_counter = parallel_process_fasta(args.input, args.window, args.threads)
    write_results(args.output, result_counter)
    print(f"结果已保存至 {args.output}")

if __name__ == "__main__":
    main()

```

运行命令行：

```
python sgRNA_30window_v5.py -i L1PA1_4_6k.fasta  -o  L1PA1_4.txt -w 30 -t 4
```

二、crRNA 检索并筛选

1. 代码 3：blast 检索，gRNA_screenTE_mismatch_strand_minus.pl

细节说明：因为上述提取过程中，生成的逆反序列。因此在 blast 比对的时候，需要逆反序列比对。

```perl
#!perl
use Getopt::Long;
```

```perl
GetOptions(
           "sgRNA=s" =>\$sgRNA,
                          "length=s" =>\$length,
                          "mismatch=s" =>\$mismatch,
                          "fastaindex=s" =>\$fastaindex,
                          "prefix=s" =>\$prefix,
           "h|help" =>\$help,

);
if($help)
{
print
"
usage:
-sgRNA     : A text file containing sgRNA sequences, with each line representing a candidate sequence; the default file name is gRNA.txt;
-length     : Length of the sgRNA,default:23;
-mismatch     : Mismatch number when searching,default:0;
-fastaindex      : Fasta file name used to build the BLAST index;default:hg38_bedtools_L1.chr1-Y.fasta;
-h         : usage of this scripts
"
}
open (MARK, "< ".$sgRNA) or die "can not open it!";
$searchLen = $length - $mismatch;
print $searchLen;
$Result = $prefix."_result_".$mismatch;
$Count = $prefix."_count_".$mismatch;
$outfile = $prefix."_output".$mismatch.".txt";
while ($line = <MARK>){
                print($line);
                chomp($line);
                print($line);
                $line = uc($line);
                system_call("mkdir -p ".$Result);
                system_call("mkdir -p ".$Count);
                system_call("mkdir -p tmp");
                system_call("echo '>".$line."\' > ".$line.".fa");
                system_call("echo ".$line." >>".$line.".fa");
                system_call("blastn -db ".$fastaindex." -query ".$line.".fa -word_size 4 -dust no -outfmt 6 -task blastn-short -max_target_seqs 100000 -strand minus | awk '((\$3 == 100.000 && \$4 >= ".$searchLen.")||(\$4 == ".$length." && \$5 <= ".$mismatch."))' >\./".$Result."/".$line.".blast.result  ");#default:word_size 4
                system_call("cat \./".$Result."/".$line.".blast.result |awk '{print \$2}' |awk -v FS=\":\" -v OFS=\":\" '{print \$1,\$2,\$3}' |sort |uniq -c |awk -v OFS=\"\t\" '{print \$2,\$1,\"".$line."\"}' >\./".$Count."/".$line.".count".$mismatch.".result");
}
system_call("cat \./".$Count."/* >".$outfile);
system_call("mv *.fa ./tmp/");
close(MARK);
sub system_call
{
  my $command=$_[0];
  print "\n\n".$command."\n\n";
  system($command);
}


```

运行命令行：

```
#PBS -N sgRNA_screening
#PBS -q gpu
#PBS -l nodes=1:ppn=4
#PBS -l mem=100gb
#PBS -M 2456392738@qq.com
#PBS -m abe
cd /home/xxzhang/workplace/project/CRISPRa/gRNA/allTE/gRNA1001/
perl "/home/xxzhang/workplace/project/CRISPRa/gRNA/allTE/gRNA_screenTE_mismatch_strand_minus.pl" -sgRNA L1PA1_4_sgRNA.txt -length 30 -mismatch 3 -fastaindex hg38_bedtools_TE.chr1-Y.processed.final.50.fasta -prefix L1PA1_4
```

2. 结合背景信息，计算 crRNA 靶向比例

代码 4：processResult.r 文件，计算 crRNA 靶向的比例，并画图。

```
setwd("I://毕业论文//实验方法//crRNA 筛选")
library(dplyr)
library(tidyr)
wid=8 #设置输出图片宽
high=8 #设置输出图片长
mismatch=3 #修改此处
inputfile<-paste("L1PA1_4_top1000_output3.txt")
```

```r
metadata<-read.table("TE.total.num.txt") #修改此处
data<-read.table(inputfile)
mergeDat<-merge(data,metadata,by="V1")
library(dplyr)
library(tidyr)
data_ext<-mergeDat %>% separate(V1, c("family","subfamily","length"),sep = "[:]")
head(data_ext)
colnames(data_ext)[4]<-"count"
colnames(data_ext)[5]<-"gRNA"
colnames(data_ext)[6]<-"label"
colnames(data_ext)[7]<-"total"
data_ext$count<-as.numeric(data_ext$count)
data_ext$total<-as.numeric(data_ext$total)
data_ext$per<-data_ext$count/data_ext$total
data_ext$length<-factor(data_ext$length,levels=rev(c("<=2k","2k-4k","4k-6k",">6k")))
mergeDat3<-data_ext
#####save the percentage result
mergeDat4<-mergeDat3[,-c(4,7)]
library(stringr)
mergeDat4$G_per<-str_count(mergeDat4$gRNA, "G")
mergeDat4$C_per<-str_count(mergeDat4$gRNA, "C")
mergeDat4$GC_per<-(mergeDat4$G_per+mergeDat4$C_per)/30
head(mergeDat4)
library(dplyr)
mergeDat4 <- mergeDat4 %>%
  mutate(TTTT = ifelse(grepl("TTTT", gRNA), TRUE, FALSE))
mergeDat4$class<-
paste(paste(mergeDat4$family,mergeDat4$subfamily,sep=":"),mergeDat4$length,sep=":")
mergeDat5<-mergeDat4[,c(11,4,6,9,10)]
data_save <- mergeDat5 %>%
  pivot_wider(names_from = class, values_from = per)
data_save[is.na(data_save)] <- 0
#保留列名中有 L1 的
data_sf<-data_save[,c(1,2,3,grep("L1",colnames(data_save)))]
head(data_sf)
order<-grep(">6k",colnames(data_sf))
order2<-append(1:3,order)
other<-setdiff(4:dim(data_sf)[2],order2)
final<-append(order2,other)
data_sf2<-data_sf[,final]
head(data_sf2)
write.csv(data_sf2,"finalResult_top3000.csv",row.names=F)

```

```

#####draw plot
#将想要作图的序列提取出来
seq<-"ATTATACTTTAAGTTTTAGGGTACATGTGC"
plotDat<-data_ext[data_ext$gRNA%in%c(seq),]
background_colors <- c("#f0f0f0", "#ffffff")
unique_categories <- unique(plotDat$subfamily)
category_positions <- as.numeric(factor(unique_categories))
background_data <- data.frame(
  xmin = category_positions - 0.5,
  xmax = category_positions + 0.5,
  ymin = -Inf,
  ymax = Inf,
  fill = rep(background_colors, length.out = length(unique_categories))
)
library(ggplot2)
options(repr.plot.width =4, repr.plot.height =3)
title<-paste("mismatch:<=",mismatch,sep="")
p2<-ggplot(data=plotDat) +
  geom_rect(data = background_data, aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax, fill = fill), alpha = 0.2,
            show.legend = FALSE ) +
  geom_bar(aes(fill=length, y=per, x=subfamily),position='dodge', stat='identity')+
  scale_fill_manual(name = "length",values = c("#98d09d","#fbf398","#f7a895","#9b8191","grey","#ffffff"))+
  theme_classic()+
  #facet_grid(. ~ gRNA) +
  scale_y_continuous(limits = c(0, 1))+
  coord_flip()+
  labs(title=title,y="Percentage(%)")+
  theme(plot.title=element_text(face="bold", #字体
                               color="steelblue", #颜色
                               size=20,  #大小
                               hjust=0.5, #位置
                               vjust=0.5,
                               angle=360))
p2
outputfile<-paste("crRNA_",seq,".pdf",sep="")
pdf(outputfile,width = 4,height = 3)
p2
dev.off()

```

对于后续这个课题涉及的每个步骤都要按照这种方式作详细的记录和总结。

三、合并相似 crRNA

代码 5：crRNA_cluster.py，将序列上相近的 crRNA，分为一类。

```python
import pandas as pd
import numpy as np
from Bio import pairwise2
from itertools import combinations
from tqdm import tqdm
import argparse
from concurrent.futures import ProcessPoolExecutor
import multiprocessing

def read_sequences_from_file(filename):
    """从文件读取并标准化序列"""
    with open(filename, 'r') as f:
        return list({s.strip().upper() for s in f if s.strip()})

def calc_identity(args):
    """多进程任务函数：计算序列相似度"""
    i, j, seq1, seq2, threshold = args
    len1, len2 = len(seq1), len(seq2)
    max_len = max(len1, len2)

    # 快速长度过滤
    if min(len1, len2) < 0.75 * max_len:
        return (i, j, False)

    # 精确比对
    align = pairwise2.align.globalxx(seq1, seq2, one_alignment_only=True)[0]
    matches = align.score
    similarity = matches / max_len * 100
    return (i, j, similarity >= threshold)

def cluster_sequences(sequences, threshold=75, workers=None):
    """多进程聚类主函数"""
    n = len(sequences)
    manager = multiprocessing.Manager()
    adjacency = manager.dict({(i,j): False for i, j in combinations(range(n), 2)})

    # 生成任务列表
    tasks = [(i, j, sequences[i], sequences[j], threshold)
             for i, j in combinations(range(n), 2)]

    # 多进程处理
    with ProcessPoolExecutor(max_workers=workers) as executor:
        results = list(tqdm(executor.map(calc_identity, tasks),
                        total=len(tasks),
                        desc="Processing pairs"))

    # 构建邻接矩阵
    adj_matrix = np.eye(n, dtype=bool)
    for i, j, match in results:
        if match:
            adj_matrix[i][j] = True
            adj_matrix[j][i] = True

    # 查找连通分量
    visited = set()
    groups = []
    for i in range(n):
        if i not in visited:
            component = set(np.where(adj_matrix[i])[0])
            visited.update(component)
            groups.append(sorted(component, key=lambda x: -len(sequences[x])))

    return [[sequences[i] for i in group] for group in groups]

def create_report(groups):
    """生成报告数据（同前）"""
    report = []
    for group_id, group in enumerate(groups, 1):
        group_size = len(group)
        for seq in group:
            report.append({
                "GroupID": f"Group_{group_id:03d}",
                "Sequence": seq,
                "Length": len(seq),
                "GC%": round((seq.count("G") + seq.count("C"))/len(seq)*100, 2),
                "GroupSize": group_size,
                "IsSingleton": group_size == 1
            })
    return pd.DataFrame(report)

def main():
```

```python
    parser = argparse.ArgumentParser(description="crRNA 序列聚类工具")
    parser.add_argument("-i", "--input", required=True, help="输入文件路径（crRNA_list.txt）")
    parser.add_argument("-o", "--output", default="crRNA_groups.xlsx", help="输出 Excel 文件路径")
    parser.add_argument("-t", "--threshold", type=float, default=75,
                        help="相似度阈值（百分比）")
    parser.add_argument("-w", "--workers", type=int,
                        default=multiprocessing.cpu_count()-1,
                        help="并行进程数（默认 CPU 核心数-1）")
    args = parser.parse_args()

    # 读取和处理数据
    print(f"正在读取序列文件: {args.input}")
    sequences = read_sequences_from_file(args.input)
    print(f"发现 {len(sequences)} 条唯一序列")

    print(f"开始聚类（阈值={args.threshold}%，进程数={args.workers}）")
    groups = cluster_sequences(sequences, args.threshold, args.workers)

    # 生成报告
    print("生成分析报告...")
    df = create_report(groups)

    # 输出 Excel
    with pd.ExcelWriter(args.output) as writer:
        df.to_excel(writer, sheet_name="Full_Report", index=False)
        summary = df.groupby("GroupID").agg(
            Sequences=("Sequence", lambda x: "\n".join(x)),
            Count=("GroupID", "count"),
            Avg_Length=("Length", "mean"),
            Avg_GC=("GC%", "mean")
        ).reset_index()
        summary.to_excel(writer, sheet_name="Group_Summary", index=False)
        df[df["IsSingleton"]].to_excel(writer, sheet_name="Singletons", index=False)

    print(f"处理完成！结果已保存至 {args.output}")

if __name__ == "__main__":
main()

```

运行命令行：

```
python crRNA_cluster.py -i L1PA1_4_sgRNA.top100.txt -o results.xlsx -t 75 -w 8
```