**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。

1. cd /home/xxzhang/workplace/project/CRISPRa/sgRNA\_30bp\_window/fasta/6kb/
2. 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
3. ## 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**。

1. import argparse

2. from Bio import SeqIO

3. from Bio.Seq import Seq

4. from collections import Counter

5. from concurrent.futures import ThreadPoolExecutor, as\_completed

6.

7. def slide\_window(sequence, window\_size):

8. """滑窗生成器，按指定窗口大小切割序列"""

9. for i in range(len(sequence) - window\_size + 1):

10. yield str(sequence[i:i + window\_size])

11.

12. def calculate\_gc\_content(sequence):

13. """计算GC含量百分比（保留两位小数）"""

14. gc = sequence.upper().count("G") + sequence.upper().count("C")

15. return round(100 \* gc / len(sequence), 2) if len(sequence) > 0 else 0.0

16.

17. def get\_reverse\_complement(seq):

18. """生成反向互补序列"""

19. return str(Seq(seq).reverse\_complement())

20.

21. def process\_single\_sequence(sequence, window\_size):

22. """处理单条序列：去重后统计反向互补序列"""

23. unique\_subseqs = set() # 使用集合自动去重

24. for subseq in slide\_window(sequence, window\_size):

25. rc\_seq = get\_reverse\_complement(subseq)

26. unique\_subseqs.add(rc\_seq)

27. return Counter(unique\_subseqs) # 每个唯一序列计数为1

28.

29. def parallel\_process\_fasta(input\_file, window\_size, max\_workers):

30. """并行处理FASTA文件并汇总结果"""

31. total\_counter = Counter()

32. with ThreadPoolExecutor(max\_workers=max\_workers) as executor:

33. futures = []

34. for record in SeqIO.parse(input\_file, "fasta"):

35. seq = str(record.seq)

36. if len(seq) < window\_size:

37. continue # 跳过短于窗口的序列

38. futures.append(executor.submit(process\_single\_sequence, seq, window\_size))

39.

40. for future in as\_completed(futures):

41. total\_counter.update(future.result())

42. return total\_counter

43.

44. def write\_results(output\_file, counter):

45. """将结果写入文件，按频率降序排列"""

46. with open(output\_file, 'w') as f:

47. f.write("Reverse\_Complement\_Sequence\tCount\tGC\_Content(%)\n")

48. for seq, count in counter.most\_common():

49. gc = calculate\_gc\_content(seq)

50. f.write(f"{seq}\t{count}\t{gc:.2f}\n")

51.

52. def main():

53. 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）")

54. parser.add\_argument("-i", "--input", required=True, help="输入FASTA文件路径")

55. parser.add\_argument("-o", "--output", required=True, help="输出文件路径")

56. parser.add\_argument("-w", "--window", type=int, default=30, help="滑窗大小（默认30）")

57. parser.add\_argument("-t", "--threads", type=int, default=4, help="并行线程数（默认4）")

58. args = parser.parse\_args()

59.

60. result\_counter = parallel\_process\_fasta(args.input, args.window, args.threads)

61. write\_results(args.output, result\_counter)

62. print(f"结果已保存至 {args.output}")

63.

64. if \_\_name\_\_ == "\_\_main\_\_":

65. main()

66.

**运行命令行：**

1. 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。

1. #!perl

2. use Getopt::Long;

3. GetOptions(

4. "sgRNA=s" =>\$sgRNA,

5. "length=s" =>\$length,

6. "mismatch=s" =>\$mismatch,

7. "fastaindex=s" =>\$fastaindex,

8. "prefix=s" =>\$prefix,

9. "h|help" =>\$help,

10.

11. );

12. if($help)

13. {

14. print

15. "

16. usage:

17. -sgRNA : A text file containing sgRNA sequences, with each line representing a candidate sequence; the default file name is gRNA.txt;

18. -length : Length of the sgRNA,default:23;

19. -mismatch : Mismatch number when searching,default:0;

20. -fastaindex : Fasta file name used to build the BLAST index;default:hg38\_bedtools\_L1.chr1-Y.fasta;

21. -h : usage of this scripts

22. "

23. }

24. open (MARK, "< ".$sgRNA) or die "can not open it!";

25. $searchLen = $length - $mismatch;

26. print $searchLen;

27. $Result = $prefix."\_result\_".$mismatch;

28. $Count = $prefix."\_count\_".$mismatch;

29. $outfile = $prefix."\_output".$mismatch.".txt";

30. while ($line = <MARK>){

31. print($line);

32. chomp($line);

33. print($line);

34. $line = uc($line);

35. system\_call("mkdir -p ".$Result);

36. system\_call("mkdir -p ".$Count);

37. system\_call("mkdir -p tmp");

38. system\_call("echo '>".$line."\' > ".$line.".fa");

39. system\_call("echo ".$line." >>".$line.".fa");

40. 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

41. 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");

42. }

43. system\_call("cat \./".$Count."/\* >".$outfile);

44. system\_call("mv \*.fa ./tmp/");

45. close(MARK);

46. sub system\_call

47. {

48. my $command=$\_[0];

49. print "\n\n".$command."\n\n";

50. system($command);

51. }

52.

53.

**运行命令行：**

1. #PBS -N sgRNA\_screening

2. #PBS -q gpu

3. #PBS -l nodes=1:ppn=4

4. #PBS -l mem=100gb

5. #PBS -M 2456392738@qq.com

6. #PBS -m abe

7. cd /home/xxzhang/workplace/project/CRISPRa/gRNA/allTE/gRNA1001/

8. 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

1. **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。

1. setwd("I://毕业论文//实验方法//crRNA筛选")

2. library(dplyr)

3. library(tidyr)

4. wid=8 #设置输出图片宽

5. high=8 #设置输出图片长

6. mismatch=3 #修改此处

7. inputfile<-paste("L1PA1\_4\_top1000\_output3.txt")

8. metadata<-read.table("TE.total.num.txt") #修改此处

9. data<-read.table(inputfile)

10. mergeDat<-merge(data,metadata,by="V1")

11. library(dplyr)

12. library(tidyr)

13. data\_ext<-mergeDat %>% separate(V1, c("family","subfamily","length"),sep = "[:]")

14. head(data\_ext)

15. colnames(data\_ext)[4]<-"count"

16. colnames(data\_ext)[5]<-"gRNA"

17. colnames(data\_ext)[6]<-"label"

18. colnames(data\_ext)[7]<-"total"

19. data\_ext$count<-as.numeric(data\_ext$count)

20. data\_ext$total<-as.numeric(data\_ext$total)

21. data\_ext$per<-data\_ext$count/data\_ext$total

22. data\_ext$length<-factor(data\_ext$length,levels=rev(c("<=2k","2k-4k","4k-6k",">6k")))

23. mergeDat3<-data\_ext

24. #####save the percentage result

25. mergeDat4<-mergeDat3[,-c(4,7)]

26. library(stringr)

27. mergeDat4$G\_per<-str\_count(mergeDat4$gRNA, "G")

28. mergeDat4$C\_per<-str\_count(mergeDat4$gRNA, "C")

29. mergeDat4$GC\_per<-(mergeDat4$G\_per+mergeDat4$C\_per)/30

30. head(mergeDat4)

31. library(dplyr)

32. mergeDat4 <- mergeDat4 %>%

33. mutate(TTTT = ifelse(grepl("TTTT", gRNA), TRUE, FALSE))

34. mergeDat4$class<-paste(paste(mergeDat4$family,mergeDat4$subfamily,sep=":"),mergeDat4$length,sep=":")

35. mergeDat5<-mergeDat4[,c(11,4,6,9,10)]

36. data\_save <- mergeDat5 %>%

37. pivot\_wider(names\_from = class, values\_from = per)

38. data\_save[is.na(data\_save)] <- 0

39. #保留列名中有L1的

40. data\_sf<-data\_save[,c(1,2,3,grep("L1",colnames(data\_save)))]

41. head(data\_sf)

42. order<-grep(">6k",colnames(data\_sf))

43. order2<-append(1:3,order)

44. other<-setdiff(4:dim(data\_sf)[2],order2)

45. final<-append(order2,other)

46. data\_sf2<-data\_sf[,final]

47. head(data\_sf2)

48. write.csv(data\_sf2,"finalResult\_top3000.csv",row.names=F)

49.

50.

51. #####draw plot

52. #将想要作图的序列提取出来

53. seq<-"ATTATACTTTAAGTTTTAGGGTACATGTGC"

54. plotDat<-data\_ext[data\_ext$gRNA%in%c(seq),]

55. background\_colors <- c("#f0f0f0", "#ffffff")

56. unique\_categories <- unique(plotDat$subfamily)

57. category\_positions <- as.numeric(factor(unique\_categories))

58. background\_data <- data.frame(

59. xmin = category\_positions - 0.5,

60. xmax = category\_positions + 0.5,

61. ymin = -Inf,

62. ymax = Inf,

63. fill = rep(background\_colors, length.out = length(unique\_categories))

64. )

65. library(ggplot2)

66. options(repr.plot.width =4, repr.plot.height =3)

67. title<-paste("mismatch:<=",mismatch,sep="")

68. p2<-ggplot(data=plotDat) +

69. geom\_rect(data = background\_data, aes(xmin = xmin, xmax = xmax, ymin = ymin, ymax = ymax, fill = fill), alpha = 0.2,

70. show.legend = FALSE ) +

71. geom\_bar(aes(fill=length, y=per, x=subfamily),position='dodge', stat='identity')+

72. scale\_fill\_manual(name = "length",values = c("#98d09d","#fbf398","#f7a895","#9b8191","grey","#ffffff"))+

73. theme\_classic()+

74. #facet\_grid(. ~ gRNA) +

75. scale\_y\_continuous(limits = c(0, 1))+

76. coord\_flip()+

77. labs(title=title,y="Percentage(%)")+

78. theme(plot.title=element\_text(face="bold", #字体

79. color="steelblue", #颜色

80. size=20, #大小

81. hjust=0.5, #位置

82. vjust=0.5,

83. angle=360))

84. p2

85. outputfile<-paste("crRNA\_",seq,".pdf",sep="")

86. pdf(outputfile,width = 4,height = 3)

87. p2

88. dev.off()

89.

**三、合并相似crRNA**

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

1. import pandas as pd

2. import numpy as np

3. from Bio import pairwise2

4. from itertools import combinations

5. from tqdm import tqdm

6. import argparse

7. from concurrent.futures import ProcessPoolExecutor

8. import multiprocessing

9.

10. def read\_sequences\_from\_file(filename):

11. """从文件读取并标准化序列"""

12. with open(filename, 'r') as f:

13. return list({s.strip().upper() for s in f if s.strip()})

14.

15. def calc\_identity(args):

16. """多进程任务函数：计算序列相似度"""

17. i, j, seq1, seq2, threshold = args

18. len1, len2 = len(seq1), len(seq2)

19. max\_len = max(len1, len2)

20.

21. # 快速长度过滤

22. if min(len1, len2) < 0.75 \* max\_len:

23. return (i, j, False)

24.

25. # 精确比对

26. align = pairwise2.align.globalxx(seq1, seq2, one\_alignment\_only=True)[0]

27. matches = align.score

28. similarity = matches / max\_len \* 100

29. return (i, j, similarity >= threshold)

30.

31. def cluster\_sequences(sequences, threshold=75, workers=None):

32. """多进程聚类主函数"""

33. n = len(sequences)

34. manager = multiprocessing.Manager()

35. adjacency = manager.dict({(i,j): False for i, j in combinations(range(n), 2)})

36.

37. # 生成任务列表

38. tasks = [(i, j, sequences[i], sequences[j], threshold)

39. for i, j in combinations(range(n), 2)]

40.

41. # 多进程处理

42. with ProcessPoolExecutor(max\_workers=workers) as executor:

43. results = list(tqdm(executor.map(calc\_identity, tasks),

44. total=len(tasks),

45. desc="Processing pairs"))

46.

47. # 构建邻接矩阵

48. adj\_matrix = np.eye(n, dtype=bool)

49. for i, j, match in results:

50. if match:

51. adj\_matrix[i][j] = True

52. adj\_matrix[j][i] = True

53.

54. # 查找连通分量

55. visited = set()

56. groups = []

57. for i in range(n):

58. if i not in visited:

59. component = set(np.where(adj\_matrix[i])[0])

60. visited.update(component)

61. groups.append(sorted(component, key=lambda x: -len(sequences[x])))

62.

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

64.

65. def create\_report(groups):

66. """生成报告数据（同前）"""

67. report = []

68. for group\_id, group in enumerate(groups, 1):

69. group\_size = len(group)

70. for seq in group:

71. report.append({

72. "GroupID": f"Group\_{group\_id:03d}",

73. "Sequence": seq,

74. "Length": len(seq),

75. "GC%": round((seq.count("G") + seq.count("C"))/len(seq)\*100, 2),

76. "GroupSize": group\_size,

77. "IsSingleton": group\_size == 1

78. })

79. return pd.DataFrame(report)

80.

81. def main():

82. parser = argparse.ArgumentParser(description="crRNA序列聚类工具")

83. parser.add\_argument("-i", "--input", required=True, help="输入文件路径（crRNA\_list.txt）")

84. parser.add\_argument("-o", "--output", default="crRNA\_groups.xlsx", help="输出Excel文件路径")

85. parser.add\_argument("-t", "--threshold", type=float, default=75,

86. help="相似度阈值（百分比）")

87. parser.add\_argument("-w", "--workers", type=int,

88. default=multiprocessing.cpu\_count()-1,

89. help="并行进程数（默认CPU核心数-1）")

90. args = parser.parse\_args()

91.

92. # 读取和处理数据

93. print(f"正在读取序列文件：{args.input}")

94. sequences = read\_sequences\_from\_file(args.input)

95. print(f"发现 {len(sequences)} 条唯一序列")

96.

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

98. groups = cluster\_sequences(sequences, args.threshold, args.workers)

99.

100. # 生成报告

101. print("生成分析报告...")

102. df = create\_report(groups)

103.

104. # 输出Excel

105. with pd.ExcelWriter(args.output) as writer:

106. df.to\_excel(writer, sheet\_name="Full\_Report", index=False)

107. summary = df.groupby("GroupID").agg(

108. Sequences=("Sequence", lambda x: "\n".join(x)),

109. Count=("GroupID", "count"),

110. Avg\_Length=("Length", "mean"),

111. Avg\_GC=("GC%", "mean")

112. ).reset\_index()

113. summary.to\_excel(writer, sheet\_name="Group\_Summary", index=False)

114. df[df["IsSingleton"]].to\_excel(writer, sheet\_name="Singletons", index=False)

115.

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

117.

118. if \_\_name\_\_ == "\_\_main\_\_":

119. main()

120.

运行命令行：

1. python crRNA\_cluster.py -i L1PA1\_4\_sgRNA.top100.txt -o results.xlsx -t 75 -w 8