One-Step DNA Pattern Search

Chen-Hanson Ting

SVFIG

October 27, 2018

Summary

- String compare and search
- Bacterial genomes
- Optimized Search
- Pearls
- Necklaces
- Examples

What is Information?

Repeated patterns

Information in Genomes

- Coding DNA
  - Genes with protein code
- Noncoding DNA
  - ???

Gene Expressions

- Coding DNA
  - Messenger RNA
  - Transfer RNA
  - MicroRNA
- Noncoding DNA
  - Noncoding RNA

Noncoding DNA/RNA

- Collections of microRNAs
- Collections of microRNA-like information
  - Pearls: 20-base patterns
  - Necklaces: clusters of pearls

Exhaustive Search of Pearls

- Identify all pearls, unique and repeated 20-base patterns in genomes.
- Identify all necklaces, which are clusters of adjacent pearls.
- Pearls are related to microRNAs.
- Necklaces are related to noncoding RNAs

Simple Python Search

- Sequencing through all 20-base patterns in genomes.
- For each 20-base pattern, search its repeats.

Simple Python Search

```
def encode(file):
    fin=open(file+'.txt','r')
    fout=open(file+'_out.dat','w')
    gen=fin.read()
    genome=gen[0:len(gen)-1]+gen[0:20]
    end=len(genome)
    fin.close()
    for i in range(len(gen)):
        g=genome[i:i+20]
        print(i,g)
        for j in range(i+1,len(gen)):
            if genome[j:j+20]==g:
                fout.write(str(i)+'\t'+g+'\n')
                break
        for j in range(i+1,len(gen)):
            if genome[j:j+20]==g:
                fout.write(str(j)+'\t'+g+'\n')
    fout.close()
encode('hsa_mito')
```

Advanced Python Search

- Break genome file into a list of 20-base patterns.
- Remove duplicated patterns with SET function.
- Save only duplicated patterns as pearls.
- Mark all pearls in genome to identify necklaces.

```
def encode(file):
    fin=open(file+'.dat','r')
    fout=open(file+'_out.dat','w')
    genome=fin.read()
    end=len(genome)
    fin.close()
    genom=[genome[i:i+20] for i in range(end)]
    gen=list(genom)
    trim=list(set(gen))
    for i in range(len(trim)):
        gen.remove(trim[i])
    trim=list(set(gen))
    last=total=0
    gen=list(genom)
    for i in range(len(gen)):
        if (i-20)>last:
            if gen[i] in trim:
                fout.write(str(i)+'\t'+gen[i]+'\n')
                last=i
                total+=1
    fout.close()
encode('nasuia')
encode('ruddii')
encode('nasuia')
encode('ruddii')
```

Advanced Python Search

- Python code is very simple because of the rich tool sets.
- It worked very well with small bacterial genomes.
- It took forever to search E. coli genome of 4.6 Mbp.

Advanced Forth Search

- **Two critical text search routines were coded in assembly:**
- **Compare two strings:**

```
Compare( s1 s2 n -- -1|0|1 )
```

- **Look up a pattern in a long string:**

```
look ( p n1 s n2 -- addr|0 )
```

Optimized Compare

```
code Compare ( string pattern n -- -1 | 0 | 1 )
( 044730 8B CB                ) mov    ecx , ebx
( 044732 8B 7D 00             ) mov    edi , 0 [ebp]
( 044735 8B 75 04             ) mov    esi , 4 [ebp]
( 044738 8D 6D 08             ) lea    ebp , 8 [ebp]
( 04473B 33 DB                ) xor    ebx , ebx
( 04473D 81 E1 FF 00 00 00 ) and    ecx , # dword $FF
( 044743 F3                   ) repz
( 044744 A6                   ) cmpsb
( 044745 77 03                ) ja     short @@1
( 044747 72 04                ) jb     short @@2
( 044749 C3                   ) ret    near
( 04474A                      ) @@1:
( 04474A FF C3                ) inc    ebx
( 04474C C3                   ) ret    near
( 04474D                      ) @@2:
( 04474D FF CB                ) dec    ebx
( 04474F C3                   ) ret    near
end-code
```

Optimized Look

```
code look ( pattern n1 string n2 -- match|0 )
( 65651E4                   ) @@1:
( 65651E4 FF CB             )   dec     ebx
( 65651E6 7E 19             )   jle     short @@3
( 65651E8 8B 7D 00          )   mov     edi , 0 [ebp]
( 65651EB 8B 4D 04          )   mov     ecx , 4 [ebp]
( 65651EE 8B 75 08          )   mov     esi , 8 [ebp]
( 65651F1 F3                )   repz
( 65651F2 A7                )   cmpsd
( 65651F3 75 07             )   jnz     short @@2
( 65651F5 8B 5D 00          )   mov     ebx , 0 [ebp]
( 65651F8 8D 6D 0C          )   lea     ebp , $C [ebp]
( 65651FB C3                )   ret     near
( 65651FC                   ) @@2:
( 65651FC FF 45 00          )   inc     0 [ebp] dword
( 65651FF EB E3             )   jmp     short @@1
( 6565201                   ) @@3:
( 6565201 8D 6D 0C          )   lea     ebp , $C [ebp]
( 6565204 C3                )   ret     near
end-code
```

Advanced Forth Search

- Break genome into 4096 threads, each associated with an unique 6-base pattern.
- Search repeated 20-base patterns in each thread.
- Threads were coded in run-length-code.

Advanced Forth Search

- Each thread can be processed after it is produced.
- Run-length coding is not necessary because thread data and links are not written into external files.
- Searching is greatly accelerated.

Advanced Forth Search

- SCAN-LOOK links the same 6-base patterns into a thread, a simple address list.
- The address list is left in PAD.

SCAN-LOOK

```
variable LinkPointer

: SCAN-look ( -- )
   pad LinkPointer !
   Genome @ ( a )
   begin ( a )
      dup source pattern rot ( a p lim a )
      Genome-end @ over - cell+ ( a p lim a len )
      LOOK ( a a1|0 )
   dup while  \ dup Genome @ - EXTputN
      pattern + dup LinkPointer @ !
      4 LinkPointer +!
      swap drop
   repeat
   LinkPointer @ ! drop
  ;
```

Advanced Forth Search

- PASS1 scans a thread, write out the first matching pattern.
- PASS2 scans a thread, write out the rest of matching patterns.
- SCAN-PASS2 calls PASS1 and PASS2 to write all repeated patterns to an output file.

PASS1

```
: pass1 ( ptr -- , write 1st match )
   dup @ target ! ( ptr )
   begin cell+ ( ptr1 )
      dup @ dup ( ptr1 addr addr )
   while ( ptr1 addr )
      dup target @ compare-len compare ( ptr1 addr f )
      if drop else
         target @ pattern - dup Genome @ - EXTputN
         Genome-limit EXTputLine CRLF
         2drop exit ( report 1st match )
      then
   repeat
   2drop ;
```

PASS2

```
: pass2 ( ptr -- , write 1st match )
   dup @ target ! ( ptr )
   begin cell+ ( ptr1 )
      dup @ dup ( ptr1 addr addr )
   while
      dup target @ compare-len compare ( ptr1 addr f )
      if drop else
         pattern - dup Genome @ - EXTputN
         Genome-limit EXTputLine CRLF
      then
   repeat
   2drop ;
```

SCAN-PASS2

```
( Scan a thread. Report all matches. )

: SCAN-pass2 ( -- )
  pad ( ptr )
  begin dup @ ( ptr addr )
  while ( ptr )
     dup pass1 ( ptr )
     dup pass2 ( ptr )
     cell+
  repeat
  drop ;
```

Advanced Forth Search

- DECODE generates 4096 threads. Write all repeated patterns to output file.
- MATCHES opens a genome file, and writes all repeated patterns to output file.

DECODE

```
( Decode 4096 threads. Report all matches. )

: decode ( -- )
   0 ( code )
   $1000 for aft
      dup ACGT ( code )
      scan-look
      SCAN-pass2 ( code )
      1+ ( code+1 )
   then next drop
   ;
```

MATCHES

```
( Encode a genome file, and decode all repeated matching patterns.
)
: matches ( matches genome -- )
  Genomeopen
  Genome-len @ Genome @ + Genome-end !
  uppercase
  EXTopen decode EXTclose
  Genomeclose Deltaclose
  ;
: bacteria
  z" ruddii_matches_1.txt" z" ruddii_data.txt" matches
  z" nasuia_matches_1.txt" z" nasuia_data.txt" matches
  z" genitalium_matches_1.txt" z" genitalium_data.txt" matches
  z" equitans_matches_1.txt" z" equitans_data.txt" matches
  z" acido_matches_1.txt" z" acido_data.txt" matches
  z" ecoli_matches_1.txt" z" ecoli_data.txt" matches
  ;
```

Bacteria Studied

Nasuia	(Nasuia deltocephalinicola)
Ruddii	(Candidatus Carsonella ruddii)
Equitans	(Nanoarchaeum equitans)
Genitalium	(Mycoplasma genitalium)
Acido	(Lactobacillus acidophilus)
Ecoli	(Escherichia coli)

Bacterial Genomes

Nasuia	112,091
Ruddii	173,806
Equitans	490,885
Genitalium	580,076
Acido	1,993,560
Ecoli	4,641,652

Genome Search Time

		Seaarch Time (min)	
Bacteria	Bp	Forth	Python
Nasuia	112,091	1	5
Ruddii	173,806	1	11
Equitans	490,885	2	1:07
Genitalium	580,076	2	1:18
Acido	1,993,560	9	13:23
Ecoli	4,641,652	25	????

Pearls and Necklaces

- Pearls are assigned unique IDs.
- Pearls are listed with gene annotations.
- Necklaces are listed separately in spread sheets.

Pearls

- Huge numbers of repeated patterns in consecutive locations, caused by duplicated genes. These patterns must be deleted.
- 20 base patterns outside of genes are called Pearls.

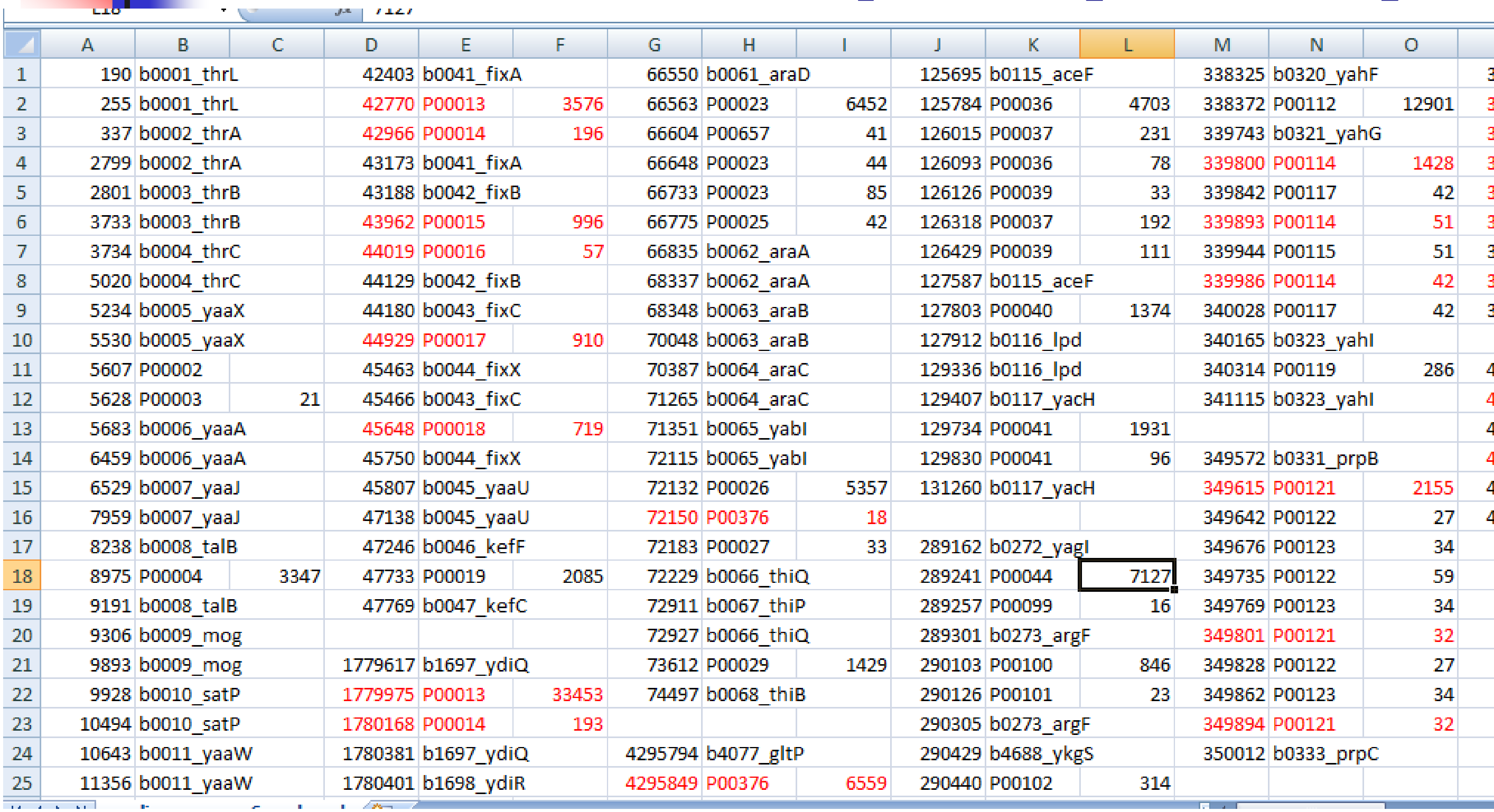
Pearls

- Pearls are extracted, and each assigned a unique ID.
- All pearls are identified in a bacterial genome.
- Clusters of pearls can then be identified as necklaces.

Necklaces

- Lots of pearls appear in clusters.
- Clusters of consecutive pearls are called Necklaces.
- Necklaces are often found in non-coding DNA, but may be found in coding regions.
- Necklaces probably represent high level functions in a cell computer.

Necklace Examples (E. coli)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	190	b0001_thrL		42403	b0041_fixA		66550	b0061_araD		125695	b0115_aceF		338325	b0320_yahF	
2	255	b0001_thrL		42770	P00013	3576	66563	P00023	6452	125784	P00036	4703	338372	P00112	12901
3	337	b0002_thrA		42966	P00014	196	66604	P00657	41	126015	P00037	231	339743	b0321_yahG	
4	2799	b0002_thrA		43173	b0041_fixA		66648	P00023	44	126093	P00036	78	339800	P00114	1428
5	2801	b0003_thrB		43188	b0042_fixB		66733	P00023	85	126126	P00039	33	339842	P00117	42
6	3733	b0003_thrB		43962	P00015	996	66775	P00025	42	126318	P00037	192	339893	P00114	51
7	3734	b0004_thrC		44019	P00016	57	66835	b0062_araA		126429	P00039	111	339944	P00115	51
8	5020	b0004_thrC		44129	b0042_fixB		68337	b0062_araA		127587	b0115_aceF		339986	P00114	42
9	5234	b0005_yaaX		44180	b0043_fixC		68348	b0063_araB		127803	P00040	1374	340028	P00117	42
10	5530	b0005_yaaX		44929	P00017	910	70048	b0063_araB		127912	b0116_lpd		340165	b0323_yahI	
11	5607	P00002		45463	b0044_fixX		70387	b0064_araC		129336	b0116_lpd		340314	P00119	286
12	5628	P00003	21	45466	b0043_fixC		71265	b0064_araC		129407	b0117_yacH		341115	b0323_yahI	
13	5683	b0006_yaaA		45648	P00018	719	71351	b0065_yabI		129734	P00041	1931			
14	6459	b0006_yaaA		45750	b0044_fixX		72115	b0065_yabI		129830	P00041	96	349572	b0331_prpB	
15	6529	b0007_yaaJ		45807	b0045_yaaU		72132	P00026	5357	131260	b0117_yacH		349615	P00121	2155
16	7959	b0007_yaaJ		47138	b0045_yaaU		72150	P00376	18				349642	P00122	27
17	8238	b0008_talB		47246	b0046_kefF		72183	P00027	33	289162	b0272_yagI		349676	P00123	34
18	8975	P00004	3347	47733	P00019	2085	72229	b0066_thiQ		289241	P00044	7127	349735	P00122	59
19	9191	b0008_talB		47769	b0047_kefC		72911	b0067_thiP		289257	P00099	16	349769	P00123	34
20	9306	b0009_mog					72927	b0066_thiQ		289301	b0273_argF		349801	P00121	32
21	9893	b0009_mog		1779617	b1697_ydiQ		73612	P00029	1429	290103	P00100	846	349828	P00122	27
22	9928	b0010_satP		1779975	P00013	33453	74497	b0068_thiB		290126	P00101	23	349862	P00123	34
23	10494	b0010_satP		1780168	P00014	193				290305	b0273_argF		349894	P00121	32
24	10643	b0011_yaaW		1780381	b1697_ydiQ		4295794	b4077_gltP		290429	b4688_ykgS		350012	b0333_prpC	
25	11356	b0011_yaaW		1780401	b1698_ydiR		4295849	P00376	6559	290440	P00102	314			

Pearls and Necklaces

- In my cell computer model,
  - Pearls and microRNAs are instructions.
  - Protein-coding genes are primitive instructions which produces messengerRNAs.
  - Necklaces are high level instructions with lists of pearls.

Questions?

Thank You!