Automatiserad primerdesign för kandidatgener

Therese Thomsson

Therese.thomsson@outlook.com

Handledare: Mats Hansson

Molekylärbiologi: Kandidatarbete

MOBK01

Lunds universitet

2021/01/15

Sammanfattning

Genom att använda algoritmer och mjukvara kan gamla metoder inom molekylärbiologin effektiviseras. Målet med arbetet var att framställa ett program som automatiserar produktionen av överlappande primrar för kandidatgener. Genom att ta tre olika inputs från användaren så skulle förslag till unika primerpar framställas. Programmet kodades i programmeringsspråket python och biblioteket Primer3 användes. Programmet kan i nuvarande form producera 10 stycken primerpar som är unika och uppfyller vissa krav på dess egenskaper, såsom smälttemperatur och guanin-cytosin%. Programmet är i nuläget inte användarvänligt och svårt att hantera om användaren jobbar med en organism med ett stort genom.

Inledning

Genom nya uppfinningar och förbättrade metoder insamlas mer biologisk data än någonsin i dagens samhälle, detta har resulterat i en ökad efterfrågan för system som snabbt kan hantera och sortera data på ett effektivt sätt. På grund av denna ständigt ökande efterfrågan för moderna system som är anpassade till dagens datamängd har ett tvärvetenskapligt fält växt i kraft, bioinformatiken (Bayat, 2002). Genom att använda sig av algoritmer kan många processer som tidigare behövt utföras manuellt bli automatiserade, vilket gör arbetet inom många olika biologiska forskningsområden enklare och minimerar misstag som kan uppstå på grund av den mänskliga faktorn. Bioinformatik har blivit ett kraftfullt verktyg inom många olika biologiska områden, och inte minst inom genetisk verksamhet. För att kunna studera en gen så behöver det relevanta DNAt kopieras upp, detta kan utföras med PCR (*polymerase chain reaction*), vilket är en teknik för att kopiera upp DNA-sekvenser till ett större antal (Bayat, 2002). För att kunna utföra en PCR behöver relevanta primrar för genen designas. Många kandidatgener är mycket stora, och polymeraset i en PCR kan enbart skapa fragment av en begränsad storlek in vitro (Ishino & Ishino, 2014). På grund av detta måste många olika primrar designas så att fragmenten har överlappande områden. Detta kan vara en tidskrävande process då varje primer måste vara unik jämfört med genomet och ha en bra guanin-cytosin%, smältpunkt samt andra faktorer (Yang et al., 2006).

Som programmeringsspråk har Perl och R varit normen inom bioinformatiken, men då andra industrier börjar använda sig allt mer av python så har det också blivit mer integrerat för bioinformatiska syften. Python är ett nybörjarvänligt och enkelt språk men samtidigt väldigt kraftfullt. En av pythons styrkor är tillgängligheten till moduler och paket, vilket är pythonfiler med funktioner man kan implementera i olika program. En av dessa moduler är Primer3, vilket är ett populärt bibliotek med funktioner som hjälper vid primerdesign.

I det här arbetet ska denna process gå från att manuellt designa varje individuell primer till en fullt automatiserad process där enbart referens genomet samt regionen och den önskade längden på fragmenten behöver uppges. Arbetet kommer att bestå av skapandet av ett program i kodspråket python som både ger förslag på primrar och tar hänsyn till relevanta faktorer som gör primrarna passande till PCR.

Metod

Programmeringsspråket python (version 3.9.1) och IDEn Spyder har använts.
För primerdesign användes Primer3.

Programmet kan delas upp i 3 större mer överskådliga delar: input, primerdesign och output. I den första delen ska programmet ta tre stycken inputs av användaren: det relevanta genomet, sekvens av intresse samt önskad fragmentstorlek. I den andra delen ska primrar genereras över den angivna sekvensen, detta med hänsyn till den angivna fragmentstorleken samt att fragmenten ska vara överlappande. Programmet måste även ta hänsyn till olika egenskaper hos primrarna som t.ex smälttemperatur och guanin-cytosin%. Sedan ska de ospecifika primrarna sorteras ut genom att jämföra de med det angivna genomet. I den sista delen ska programmet mata ut förslag till primers samt relevant info.

Primer3 laddades ner och installerades med hjälp av pakethanteraren pip genom att ge kommandot "*pip install primer3-py*" i kommandotolken.

Input:
Import av Primer3 var det första som implementeras in i koden. Sedan skapades input kommandon för genomet, sekvensen samt fragmentstorlek (Figur 1).
Fragmentstorleken behövde omvandlas från en string, vilket är vad python ser som ett fält (eng. Array), till en integer, vilket python behandlar som ett nummer. Detta gjordes med int kommandot.

```
1  import primer3
2  genome_sequence=input("Enter the Genome Sequence in FASTA format:\n")
3  gene_seq=input("\nEnter sequence of our Gene of Interest:\n")
4  frag_length=int(input("\nEnter fragment length (above 50):"))
```

Figur 1: Den första delen av programmet som tar emot input från användaren. Då fragmentstorleken kommer användas i senare skede för att beräkna positionen av primrarna så behövde den omvandlas från en string till en integer (rad 4).

Primerdesign:
Två dictionaries skapas, ett med sekvensparametrar (Figur 2) och ett med primerparametrar (Figur 3). I sekvensparametrarna sätts sekvensen given av användaren som sekvenstemplatet. Möjligheten att utesluta en region ur templatet lades in, dock har användaren ingen möjlighet att utnyttja funktionen i nuläget.

```
6  a={'SEQUENCE_ID': 'Pick distinctive primers',
7     'SEQUENCE_TEMPLATE': gene_seq,
8     'SEQUENCE_EXCLUDED_REGION': [0, 0]}
```

Figur 2: Tre olika sekvensparametrar lades in. Den första är en identifierare för primrarna, den andra är att sätta den angivna sekvensen som sekvenstemplatet. Den sista parametern är möjligheten att exkludera en viss region ur templatet.

Primerparametrarna skapades i ett separat dictionary. Allt programmet tar hänsyn till vid primerdesign kodades in här, såsom primerstorlek, smälttemperatur, guanin-cytosin%, antal primrar e.t.c. För varje enskild parameter se appendix. För PRIMER_SEQUENCING_SPACING, vilket definierar avståndet från 3'-änden av en primer till 3'-änden av nästa primer på den samma DNA kedja, användes fragmentstorleken given av användaren dividerat på 1.2, vilket ger överlappande fragment.

```
10  b= {
11                  'PRIMER_TASK': 'pick_sequencing_primers',
12                  'PRIMER_PICK_LEFT_PRIMER':1,
13                  'PRIMER_PICK_INTERNAL_OLIGO': 0,
14                  'PRIMER_PICK_RIGHT_PRIMER': 1,
15                  'PRIMER_NUM_RETURN': 10,
16                  'PRIMER_OPT_SIZE': 20,
17                  'PRIMER_MIN_SIZE': 18,
18                  'PRIMER_MAX_SIZE': 25,
19                  'PRIMER_OPT_TM': 60.0,
20                  'PRIMER_MIN_TM': 57.0,
21                  'PRIMER_MAX_TM': 63.0,
22                  'PRIMER_MIN_GC': 20.0,
23                  'PRIMER_MAX_GC': 80.0,
24                  'PRIMER_MAX_POLY_X': 5,
25                  'PRIMER_SEQUENCING_INTERVAL':int(frag_length/1.2),
26                  'PRIMER_SEQUENCING_SPACING':frag_length,
27                  'PRIMER_SEQUENCING_ACCURACY':20,
28                  'PRIMER_SEQUENCING_LEAD':50,
29                  'PRIMER_SALT_MONOVALENT': 50.0,
30                  'PRIMER_DNA_CONC': 50.0,
31                  'PRIMER_MAX_NS_ACCEPTED': 0,
32                  'PRIMER_MAX_SELF_ANY': 12,
33                  'PRIMER_MAX_SELF_END': 8,
34                  'PRIMER_PAIR_MAX_COMPL_ANY': 12,
35                  'PRIMER_PAIR_MAX_COMPL_END': 8,
36                  'PRIMER_PRODUCT_SIZE_RANGE': [100,200],
37                 }
```

Figur 3: Dictionary för de olika parametrarna som används vid primerdesign.

Genom att sätta dictionariet med primerdesign som ett globalt argument så kan designPrimers kallas med enbart SeqArg för att optimera koden (Figur 4). Detta kommer generera ett nytt dictionary med genererade primrar. Nya dictionaries skapas för att lagra de genererade primerparen samt deras Guanin-Cytosin% och smälttemperatur.

```
40  x=primer3.setP3Globals(b,misprime_lib=None,mishyb_lib=None)
41  p=primer3.bindings.designPrimers(a,debug=False)
```

Figur 4: Primrar genereras vid linje 41 med de tidigare satta parametrarna.

Output:

Två nya listor skapas (Figur 5) så att primrarna som ska sitta på 3′-5′och 5′-3′ strängarna lagras i separata listor. För att ta bort upprepade sekvenser användes *list_1=list(dict.fromkeys(list_1))* för list_1, och motsvarande för list_2.

```
61  list_1=[p['PRIMER_LEFT_0_SEQUENCE'],p['PRIMER_LEFT_1_SEQUENCE'],p['
62  list_1=list(dict.fromkeys(list_1))
63
64  list_2=[p['PRIMER_RIGHT_0_SEQUENCE'],p['PRIMER_RIGHT_1_SEQUENCE'],p
65  list_2=list(dict.fromkeys(list_2))
```

Figur 5: I linje 61 och 64 sparas primrarna nu i separata listor. Linje 62 och 65 eliminerar upprepade sekvenser.

För att kunna fastställa att primrarna är unika behöver de jämföras med genomet. Då sekvensen där primrarna har genererats från redan finns i genomet så var den delen tvungen att klippas ut. Detta utfördes genom att den första indexen av sekvensen defineras med *ind=genome_sequence.index(gene_seq)* och den sista indexen med *last=ind+len(gene_seq).* För att sedan exkludera den delen från genomet skapades en ny string genom att skriva *seq=genome_sequence[0:ind]+genome_sequence[last:-1]* (Figur 6).

```
70  ind=genome_sequence.index(gene_seq)
71  last=ind+len(gene_seq)
72  seq=genome_sequence[0:ind]+genome_sequence[last:-1]
```

Figur 6: Genom att definiera den första och sista indexen av sekvensen kan den delen klippas ut från genomet. Det "nya" genomet som saknar sekvensen sparas i en nya string.

För att kunna sortera bort ospecifika reverse primrar så måste de jämföras med genomet skrivet i 3′-5′. För att omvandla genomet från 5′-′3 till 3′-5′ användes original-genomet exklusive sekvensregionen och varje nukleotid byttes ut till sin motsvarande motpart. Detta utfördes med en elif loop (Figur 7) och det nya genomet sparades i en separat string.

```
79  my_seq=genome_sequence[0:ind]+genome_sequence[last:-1]
80  revgenome=""
81  for letter in my_seq:
82      if letter == "T":
83              revgenome+="A"
84      elif letter == "A":
85               revgenome+="T"
86      elif letter == "C":
87               revgenome+="G"
88      elif letter == "G":
89               revgenome+="C"
90      elif letter == ' ':
91          revgenome+=""
92
93  my_seq=revgenome
```

Figur 7: En tom string kallad revgenome skapas i rad 80. Sedan har en elif loop skapats där python läser av genomet bokstav för bokstav och lägger till motsvarande nukleotid i revgenome.

Sedan jämförs både forward och reverse primrarna med respektive genom med hjälp av en for loop (Figur 8). Om den exakta kombinationen och ordningen på nukleotiderna inte återfinns någonstans i genomet läggs de unika primrarna till i en lista som sedan printas ut till användaren. Om ingen unik primer upptäcks printas istället ett meddelande ut som informerar användaren om detta.

```
95    l_1=[]
96    for for_pri in list_1:
97        if for_pri not in seq:
98            l_1.append(for_pri)
99
100
101   if len(l_1)==0:
102       print("\nNo unique forward primers found")
103   else:
104       print("\nUnique Forward primers:\n",l_1)
```

Figur 8: rad 95 till 98 jämför varje forward primer till genomet. Rad 101 till 104 printar antingen ut de utvalda primrarna eller meddelar användaren att ingen unik forward primer hittades.

Resultat

Det färdigställda programmet ger förslag på upp till 10 stycken överlappande primerpar och de är unika gentemot organismens genom. Dessa primerpar verkar följa de angivna parametrarna, men vissa parametrar (som ex. smälttemperatur) är svåra att konstatera om de stämmer då ingen in vitro testing har utförts. Om sekvensen som anges är av en större längd (cirka 7000 bp) har primrarna en tendens att genereras bara i sekvensens början. Genomet anges genom copy/paste istället för att öppna/dra in en FASTA fil i programmet. Informationen som ges till användaren är två listor, den ena är de genererade forward primrarna och den andra reverse primrarna. Om inga unika primerpar hittas printas "No unique forward primers found" respektive "No unique reverse primers found".

Diskussion

Målet med det här arbetet var att förenkla och påskynda arbetet vid primerdesign. Programmet fungerar i den grad att den anger unika primrar, men programmet är i sitt nuvarande skede långt från att förenkla standardprocessen vid primerdesign för kandidatgener. Programmet är dåligt optimerat på många vis, men det största problemet är hur genomet anges. Just nu anges genomet som en vanlig input, alltså genom copy-paste. Detta är inte optimalt då de flesta genom är relativt stora, en textfil blir ofta över 4 GB stort. Detta är ett problem då de flesta texthanterhingsprogram kraschar innan den storleken är nådd. Detta skulle kunna justeras genom att låta användaren dra in genomet som en FASTA fil, som programmet sedan öppnar, hoppar till första > tecknet och läser in samt sparar allt efter det. Detta skulle kunna implementeras genom att byta ut input funktionen till ett open funktion.

Programmet jämför också primrarna då primerdesign redan skett, vilket skapar problem i senare i koden. Då de potentiella primrarna redan lagrats i en lista kan de gå förlorade ifall de vid ett senare tillfälle visar sig vara identiska till en del av genomet, vilket kan resultera i för få primrar. Detta skulle kunna lösas på på två sätt.

1. Tillverka en loop som genererar en ny primer varje gång en primer tas bort.

2. Helt ta bort systemet med att spara ett fixerat antal primrar.

Av dessa lösningar är alternativ 2 att föredra. Detta på grund av att det medkommer en rad problem med att ha en fixerad mängd primrar som kan sparas. Ett av problemen är utifall om mängden primrar som behöver genereras går över den fixerade gränsen vilket skulle resultera i att primrarna inte täcker hela sekvensen. Detta skulle naturligtvis kunna lösas genom att generera en stor mängd listor som kan spara en större mängd primrar, men den här metoden skulle inte vara optimal då samma problem skulle kunna uppstå igen samt koden skulle inte vara optimerad. Istället skulle ett system kunna implementeras som designar en primer i början av sekvensen. Denna primer skulle jämföras med genomet, om den visar sig vara unik sparas den i en lista och programmet hoppar sedan en viss distans och försöker generera en ny primer. Om primern är ospecifik fortsätter programmet att hoppa en nukleotid i taget tills den hittar en primer som uppfyller parametrarna och är unik. Denna process skulle då fortsätta tills hela sekvensen har lästs av.

Ett annat problem är hur programmets uppvisar outputen. De genererade primrarna matas ut i två separata listor, en lista med forward och en lista med reverse där den första forward korresponderar med den första reverse. För att göra det enklare att läsa och förstå hade paren kunnat printas ut var för sig. För att göra programmet mer användarvänligt hade även ett användargränssnitt kunnat tillverkats.

Huvudanledningen som ledde till dessa problem var en för stor optimism och tidsbrist. Trots programmets brister så fungerar det och kan användas för primerdesign. Med mer tid och arbete så kan programmet nog förbättras till den grad att det underlättar arbetet vid framställning av primerpar.

Referenser

Bayat A. (2002). Science, medicine, and the future: Bioinformatics. BMJ (Clinical research ed.), 324(7344), 1018–1022. https://doi.org/10.1136/bmj.324.7344.1018

Garibyan, L., & Avashia, N. (2013). Polymerase chain reaction. The Journal of investigative dermatology, 133(3), 1–4. https://doi.org/10.1038/jid.2013.1

Green, S. J., Venkatramanan, R., & Naqib, A. (2015). Deconstructing the polymerase chain reaction: understanding and correcting bias associated with primer degeneracies and primer-template mismatches. PloS one, 10(5), e0128122. https://doi.org/10.1371/journal.pone.0128122

Ishino, S., & Ishino, Y. (2014). DNA polymerases as useful reagents for biotechnology - the history of developmental research in the field. Frontiers in microbiology, 5, 465. https://doi.org/10.3389/fmicb.2014.00465

Kwon, J. M., & Goate, A. M. (2000). The candidate gene approach. Alcohol research & health : the journal of the National Institute on Alcohol Abuse and Alcoholism, 24(3), 164–168.

Lee, P. Y., Costumbrado, J., Hsu, C. Y., & Kim, Y. H. (2012). Agarose gel electrophoresis for the separation of DNA fragments. Journal of visualized experiments : JoVE, (62), 3923. https://doi.org/10.3791/3923

Pearson W. R. (2016). Finding Protein and Nucleotide Similarities with FASTA. Current protocols in bioinformatics, 53, 3.9.1–3.9.25. https://doi.org/10.1002/0471250953.bi0309s53

Russell, P. H., Johnson, R. L., Ananthan, S., Harnke, B., & Carlson, N. E. (2018). A large-scale analysis of bioinformatics code on GitHub. PloS one, 13(10), e0205898. https://doi.org/10.1371/journal.pone.0205898

Yang, X., Scheffler, B. E., & Weston, L. A. (2006). Recent developments in primerdesign for DNA polymorphism and mRNA profiling in higher plants. Plant methods, 2(1), 4. https://doi.org/10.1186/1746-4811-2-4

Rozen S, Skaletsky H (2000) Primer3 on the WWW for general users and for biologist programmers. In: Krawetz S, Misener S (eds) Bioinformatics Methods and Protocols: Methods in Molecular Biology. Humana Press, Totowa, NJ, pp 365-386

Appendix:

De olika primerparametrarna och vad de innebär:

'PRIMER_TASK': 'pick_sequencing_primers

PRIMER_TASK: säger till primer3 vad för uppgift som ska utföras. pick_sequencing_primers ger distansen mellan 3'-änden från en primer till "trace signal" är kan läsas av.

'PRIMER_PICK_LEFT_PRIMER':1,

Utifall värdet är 1 (alltså inte ett nollvärde) försöker primer3 att en vänster primer. Standard boolean värdet är 1.

.

'PRIMER_PICK_INTERNAL_OLIGO': 0,

Utifall värdet är 1 (alltså inte ett nollvärde) försöker primer3 välja en hybridiseringssond vid en intern oligo för att kunna detektera PCR-produkten. Standard boolean värdet är 0.

'PRIMER_PICK_RIGHT_PRIMER': 1,

Utifall värdet är 1 (alltså inte ett nollvärde) försöker primer3 att en vänster primer. Standard boolean värdet är 1.

'PRIMER_NUM_RETURN': 10'

Maximala antalet primerpar som kan genereras och ges till användaren (i detta fall 10). Dessa väljs ut och sorteras på värdet för objektivfunktionen (mindre tal indikerar är bättre primerpar) Storleken på den här parametern påverkar körtiden (större värde orsakar längre körtid.)

'PRIMER_OPT_SIZE': 20,

Den önskade längden på varje enskild primer, Primer3 kommer försöka hålla sig så nära detta värde som möjligt. I detta fall valdes 20, vilket tillåter hög specificitet men samtidigt enkel hybridisering/annelering.

'PRIMER_MIN_SIZE': 18,

Primer3 kommer inte generera en primer under den här storleken.

'PRIMER_MAX_SIZE': 25,

Primer3 kommer inte generera en primer över den här storleken.

'PRIMER_OPT_TM': 60.0,

Önskad smälttemperaturen i Celsius.

'PRIMER_MIN_TM': 57.0,

Primer3 kommer inte generera en primer med smälttemperatur under det här värdet.

'PRIMER_MAX_TM': 63.0,

Primer3 kommer inte generera en primer med smälttemperatur över det här värdet.

'PRIMER_MIN_GC': 45.0,

Lägsta guanin-cytosin% tillåtet för en primer.

'PRIMER_MAX_GC': 80.0,

Högsta guanin-cytosin% tillåtet för en primer.

'PRIMER_MAX_POLY_X': 5,

Största acceptabla längden för mononukleotidupprepning. Värdet lades som 5, och det betyder att maximalt så får en typ av nukleotid upprepas max 5 gånger i rad t.ex CCCCC.

'PRIMER_SEQUENCING_INTERVAL': int(frag_length/1.2),

Avståndet mellan 3'-änden av en primer till 3'-änden av nästa primer på motsvarande DNA sträng. används enbart om PRIMER_TASK = pick_sequencing_primers.

'PRIMER_SEQUENCING_SPACING':frag_length,

Avståndet mellan 3'-änden av en primern till 3'-änden av nästa primer på samma DNA sträng. används om PRIMER_TASK = pick_sequencing_primers.

'PRIMER_SEQUENCING_ACCURACY':20,

Regionen från den beräknade positionen för 3'-änden till varsin sida definieras och den bästa primern väljs ut. Används om PRIMER_TASK = pick_sequencing_primers.

'PRIMER_SEQUENCING_LEAD':50,

Regionen från 3'-änden av primern till området där "trace signal" är möjliga att läsas av defineras. Används om PRIMER_TASK = pick_sequencing_primers.

'PRIMER_SALT_MONOVALENT': 50.0,

mM konc av monovalenta salt-katjoner (huvudsakligen kaliumklorid) i PCR. Detta används för att beräkna smälttemperatur.

'PRIMER_DNA_CONC': 50.0,

Används för att begränsa oligosmältningstemperaturer.

'PRIMER_MAX_NS_ACCEPTED': 0,

Maxantalet okända nukleotidbaser som får finnas i en enskild primer.

'PRIMER_MAX_SELF_ANY': 12,

Värde på hur sannolikt det är för en primer att hybridisera med sig själv.

'PRIMER_MAX_SELF_END': 8,

Hur sannolikt det är för 3′-änden av primern att binda med sig själv.

'PRIMER_PAIR_MAX_COMPL_ANY': 12,

Sannolikheten att den vänstra primern binder till den högra.

'PRIMER_PAIR_MAX_COMPL_END': 8,

Hur sannolikt det är att 3-′änden av den vänstra primern binder med den högra primern.

'PRIMER_PRODUCT_SIZE_RANGE': [100,200],

Värdena ger storleken på produkten som primerparen skapar.