public int pairwiseAlignment(SymbolList query, SymbolList subject)

throws BioRuntimeException {

Sequence squery = null;

Sequence ssubject = null;

if(query instanceof Sequence){

squery = (Sequence)query;

}else{

//make it a sequence

squery = new SimpleSequence(query, "", "query", new SimpleAnnotation());

}

if(subject instanceof Sequence){

ssubject = (Sequence)subject;

}else{

//make it a sequence

ssubject = new SimpleSequence(subject, "", "subject", new SimpleAnnotation());

}

if (squery.getAlphabet().equals(ssubject.getAlphabet())

&& squery.getAlphabet().equals(subMatrix.getAlphabet())) {

long time = System.currentTimeMillis();

int i, j;

this.CostMatrix = new int[squery.length() + 1][ssubject.length() + 1]; // Matrix

// CostMatrix

/\*

\* Variables for the traceback

\*/

String[] align = new String[] {"", ""};

String path = "";

// construct the matrix:

CostMatrix[0][0] = 0;

/\*

\* If we want to have affine gap penalties, we have to initialise

\* additional matrices: If this is not necessary, we won't do that

\* (because it's expensive).

\*/

if ((gapExt != delete) || (gapExt != insert)) {

int[][] E = new int[squery.length() + 1][ssubject.length() + 1]; // Inserts

int[][] F = new int[squery.length() + 1][ssubject.length() + 1]; // Deletes

E[0][0] = F[0][0] = Integer.MAX\_VALUE; //Double.MAX\_VALUE;

for (i = 1; i <= squery.length(); i++) {

// CostMatrix[i][0] = CostMatrix[i-1][0] + delete;

E[i][0] = Integer.MAX\_VALUE; //Double.POSITIVE\_INFINITY;

CostMatrix[i][0] = F[i][0] = delete + i \* gapExt;

}

for (j = 1; j <= ssubject.length(); j++) {

// CostMatrix[0][j] = CostMatrix[0][j - 1] + insert;

F[0][j] = Integer.MAX\_VALUE; //Double.POSITIVE\_INFINITY;

CostMatrix[0][j] = E[0][j] = insert + j \* gapExt;

}

for (i = 1; i <= squery.length(); i++)

for (j = 1; j <= ssubject.length(); j++) {

E[i][j] = Math.min(E[i][j - 1], CostMatrix[i][j - 1] + insert)

+ gapExt;

F[i][j] = Math.min(F[i - 1][j], CostMatrix[i - 1][j] + delete)

+ gapExt;

CostMatrix[i][j] = min(E[i][j], F[i][j], CostMatrix[i - 1][j - 1]

- matchReplace(squery, ssubject, i, j));

}

/\*

\* Traceback for affine gap penalties.

\*/

try {

boolean[] gap\_extend = {false, false};

j = this.CostMatrix[CostMatrix.length - 1].length - 1;

SymbolTokenization st = subMatrix.getAlphabet().getTokenization(

"default");

for (i = this.CostMatrix.length - 1; i > 0;) {

do {

// only Insert.

if (i == 0) {

align[0] = '~' + align[0];

align[1] = st.tokenizeSymbol(ssubject.symbolAt(j--)) + align[1];

path = ' ' + path;

// only Delete.

} else if (j == 0) {

align[0] = st.tokenizeSymbol(squery.symbolAt(i--)) + align[0];

align[1] = '~' + align[1];

path = ' ' + path;

// Match/Replace

} else if ((CostMatrix[i][j] == CostMatrix[i - 1][j - 1]

- matchReplace(squery, ssubject, i, j))

&& !(gap\_extend[0] || gap\_extend[1])) {

if (squery.symbolAt(i) == ssubject.symbolAt(j))

path = '|' + path;

else

path = ' ' + path;

align[0] = st.tokenizeSymbol(squery.symbolAt(i--)) + align[0];

align[1] = st.tokenizeSymbol(ssubject.symbolAt(j--)) + align[1];

// Insert || finish gap if extended gap is opened

} else if (CostMatrix[i][j] == E[i][j] || gap\_extend[0]) {

// check if gap has been extended or freshly opened

gap\_extend[0] = (E[i][j] != CostMatrix[i][j - 1] + insert

+ gapExt);

align[0] = '-' + align[0];

align[1] = st.tokenizeSymbol(ssubject.symbolAt(j--)) + align[1];

path = ' ' + path;

// Delete || finish gap if extended gap is opened

} else {

// check if gap has been extended or freshly opened

gap\_extend[1] = (F[i][j] != CostMatrix[i - 1][j] + delete

+ gapExt);

align[0] = st.tokenizeSymbol(squery.symbolAt(i--)) + align[0];

align[1] = '-' + align[1];

path = ' ' + path;

}

} while (j > 0);

}

} catch (BioException exc) {

throw new BioRuntimeException(exc);

}

/\*

\* No affine gap penalties, constant gap penalties, which is much faster

\* and needs less memory.

\*/

} else {

for (i = 1; i <= squery.length(); i++)

CostMatrix[i][0] = CostMatrix[i - 1][0] + delete;

for (j = 1; j <= ssubject.length(); j++)

CostMatrix[0][j] = CostMatrix[0][j - 1] + insert;

for (i = 1; i <= squery.length(); i++)

for (j = 1; j <= ssubject.length(); j++) {

CostMatrix[i][j] = min(CostMatrix[i - 1][j] + delete,

CostMatrix[i][j - 1] + insert, CostMatrix[i - 1][j - 1]

- matchReplace(squery, ssubject, i, j));

}

/\*

\* Traceback for constant gap penalties.

\*/

try {

j = this.CostMatrix[CostMatrix.length - 1].length - 1;

SymbolTokenization st = subMatrix.getAlphabet().getTokenization(

"default");

// System.out.println(printCostMatrix(CostMatrix,

// query.seqString().toCharArray(),

// subject.seqString().toCharArray()));

for (i = this.CostMatrix.length - 1; i > 0;) {

do {

// only Insert.

if (i == 0) {

align[0] = '~' + align[0];

align[1] = st.tokenizeSymbol(ssubject.symbolAt(j--)) + align[1];

path = ' ' + path;

// only Delete.

} else if (j == 0) {

align[0] = st.tokenizeSymbol(squery.symbolAt(i--)) + align[0];

align[1] = '~' + align[1];

path = ' ' + path;

// Match/Replace

} else if (CostMatrix[i][j] == CostMatrix[i - 1][j - 1]

- matchReplace(squery, ssubject, i, j)) {

if (squery.symbolAt(i) == ssubject.symbolAt(j))

path = '|' + path;

else

path = ' ' + path;

align[0] = st.tokenizeSymbol(squery.symbolAt(i--)) + align[0];

align[1] = st.tokenizeSymbol(ssubject.symbolAt(j--)) + align[1];

// Insert

} else if (CostMatrix[i][j] == CostMatrix[i][j - 1] + insert) {

align[0] = '-' + align[0];

align[1] = st.tokenizeSymbol(ssubject.symbolAt(j--)) + align[1];

path = ' ' + path;

// Delete

} else {

align[0] = st.tokenizeSymbol(squery.symbolAt(i--)) + align[0];

align[1] = '-' + align[1];

path = ' ' + path;

}

} while (j > 0);

}

} catch (BioException exc) {

throw new BioRuntimeException(exc);

}

}

/\*

\* From here both cases are equal again.

\*/

try {

squery = new SimpleGappedSequence(new SimpleSequence(

new SimpleSymbolList(squery.getAlphabet().getTokenization("token"),

align[0]), squery.getURN(), squery.getName(), squery

.getAnnotation()));

ssubject = new SimpleGappedSequence(new SimpleSequence(

new SimpleSymbolList(

ssubject.getAlphabet().getTokenization("token"), align[1]),

ssubject.getURN(), ssubject.getName(), ssubject.getAnnotation()));

Map<String, Sequence> m = new HashMap<String, Sequence>();

m.put(squery.getName(), squery);

m.put(ssubject.getName(), ssubject);

pairalign = new SimpleAlignment(m);

// this.printCostMatrix(queryChar, targetChar); // only for tests

// important

this.alignment = formatOutput(squery.getName(), // name of the query

// sequence

ssubject.getName(), // name of the target sequence

align, // the String representation of the alignment

path, // String match/missmatch representation

0, // Start position of the alignment in the query sequence

CostMatrix.length - 1, // End position of the alignment in the

// query sequence

CostMatrix.length - 1, // length of the query sequence

0, // Start position of the alignment in the target sequence

CostMatrix[0].length - 1, // End position of the alignment in the

// target sequence

CostMatrix[0].length - 1, // length of the target sequence

getEditDistance(), // the edit distance

System.currentTimeMillis() - time)

+ System.getProperty("line.separator"); // time consumption

// System.out.println(printCostMatrix(CostMatrix,

// query.seqString().toCharArray(), subject.seqString().toCharArray()));

return getEditDistance();

} catch (BioException exc) {

throw new BioRuntimeException(exc);

}

} else

throw new BioRuntimeException(

"Alphabet missmatch occured: sequences with different alphabet cannot be aligned.");

}