Silesian University of Technology
Macro Faculty

Introduction to Bioinformatics

Laboratory

Genomic and protein databases

Piotr Wittchen

Specialization: Informatics
Contact e-mail: piotr.wittchen@gmail.com

Gliwice 2010

Description of the algorithm “Greedy profile motif search”

Due to better understanding of the algorithm, I analyzed instruction and presentation. Afterwards, I created more detailed description of the algorithm, which is as follows:

Assumption: I am operating on the following set of letters:

{'A','R','N','D','C','E','Q','G','H','I','L','K','M','F','P','S','T','W','Y','V','X'}

1. I am choosing from the file one great sequence of letters.
2. From the sequence obtained in the previous point I am choosing *M* random sequences of the length *N*.
3. I am randomly assigning starting position for the subsequences of the sequences obtained in the previous point. Each subsequence must have the same length *L*.
4. I am removing one random sequence from the set of sequences.
5. I am creating the profile from the remaining sequences.
6. I am creating consensus string basing on the created profile.
7. I am calculating probability *prob(a|P)* for each subsequence of length *L* from the sequence removed in the point 4.
8. I am dividing probabilities from the previous point by the sum of all probabilities calculated in the previous point.
9. I am setting the new starting point for the subsequence in the removed sequence. Starting point is determined by the highest probability calculated in the previous point.
10. I am calculating the product of the all non-zero probabilities calculated in the point 8. and marking this product as *motifWeight*.
11. I am repeating steps 4-10 up to the moment when *motifWeight* in the current iteration will be lower than *motifWeight* calculated in the previous iteration **or** I am repeating steps 4-10 particular number of times (stop condition).

Implementation of the algorithm

I have decided to use PHP in order to write this algorithm because of the easier way of implementation that e.g. in C++ language.

I wrote two scripts. First (*prepare.php*) is responsible for extracting single protein from the flat file and saving chosen sequence to the another file (*mito_prepared.aa*). Another one (*gapd.php*) is responsible for performing the algorithm of the g*reedy profile motif search*. Both scripts can be run from the console in the Windows or Linux OS or directly from the web-browser on the system with PHP interpreter installed.

Source code of the scripts is placed below. It has quite unfriendly form, so I attached files with the source code to this report.

Source code – prepare.php

```
<?php

set_time_limit(0);

$lines = file('mito.aa');
$start = rand(0, count($lines));
$i = $start;
$loop = TRUE;
$output = array();
$outputString = '';
$beginString = FALSE;
$endString = FALSE;

while($loop)
{
	if($beginString && !$endString)
	{
		$output[] .= $lines[$i];
	}

	if($lines[$i]{0} == '>' && !$beginString && !$endString)
	{
		$name = $lines[$i];
		$beginString = TRUE;
	}
	elseif($lines[$i]{0} == '>' && $beginString && !$endString)
	{
		$endString = TRUE;
		$loop = FALSE;
	}

	$i++;
}

unset($output[count($output) - 1]);
unset($lines);

foreach($output as $value)
	$outputString .= $value;

$outputString = str_replace("\n","",$outputString);

echo $name;

file_put_contents('mito_prepared.aa',$outputString);

?>
```

Source code – gapd.php

```
<?php

set_time_limit(0);

echo "<pre>\n"; // tag for preformated text (in case of running script in a web
browser)

// reading and preparing the data

$alphabet =
array('A','R','N','D','C','E','Q','G','H','I','L','K','M','F','P','S','T','W','
Y','V','X');
$countAlphabetElements = count($alphabet);
$initialSequenceLines = file('mito_prepared.aa');
$initialSequenceString = '';
$initialSequenceArray = array();
$newStartingPoint = NULL;

foreach($initialSequenceLines as $line)
	$initialSequenceString .= str_replace("\r\n","",$line);

// array of letters
$initialSequenceArray = str_split($initialSequenceString);

unset($initialSequenceLines);
unset($initialSequenceString);

$numberOfSequences = 50;		// M - number of Sequences
$lengthOfSequence = 200;		// N - length of a single sequence
$lengthOfSubsequence = 100;	// L - length of subsequence
$iterations = 1000;			// iterations of an algorithm

$sequenceUpperBound = count($initialSequenceArray) - $lengthOfSequence;
$sequenceArray = array();

// creating random sequences

for($i = 0; $i < $numberOfSequences; $i++)
	$sequenceArray[$i] = array_slice($initialSequenceArray, rand(0,
$sequenceUpperBound), $lengthOfSequence);

unset($initialSequenceArray);

// creating subsequences (with random starting points)

$subSequenceUpperBound = $lengthOfSequence - $lengthOfSubsequence;
$subSequenceArray = array();

for($i = 0; $i < $numberOfSequences; $i++)
	$subSequenceArray[$i] = array_slice($sequenceArray[$i], rand(0,
$subSequenceUpperBound), $lengthOfSubsequence);

// point no. 4 - beginning of the loop

for($loop = 0; $loop < $iterations; $loop++)
{
	if($newStartingPoint != NULL)
		$subSequenceArray[$indexOfSubSequenceToDelete] =
array_slice($sequenceArray[$indexOfSubSequenceToDelete], $newStartingPoint,
$lengthOfSubsequence);
```

```
	// choosing random sequence to delete

	$indexOfSubSequenceToDelete = rand(0, ($numberOfSequences - 1) );

	// creating the profile

	// preparing the data

	$profile = array();

	for($i = 0; $i < $lengthOfSubsequence; $i++)
		$profile[] = array_fill(0, ($countAlphabetElements - 1), 0);

	// calculating occurence of letters on each postion of each subSequence

	foreach($subSequenceArray as $subSequenceNumber => $subSequence)
	{
		if($subSequenceNumber != $indexOfSubSequenceToDelete) //
neglecting removed sequence
		{
			foreach($subSequence as $letterPosition => $letter)

	$profile[$letterPosition][array_search($letter,$alphabet)]++;
		}
	}

	// calculating probability of letter occurrence - prob(a|P)

	foreach($profile as $aKey => $aValue)
	{
		foreach($aValue as $aLetterKey => $aLetterCount)
			$profile[$aKey][$aLetterKey] =
$aLetterCount/$countAlphabetElements;
	}

	// generating the consensus string

	$consensusString = '';

	foreach($profile as $aKey => $aValue)
		$consensusString .= $alphabet[array_search(max($aValue),
$aValue)];

	// generating slices of the removed sequence

	$fragmentsOfRemovedSequence = array();

	for($i = 0; $i <= $subSequenceUpperBound; $i++)
		$fragmentsOfRemovedSequence[] =
array_slice($sequenceArray[$indexOfSubSequenceToDelete], $i,
$lengthOfSubsequence);

	// calculating probability prob(a|P)

	$probs = array();

	foreach($fragmentsOfRemovedSequence as $kFrag => $vFrag)
	{
		$probs[$kFrag] = 1;
		foreach($vFrag as $kLetter => $vLetter)
			$probs[$kFrag] *= $profile[$kLetter][array_search($vLetter,
$alphabet)];
```

```
	}

	$motifWeight = max($probs);

	if($motifWeight != 0)
		$newStartingPoint = array_search($motifWeight, $probs);
	else
		$newStartingPoint = NULL; // avoidance of generating worse
motifWeight

	echo("Iteration: ".$loop."\n");
	echo("Motif weight: ".$motifWeight."\n");
	echo("Consensus string: ".$consensusString."\n\n");
}

echo "\n</pre>";

?>
```

Tests

I made several tests of this scripts. Below, I am presenting one of them.

First script chose the following sequence form the *mito.aa* file:

```
>gi|77020002|ref|YP_337891.1| NADH dehydrogenase subunit 2 [Aspergillus niger]

MLLSSIFCLLLSNALSFRRDTAILYSRIGIIVLFYCIYLAYNNLFLTYLDNGIGLFGGLFYTSSITQIFHILIFLVSLLI
LNMTGFYPRKLISSEYMSLHKLIFTKLSFVKNLTVSNIILKKGEQYTIIEYTLMLLFIITGSVLLISSSDLVSIFLSIEL
QSYGLYLLCAMYRNSESSTSASLTYFLLGGLSSCFILLGIGLIYANLGVTYLDSFYVINNLAGIINNQEITTYIPYCLLL
ISVGFLFKISAAPFHFWSPDVYDGIPTIVTTFVAIIAKISILTLLLQLVHYTNSIYITTSYSWTTSLLVSSLLSLIIGTV
LGLTQFRIKRLFAYSTISHLGFMLLALTINSVESIQSFIFYLIQYSLSNLNAFILLVAIGYSLYAYNDKNINHNNLIDKN
NSPIQLISQLKGYFHINSILALSLSITLFSFAGIPPLMGFFAKQMVFSAALPEGFIFLLLIGVLTSVISAVYYLFIVKTM
FFDGHTYTSFNKLKDLKIPALVLQKDKVINKIYFDSKFALSSSLSITISILTLIILLFMFMPNELLHISNLLSIILFIPN
SI
```

Second script performed the algorithm on the sequence above
with the following parameters:

```
Number of sequences:       50
Length of sequence:        200
Length of sub-sequence:    100
Iterations:                1000
```

Results of the tests

```
Iteration: 0
Motif weight: 5.7905373268191E-60
Consensus string:
LSLLLLLLLLLLIELILIGLLLLTLLSLLSILISLSLLLLLIIIILLLLIILLILSLLSALLISLLIIILL
SIFLLILILFSLIILIIILGLLLLISLIL

...

Iteration: 24
Motif weight: 5.2254617410283E-33
Consensus string:
LILTNTILGLGLQLLILLGILLITFQIFVTYLLSVSSLLLLIFIINVLEIYSLILYCLLYINISFLIFILL
APFLLISLSKGYFIIIIILTLVLLIALFS

Iteration: 25
Motif weight: 8.8669103114113E-32
Consensus string:
LIAYSTILGLGLSLLILLIIILISIQSFVTYLLSVSLLLLAFIINVLIIYSLIAYCLLYINISNLIFILL
APFLLISLLKGYFIINIILTLVLSITLFS

...

Iteration: 998
Motif weight: 1.8807534945082E-24
Consensus string:
ASLTYTILGLGSSLFILLGIGLIYAQLFVTYLDSFSLINLNAFIINNQEIYSLIPYCLLLINVGFLFKISN
APFHLISPDKGDFIPTIILTLVLIIAKFS

Iteration: 999
Motif weight: 2.9167079672681E-10
Consensus string:
ASLTYTILGLGSSLFILLGIGLIYAQLFVTYLDSFSLLNLNAGIINNQEITSLIPYCLLLINVGFLFKISN
APFHLISPDKGDFIPTIILTLVLIIAKFS
```

Note: Whole result is included in the file: *output.txt* attached to this report.

Conclusions

I observed, that sometimes all values of *prob(a|P)* were equal to zero. In such case, I did not choose the new starting point for the sub-sequence, because it caused worse motif weight generation in the successive iterations. In these situations, I left the old starting point. In the example presented above, we can see, that motif weight becomes greater in the successive iterations and quality of the consensus string is improved. Difference of the motif weight can be clearly seen, if we set large number of sequences. When, I set e.g. “Number of Sequences” = 5 in the earlier tests, the difference was not as great as in the example from this report. Regardless this fact, quality of the chosen protein was improved by the script.