

# Algorithms in Bioinformatics I, WS2002/3

## Assignment sheet # 5

Daniel Huson

November 11, 2002

In the lecture we discussed how a multiple alignment problem can be formulated as an integer linear program ILP. The aim of this exercise sheet is to implement this approach in the case of two sequences. First, we need a program that takes as input two sequences, a match and mismatch score and formulates the ILP (Problem 1). Secondly, we need to run a program that solves the ILP (Problem 2). We will use the free software `lp_solve` to attempt to solve the ILP. As discussed in the lecture, ILPs are hard to solve and `lp_solve` will only handle simple cases. Once the ILP has been solved, we need a third program that captures the solution of the ILP, extracts the trace from it and prints out the corresponding alignment (Problem 3).

All three programs should be run on the following data sets:

1. `ex1.fa ex2.fa` with match score=3 and mismatch score= 1,
2. `ex1.fa ex2.fa` with match score=1 and mismatch score= -1,
3. `in1.fa ex2.fa` with match score=3 and mismatch score= 1,
4. `in1.fa ex2.fa` with match score=1 and mismatch score= -1,
5. `big1.fa big2.fa` with match score=1 and mismatch score= -1, and
6. `big1.fa big2.fa` with match score=5 and mismatch score= 1, if you have the time to wait...

### 1 Setting up an ILP for a pairwise alignment (3 points)

Write a program that takes as input two sequences  $x$  and  $y$ , a match score  $p$  and a mismatch score  $q$ . Output is an ILP which can be read by the program `lp_solve`.

To do this, modify `Program6.java`, which is available from:

[www-ab.informatik.uni-tuebingen.de/teaching/ws02/abi1/programs/program06.zip](http://www-ab.informatik.uni-tuebingen.de/teaching/ws02/abi1/programs/program06.zip). This download also contains data files to run the algorithms on. Additionally, this contains two executable versions of `lp_solve`, one for linux and one for solaris, in directories `linux` and `solaris`, respectively. (If you are interested, the complete sources and documentation can be downloaded from [ftp://ftp.es.ele.tue.nl/pub/lp\\_solve](ftp://ftp.es.ele.tue.nl/pub/lp_solve))

Command-line options of the program should be the names of two files containing the two sequences (in fasta format, as usual) and two numbers  $s$  and  $m$  with the following interpretations:

- $s$  is the score for a match, and
- $m$  is the score for a mismatch.

For example, for input files `ex1.fa` and `ex2.fa`, with match- and mismatch scores 3 and 1, respectively, run the program like this:

```
java Program6 ex1.fa ex2.fa 3 1 >ex1_ex2_3_1.ilp
```

using the unix “>” command to pipe the output to a file.

## 2 Running `lp_solve` to solve the ILP (2 points)

For each of the data sets, run the program `lp_solve` on the output of `Program6` to produce the input for `Program7`. Continuing the above example, run `lp_solve` like this:

```
./solaris/lp_solve <ex1_ex2_3_1.ilp >ex1_ex2_3_1.crunch
```

This produces a *crunch* file, i.e. a file produced by number crunching.

## 3 Producing an alignment from the ILP solution (4 points)

Write a program that takes as input two sequences in fasta format and a corresponding crunch file. The program should report the optimal score obtained for the ILP, which is contained in the crunch file.

To do so, please modify `Program7` in the above mentioned download.

More importantly, the program should capture the values of the different edge variables, thus obtaining a trace for the optimal alignment.

This trace should then be converted into an actual alignment and the alignment is then printed out.

Please first write down an algorithm for converting a trace into an alignment. Then try to implement it! Continuing the above example, this program should be launched like this:

```
java Program7 ex1.fa ex2.fa ex1_ex2_3_1.crunch
```

This should produce the following output:

```
Program7: produces an alignment from the solution of an ILP
Read ex1.fa: 5, ex2.fa: 5, ex1_ex2_3_1.crunch: 25
Optimal score: 12
Alignment:
-AGGTT
AAGGT-
```

## Note

For debugging purposes, the download contains the following files:

- `ex1.fa` and `ex2.fa` - two example input files,
- `ex1_ex2_3_1.ilp` - the ILP file produced by `Program6`,
- `ex1_ex2_3_1.crunch` - the solution to the ILP produced by `lp_solve`, and
- `ex1_ex2_3_1.align` - the alignment produced by `Program7`.

A clean run of the examples looks something like this:

```
[huson@tolaga]$ java Program6 ex1.fa ex2.fa 3 1 >ex1_ex2_3_1.ilp
Program6: generates an ILP for the pairwise alignment problem
Read ex1.fa: 5, ex2.fa: 5
Match score: 3, mismatch score: 1

[huson@tolaga]$ ./solaris/lp_solve <ex1_ex2_3_1.ilp >ex1_ex2_3_1.crunch

[huson@tolaga]$ java Program7 ex1.fa ex2.fa ex1_ex2_3_1.crunch
Program7: produces an alignment from the solution of an ILP
Read ex1.fa: 5, ex2.fa: 5, ex1_ex2_3_1.crunch: 25
Optimal score: 12
Alignment:
-AGGTT
AAGG-T
```

The template versions of `Program6.java` and `Program7.java` contain code for doing input and output and contain hints on how to structure your code. Additionally, `Program6.java` contains a method `String nodesToVariable(int i,int j)` that takes two positions  $i$  and  $j$  in the two sequences  $x$  and  $y$ , respectively, and returns the name of the corresponding variable to be used in the ILP. Conversely, `Program7.java` contains two methods `int xNode (String name)` and `int yNode (String name)` that takes the name of a variable read from the crunch file produced by `lp_solve` and returns the position in the  $x$  sequence, or  $y$  sequence, respectively.

## Additional useful activities

How could one speed up the computation? Instead of considering the complete alignment graph, one could consider subgraphs. For example, we could restrict the number of positions that we want a base in  $x$  to be able to move in  $y$ . Experiment with this.

Remove the last line in the input file for `lp_solve`. This statement starting with `int` tells `lp_solve` that the variables contained in the statement must be assigned integer values. Without this statement the program runs much, much faster. But is the result obtained for the LP-relaxation any use?

The slowness of a generic ILP solver and the uselessness of the LP-relaxation are the two reasons why one needs to consider elaborate branch-and-cut algorithms to solve alignment using this approach.

**Due by 10am, Monday, 18 Nov 2002**