PENERAPAN ALGORITMA DIVIDE AND CONQUER DALAM SINTESIS DAN KOREKSI DNA

Restya Winda Astari (13506074)

Sekolah Teknik Elektro dan Informatika
Program Studi Teknik Informatika, Institiut Teknologi Bandung
Jalan Ganesha nomor 10 Bandung
e-mail: if16074@students.if.itb.ac.id,
restya_wa@yahoo.com

ABSTRAK

Penelitian di bidang biologi molekuler menimbulkan kebutuhan akan DNA yang panjang. Sintesis dan koreksi DNA memiliki kesulitan karena tingginya kompleksitas dan panjangnya DNA. Pengurangan kompleksitas DNA dapat dilakukan dengan cara membagi-bagi DNA menjadi potongan-potongan yang mudah disintesis dan dikoreksi. Setelah itu, potongan-potongan disatukan untuk membentuk DNA yang utuh.

Algoritma Divide and Conquer merupakan suatu metode penyelesaian masalah dengan cara membagi-bagi masalah menjadi upa-masalah dan menggabungkan solusi upa-masalah menjadi solusi masalah semula. Penerapan algoritma Divide and Conquer dalam sintesis dan koreksi DNA berarti membagi-bagi DNA, menyelesaikan masalah sintesis dan koreksi untuk potongan-potongan DNA lalu menyatukan potongan-potongan DNA secara rekursif. Penerapan algoritma Divide and Conquer diharapkan dapat mengurangi kompleksitas proses sintesis dan koreksi DNA

Kata kunci: sintesis DNA, koreksi DNA, algoritma Divide and Conquer, bioinformatika

1 PENDAHULUAN

DNA merupakan komponen yang berperan sebagai cetak biru kehidupan. Sebagai komponen yang penting, molekul-molekul DNA yang panjang sangat dibutuhkan dalam bidang penelitian. Namun, mesin hanya dapat melakukan sintesis *oligonukleotida* yang pendek secara cepat. Sementara itu, melakukan sintesis DNA yang sekaligus panjang merupakan hal yang kompleks.

Sintesis DNA akan bermafaat bila menghasilkan DNA yang bebas dari kesalahan (*error-free*). Namun, saat ini *oligonukleotida* sintetik masih rawan kesalahan (paling tidak satu kesalahan per 160 nt). Resiko timbulnya kesalahan dalam sintesis meningkat secara eksponensial terhadap panjang DNA. Akibatnya, proses koreksi terhadap kesalahan DNA juga meningkat secara eksponensial terhadap panjang DNA.

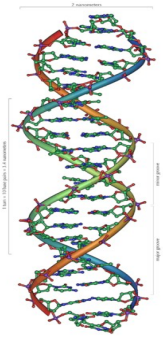
Sintesis dan koreksi DNA yang panjang sekaligus merupakan suatu metode yang kurang mangkus. Permasalahan sintesis dan koreksi DNA yang panjang sebenarnya dapat direduksi dengan membuat partisi pada DNA. Oleh karena itu, algoritma *Divide and Conquer* dapat diterapkan untuk menghasilkan metode sintesis dan koreksi DNA yang lebih mangkus.

2 PENERAPAN ALGORITMA DIVIDE AND CONQUER DALAM SINTESIS DNA

2.1 Pengertian DNA

Deoxyribunucleic acid (DNA) atau Asam Deoksiribo Nukleat (ADN) merupakan asam nukleat yang mengandung instruksi genetik. Instruksi genetik tersebut digunakan dalam sintesis protein dari semua makhluk hidup dan beberapa jenis virus. Fungsi utama DNA adalah untuk menyimpan informasi jangka panjang

DNA, secara kimia, merupakan polimer panjang yang terbentuk dari unit sederhana yang disebut nukleotida. Nukleotida merupakan molekul yang terbentuk dari gula dan fosfat yang menyatu karena adanya ikatan ester.



Gambar 1. DNA

2.2 Definisi Istilah

Polimer = rangkaian atom yang panjang dan berulang-ulang dan dihasilkan dari sambungan beberapa molekul lain yang dinamakan monomer

Nukleotida = molekul yang tersusun dari gugus basa heterosiklik, gula, dan satu atau lebih gugus fosfat.

Oligonukleotida = (umumnya disingkat "*oligo*") merupakan seberkas pendek polimer (DNA atau RNA) yang sering digunakan sebagai primer atau sebagai penuntun probe pada berbagai teknik analisis deteksi dalam biologi molekular.

PCR (polymerase chain reaction) = perbanyakan DNA in vitro, hanya di tabung reaksi, tidak di dalam sel

ssDNA (single-stranded DNA) = DNA yang hanya memiliki satu basa pada setiap anak tangganya

dsDNA (double-stranded DNA) = DNA yang hanya memiliki dua pasangan basa pada setiap anak tangganya

in vitro = reaksi dalam tabung

in silico = reaksi di dalam komputer atau simulasi reaksi

GFP (green fluorescent protein) = protein, yang terdiri atas 238 asam amino. Gen GFP sering digunakan sebagai reporter dalam biologi molekuler

Hibridisasi = proses yang menghasilkan sebuah *dsDNA* dari dua buah *ssDNA* yang komplementer melalui pembentukan ikatan hidrogen antara dua basa.

Primer = oligonukleotida pendek yang menyediakan 3 gugus hidroksida dalam PCR

2.3 Algoritma Divide and Conquer

Divide and Conquer adalah metode pemecahan masalah yang bekerja dengan membagi masalah (problem) menjadi beberapa upamasalah (subproblem) yang lebih kecil, kemudian menyelesaikan masing-masing upamasalah secara independen, dan akhirnya menggabung solusi masing-masing upamasalah sehingga menjadi solusi masalah semula. Metode Divide and Conquer lebih natural bila diungkapkan dengan skema rekursif. [1]

2.4 Algoritma Divide and Conquer dalam Sintesis DNA

Telah diketahui bahwa melakukan sintesis DNA yang panjang merupakan hal yang kompleks, sementara melakukan sintesis *oligonukleotida* yang pendek dapat dilakukan dengan lebih cepat. Algoritma Divide and Conquer dapat diterapkan untuk menyederhanakan sintesis DNA dengan tiga proses utama berikut

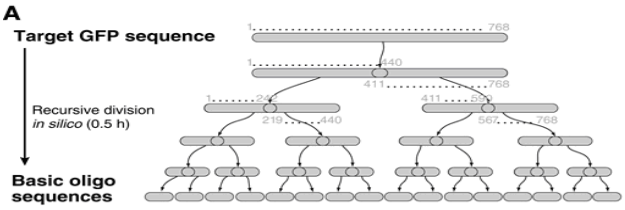
Divide: membagi DNA template yang ingin diduplikasi menjadi molekul ssDNA template yang lebih pendek secara rekursif.

Conquer: bila ssDNA yang dihasilkan sudah cukup pendek, melakukan sintesis *oligonukleotida* sesuai dengan ssDNA template

Combine: menggabungkan *oligonukleotida* hasil sintesis menjadi ssDNA baru secara rekursif hingga membentuk DNA baru yang utuh

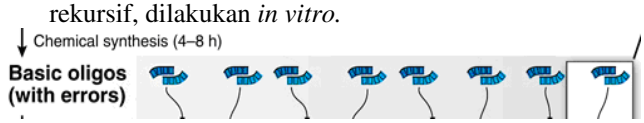
Algoritma tersebut dapat diilustrasikan dalam gambar dan penjelasan berikut.

1. Rentetan target GFP DNA dibagi secara rekursif secara *in silico* menjadi *oligonukleotida* yang saling overlapping. (16 oligo per rata-rata ukuran 75 bp dalam sintesis *GFP*).



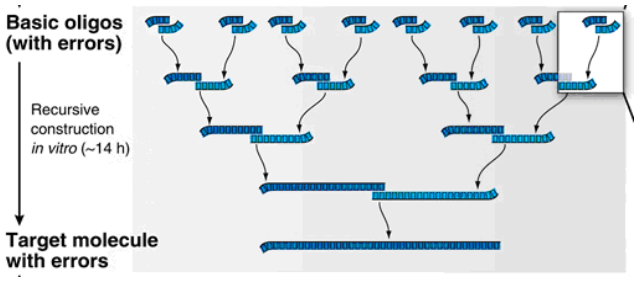
Gambar 2. Pembagian target GFP secara rekursif

2. *Oligo* spesifik disintesis secara konvensional dan digunakan sebagai input (warna biru) untuk konstruksi rekursif, dilakukan *in vitro*.



Gambar 3. Oligo hasil sintetik

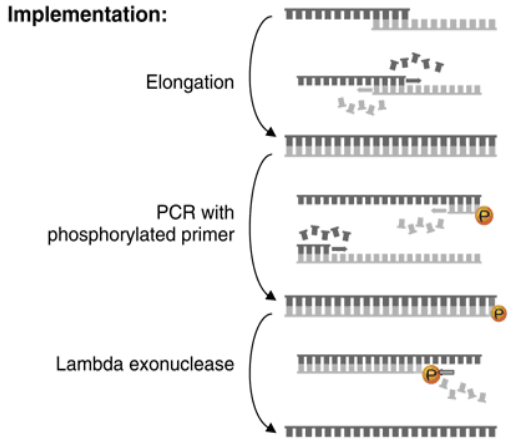
3. Proses konstruksi dengan menggabungkan pasangan *ssDNA* (*oligo*) yang saling overlapping menjadi *ssDNA* yang lebih panjang sehingga target molekul terbentuk.



Gambar 4. Konstruksi *ssDNA*

Proses penggabungan tiap dua pasangan *ssDNA* terjadi dalam beberapa tahap berikut

a. Elongasi : molekul *ssDNA* saling menghibridasi dan saling melengkapi untuk membentuk sebuah molekul *dsDNA*.
b. Molekul *dsDNA* tersebut dikuatkan oleh PCR dengan primer phosphorilated
c. Lambda exonuclease : *dsDNA* yang telah dikuatkan oleh PCR tersebut didegradasi menjadi molekul *ssDNA*



Gambar 5. Penggabungan pasangan *ssDNA*

2.5 Pseudo-Code Algoritma Divide and Conquer dalam Sintesis DNA

Berikut ini akan dituliskan pseudo-code yang disederhanakan untuk menghasilkan aturan sintesis DNA (berupa titik mana saja yang perlu untuk dipotong). Masukan pseudo-code berupa target DNA (dapat berupa DNA template) yang diinginkan dan batasan-batasan (dapat berupa jumlah dan jenis pereaksi, suhu reaksi).

```
procedure DC_DNA (input tDNA : DNA target,lb :
list of batasan, j_cache : integer, output
at_terbaik : aturan, current_best_cost :
integer)
{Mencari aturan optimal dalam sintesis DNA
Masukan : target DNA
Keluaran : aturan optimal dalam sintesis DNA
dan harga terbaik dari aturan tersebut}
Deklarasi :
Cache : list of aturan
pjg_min_oligo : 30
pjg_max_oligo : 80
current_at : aturan
current_cost : integer
cek1,cek2,cek3,cek4=boolean
stop : boolean
k : integer
l_oli : list of oligo

boolean Cek_overlap (input tb :
titik_bagi_DNA, lb : list of batasan,l_oli:
list of oligo)
{melakukan pemeriksaan apakah ada oligo yang
overlap dan memenuhi batasan di titik
tersebut, bila ada → mengembalikan true}


boolean Cek_primer (input tb : titik_bagi_DNA,
lb : list of batasan,l_oli : list of oligo)
{melakukan pemeriksaan apakah ada primer dalam
oligo yang memenuhi batasan di titik tersebut,
bila ada → mengembalikan true }

integer level_cost(input at : aturan)
{menghitung harga dari sebuah aturan, selain
harga untuk melakukan reaksi}


integer reaksi_cost(input at : aturan)
{menghitung harga reaksi dari sebuah aturan}

Algoritma :
stop = false
for (j=0 to j_cache) do
        if (tDNA ada di Cache) then
          at_terbaik ← Cache[j]
          stop ← true
```

```
if (stop=false)

      if ((tDNA>pjg_min_oligo)and
      (tDNA<pjg_max_oligo)) then
          current_at ← oligo
          l_oli[k]← oligo

      else
       current_at ← null
       current_best_cost ← 999999
       cek1←Cek_overlap(tengah(t_DNA),lb,l_oli)
       cek2←Cek_primer(tengah(t_DNA),lb,l_oli)
       cek3←Cek_primer(sub_tDNA_kiri,lb,l_oli)
       cek4←Cek_primer(sub_tDNA_kanan,lb,l_oli)

       if(not(cek1 and cek2 and cek3 and cek4))
       then
            DC_DNA(sub_tDNA_kiri,lb,i,
            at_best_kiri,kiri_cost)

            DC_DNA(sub_tDNA_kanan,lb,i,
            at_best_kanan,kanan_cost)

            current_at←Merge(at_best_kiri,
            at_best_kanan)

            current_cost←kiri_cost+kanan_cost
            +level_cost(current_at)
            +reaksi_cost(current_at)

            if (current_cost<current_best_cost)
            then
              current_best_cost← current_cost
              at_terbaik ← current_at
              cache[i] ← at_terbaik
              j_cache ← j_cache + 1
```

Tanpa memperhitungkan rekursif, kompleksitas waktu algoritma DC_DNA tersebut dapat diuraikan sebagai berikut

1. Pemeriksaan ada atau tidaknya tDNA dalam Cache = mb, dengan b = sebuah konstanta dan m = j_cache
2. Perbandingan panjang tDNA dengan panjang oligo=c, dengan c adalah sebuah konstanta
3. Pemeriksaan dengan sebuah fungsi Cek_overlap dan 3 buah fungsi Cek_primer = d + 3e, dengan d = konstanta harga untuk fungsi Cek_overlap, dan e = konstanta harga untuk fungsi Cek_primer
4. Penghitungan current_cost = f, dengan f adalah sebuah konstanta
5. Perbandingan current_cost dengan current_best_cost = gh, dengan g adalah sebuah konstanta

Sedangkan dalam relasi rekurens, kompleksitas total dapat dituliskan menjadi :

$$T(n)\begin{cases} nb+c+d+3e+f+g, & n=1 \\ 2T(n/2)+hn, & n>1 \end{cases}$$

dengan hn adalah waktu yang dibutuhkan oleh fungsi Merge.

Penyelesaian persamaan rekurens :

$T(n) = 2T(n/2) + hn$

$= 2(2T(n/4) + hn/2) + hn)$

$= 4T(n/4) + 2hn$

$= ...$

$= 2^k\ T(n/2^k+khn)$

Persamaan terakhir dapat diselesaikan karena basis rekursif yaitu n = 1

$n/2^k = 1 \rightarrow k = {}^2\log n$

sehingga

$T(n) = nT(1) + hn\ {}^2\log n$

$= n(mb+c+d+3e+f+g) + hn\ {}^2\log n$

untuk kasus rata-rata, dapat diambil nilai m = n/2

sehingga

$T(n) = n((n/2)b+c+d+3e+f+g) + hn\ {}^2\log n$

$= O(n^2 + n\ {}^2\log n) = O(n^2)$

2.6 Algoritma Divide and Conquer dalam Koreksi DNA

Hasil sintesis DNA sering memiliki kesalahan. Oleh karena itu, dibutuhkan tahapan lanjutan berupa koreksi DNA.

Umumnya, kesalahan terdistribusi secara acak pada komponen DNA dan terjadi secara acak pada proses konstruksi DNA. Oleh karena itu, kloning terhadap DNA diharapkan dapat menghasilkan beberapa bagian DNA yang bebas kesalahan. Bila bagian DNA yang bebas kesalahan (pada setiap klone) dapat dideteksi, bagian tersebut dapat dipotong dan disatukan dengan bagian yang lain (dari klone yang sama atau berbeda).

Untuk mengoptimalkan pemotongan dan penggabungan DNA yang bebas kesalahan, algoritma Divide and Conquer dapat diterapkan sebagai berikut.

Pra-prosedur : mengilustrasikan target DNA sebagai akar dari sebuah pohon dan DNA hasil kloning menjadi anak-anak dari akar tersebut. Lakukan iterasi untuk

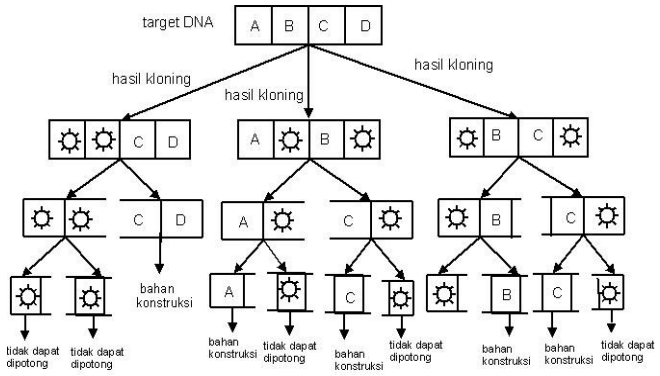
menemukan kesalahan pada DNA setiap anak tersebut. Bila terdapat kesalahan, lakukan proses Divide

Divide : lakukan pemotongan DNA menjadi 2 bagian yang saling overlapping

Conquer: bagian DNA anak yang tidak memiliki kesalahan diambil sebagai *oligonukleotida* untuk bahan rekonstruksi. Bagian yang masih memilki kesalahan dijadikan anak baru dan menjadi input proses Divide.
Proses Divide dan Conquer selesai bila anak yang terbentuk tidak dapat dipotong lagi

Combine: menggabungkan bahan-bahan konstruksi sesuai dengan prioritas bahan (semakin dekat suatu bahan dengan akar, prioritas semakin tinggi)

Contoh pohon yang dihasilkan oleh algoritma Divide and Conquer sebelum proses Combine digambarkan sebagai berikut. Kesalahan ditandai dengan lambang matahari.



Gambar 6. Pohon Koreksi Kesalahan

Oligonukleotida bahan-bahan konstruksi yang dihasilkan dari pohon tersebut :

Tabel 1. *Oligo* dari Pohon Koreksi Kesalahan

No.	Oligo	Jarak ke akar	Prioritas
1	C,D	2 sisi	2
2	A	3 sisi	3
3	C	3 sisi	3
4	B	3 sisi	3
5	C	3 sisi	3

Karena *oligo* pada nomor 3 dan 5 telah terdapat pada *oligo* nomor 1 yang memiliki prioritas lebih tinggi, maka *oligo* nomor 3 dan 5 tidak digunakan.

2.7 Pseudo-code Algoritma Divide and Conquer dalam Koreksi DNA

```
procedure koreksiDNA (input tDNA : DNA(target),
klone: list of DNA(hasil kloning), m : integer,
output efDNA : DNA)
{Menghasilkan DNA bebas kesalahan
Masukan berupa DNA target, DNA hasil kloning, dan
integer jumlah kloning
Keluaran berupa DNA bebas kesalahan sesuai target
DNA}

Deklarasi :
i : integer
error : boolean
l : integer
temp : DNA {DNA sementara}

procedure DC_korek (input level:integer, pDNA :
DNA, output part : DNA)

boolean Cek_error (input DNA1 : DNA)
{menghasilkan true bila DNA yang diperiksa
memiliki kesalahan}

DNA Combine(input DNAa, DNAb : DNA)
{menghsilkan DNA hasil gabungan DNAa dan DNAb
dengan prioritas, setiap DNA memiliki atribut
prioritas berupa level}

Algoritma :
Error← true
i ← 1
stop ← false
while ((i<=m) and (not stop))do
    if((Error and Cek_error(klone[i]))=false)
    then {bila terdapat kloning yang tidak
                memiliki kesalahan}
        stop ← true
        efDNA ← klone[i]
    else
        i ← i+1
if (i>m) {semua kloning memiliki kesalahan}
then
    l ← 1
    for (i = 1 to m) do
        DC_korek(l,klone[i],part1)
        temp←Combine(temp,part1)
    efDNA← temp


procedure DC_korek (input level:integer, pDNA :
DNA, output part : DNA)

Deklarasi :
```

```
Algoritma :
if (Cek_error(pDNA)=false)
then
    part ← pDNA
    part.prio←1
else
    if (pDNA.panjang>1) then
        DC_korek(l+1,pDNA_kiri,part_ki)
        DC_korek(l+1,pDNA_kanan,part_ka)
        part←Combine(part_ka,part_ki)
```

Kompleksitas algoritma tersebut dapat dihitung sebagai berikut :

1. Pemeriksaan ada atau tidaknya kesalahan pada setiap klone,
   a. untuk kasus terbaik (ditemukan klone tanpa kesalahan) dengan a suatu konstanta
   b. kasus terburuk (semua klone memiliki kesalahan) = ma dengan m adalah jumlah klone
2. Kompleksitas prosedur DC_korek dalam relasi rekurens

$$T(n)\begin{cases} a, n=1 \\ 2T(n/2)+cn, n>1 \end{cases}$$

dengan a adalah konstanta untuk prosedur Cek_eror, c adalah konstanta untuk prosedur Combine, dan n adalah konstanta yang maksimal setara dengan panjang DNA.

Penyelesaian persamaan rekurens :

$T(n) = 2T(n/2) + cn$
$= 2(2T(n/4) + cn/2) + hn)$
$= 4T(n/4) + 2cn$
$= ...$
$=2^kT(n/2^k+kcn)$

Persamaan terakhir dapat diselesaikan karena basis rekursif yaitu n = 1

$n/2^k = 1 \rightarrow k = {}^2\log n$

sehingga

$T(n) = nT(1) + cn\ {}^2\log n$
$= na + cn\ {}^2\log n$

3. Karena prosedur DC_korek dan prosedur Combine dilakukan sebanyak m kali dalam algoritma utama (m= jumlah kloning), maka

$T(n) = m(na + cn\ {}^2\log n)$

Untuk kasus terburuk, jumlah kloning (m) = panjang DNA (n), sehingga

$T(n) = n(na + cn\ {}^2\log n)$
$= O(n^2 + n\ {}^2\log n) = O(n^2)$

3 KESIMPULAN

1. Algoritma Divide and Conquer memungkinkan sintesis pembentukan DNA yang panjang dengan membentuk dan menyusun *oligo-oligo* yang pendek
2. Algoritma Divide and Conquer mengurangi kompleksitas koreksi DNA dari eksponensial menjadi polinomial : $O(n^2)$
3. Untuk mengoptimalkan algoritma Divide and Conquer dalam sintesis dan koreksi DNA, dibutuhkan penerapan algoritma lain misalnya algoritma Branch and Bound atau algoritma Pemrograman Dinamis.
4. Algoritma Branch and Bound atau algoritma Pemrograman Dinamis pada sintesis DNA diharapkan dapat menghitung cost untuk setiap reaksi kimia yang terjadi dengan lebih akurat. Cost reaksi kimia yang akurat dapat menyebabkan pembagian DNA tidak selalu dilakukan di tengah-temgah.
5. Algoritma Branch and Bound atau algoritma Pemrograman Dinamis pada koreksi DNA diharapkan dapat mencegah duplikasi *oligo*. Pencegahan duplikasi *oligo* berarti *oligo* dengan prioritas lebih rendah tidak dihasilkan bila telah terdapat *oligo* yang sama dengan prioritas yang lebih tinggi.

REFERENSI

1. Linshiz, Gregory, dkk, ”Recursive Construction oOf Perfect DNA Molecules from Imperfect Oligonucleotides”, <http://www.nature.com/msb/journal/v4/n1/full/msb200826.html>, diakses tanggal 17 Mei 2008 pukul 16.29
2. MSN Encarta, “Deoxyribonucleic Acid” <http://encarta.msn.com/encyclopedia_761561874/deoxyribonucleic_acid.html>, diakses tanggal 17 Mei 2008 pukul 16.31
3. Retnoningrum, Debi Sofie, dkk, “Diktat Kuliah Bioteknologi Farmasi (FA-3221)”, Sekolah Farmasi ITB, 2007
4. [1] Munir, Rinaldi, “Diktat Kuliah IF2251 Strategi Algoritmik” , Program Studi Teknik Informatika ITB, 2007
5. Wikipedia,“DNA”,<http://en.wikipedia.org/wiki/DNA>, diakses tanggal 17 Mei 2008 pukul 16.34