From: Duane Johnson <duane.johnson@gmail.com>
Subject: **Gene Alignment Project**
Date: March 17, 2008 2:44:48 PM MDT
To: Duane and Kelty Johnson <duane.johnson@gmail.com>

1 Attachment, 39.1 KB

CS 312 : Project 5
Duane Johnson

Methods

Pseudo-code for Scoring Algorithm

Input: sequence A of size n, sequence B of size m
Output: an integer value indicating edit distance

Create a *cost* array of size n
Initialize the *cost* array with increments of size IndelCost (5)

```
Loop from 1..b
    costBelow <- cost[0] + IndelCost
    Loop from 1..a
        diagonalCost <- cost[a - 1] + (A[a] == B[b] ? MatchCost : SubstCost)
        costHere <- MIN(cost[a] + IndelCost, costBelow + IndelCost, diagonalCost)
        cost[a - 1] <- costBelow
        costBelow <- costHere
```

Pseudo-code for Alignment Algorithm

Input: sequence A of size n, sequence B of size m
Output: console gets a list of aligned pairs, with a description of action taken to get there

Create a *cost* array of size n x m
Initialize the first row and first column of *cost* array to increments of size IndelCost (5)

```
Loop from 1..b
    Loop from 1..a
        diagonal <- cost[a - 1][b - 1] + (A[a] == B[b] ? MatchCost : SubstCost)
        cost[a][b] <- MIN(diagonal, cost[a][b - 1] + IndelCost, cost[a - 1][b] + IndelCost)
```

Display the results by backtracking from cost[n][m] to cost[0][0]

How Alignment / Extraction Works

The alignment algorithm works by considering all possible actions in each step (cell) along the way and caching the results in an n x m matrix. The action that is chosen (Match/Subst/Indel) is an optimal path from the start to the current location in the matrix, therefore future computations can rely on sub-problems to reduce computational load.

The results of the alignment are displayed by backtracking through the cached results and finding the minimum path again. Depending on which branch of the 3-way branches leads to a minimal cost, the branch is chosen and the operation (Match/Subst/Indel) is displayed.

Proof of Algorithm Complexity

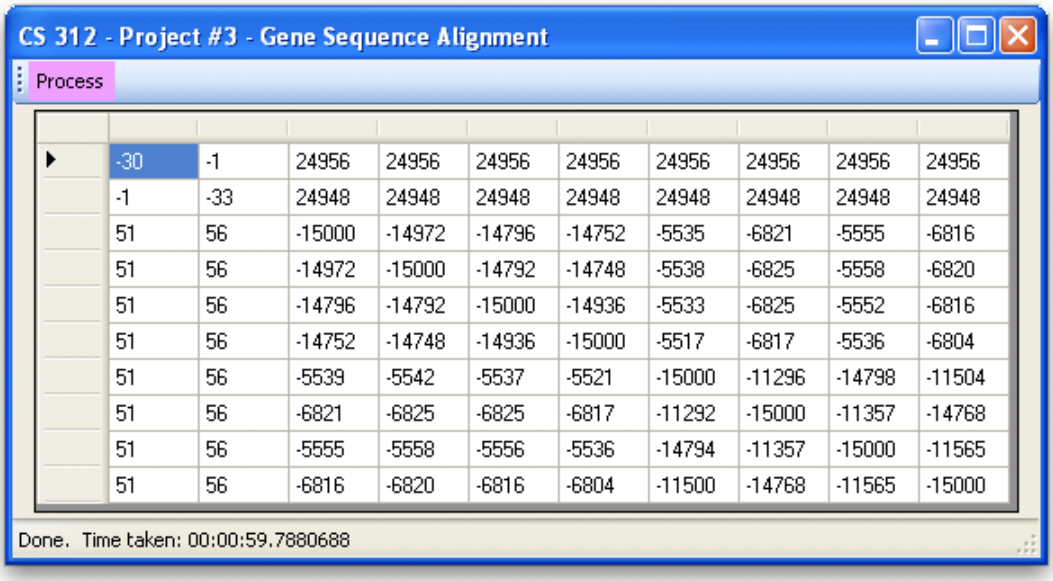
Scoring Algorithm:
- Space complexity is O(n) because the cost array is reused.
- Time complexity is O(n*m):
  - Consider array lookups and data assignments as O(1) operations;
  - outer loop is n, inner loop is m;
  - therefore, the algorithm is O(n*m) because the time depends only on the loops

Extraction Algorithm
- Space complexity is O(n*m) because the cached results are needed in step 2
- Time complexity is O(n*m):
  - Just like the scoring algorithm above, but multiplied by a constant of 2;

the constant is simply due to retracing our steps to display the alignment information

Results

Screenshot of 10x10 Scoring Matrix



-30	-1	24956	24956	24956	24956	24956	24956	24956	24956
-1	-33	24948	24948	24948	24948	24948	24948	24948	24948
51	56	-15000	-14972	-14796	-14752	-5535	-6821	-5555	-6816
51	56	-14972	-15000	-14792	-14748	-5538	-6825	-5558	-6820
51	56	-14796	-14792	-15000	-14936	-5533	-6825	-5552	-6816
51	56	-14752	-14748	-14936	-15000	-5517	-6817	-5536	-6804
51	56	-5539	-5542	-5537	-5521	-15000	-11296	-14798	-11504
51	56	-6821	-6825	-6825	-6817	-11292	-15000	-11357	-14768
51	56	-5555	-5558	-5556	-5536	-14794	-11357	-15000	-11565
51	56	-6816	-6820	-6816	-6804	-11500	-14768	-11565	-15000

Alignment of Taxa 3 and Taxa 4

Whew... it's a long one. Here ya go... source code is at the end.

a a Match
g g Match
t t Match
g g Match
t t Match
c t Subst
t t Match
t t Match
g g Match
t t Match
g g Match
t t Match
a a Match
a a Match
a a Match
g g Match
g g Match
a a Match
t t Match
c c Match
t t Match
g g Match
a a Match
g g Match
a a Match
a a Match
t t Match
g g Match
g g Match
t t Match
t t Match
t t Match

t t Match
a g Subst
a a Match
a a Match
a a Match
g g Match
t t Match
g g Match
g g Match
a a Match
t t Match
g g Match
t t Match
c c Match
c c Match
g g Match
t t Match
g g Match
t t Match
t t Match
t t Match
g g Match
g g Match
a a Match
c c Match
a a Match
a a Match
t t Match
c c Match
a a Match
c c Match
t t Match
t t Match
t t Match
a a Match
a a Match
c t Subst
t t Match
g g Match
t t Match
g g Match
g g Match
t t Match
a a Match
g g Match
t t Match
t t Match
g g Match
t t Match
c c Match
a a Match
a a Match
t t Match
c c Match
g g Match
t t Match
t t Match
c c Match
t t Match
a a Match
c c Match
a a Match
g g Match
g g Match
t t Match
g g Match
a a Match
a a Match

a a Match
t t Match
a a Match
g g Match
g g Match
t t Match
t t Match
t t Match
t t Match
t t Match
g g Match
t t Match
t t Match
c c Match
c c Match
g g Match
a a Match
a a Match
a a Match
t t Match
t t Match
a a Match
a a Match
a a Match
c c Match
t t Match
c c Match
g g Match
t t Match
t t Match
g g Match
t t Match
c c Match
a a Match
t t Match
c c Match
g g Match
g g Match
a a Match
a a Match
t t Match
a a Match
a a Match
t t Match
t t Match
t t Match
t t Match
g g Match
a a Match
t t Match
g g Match
g g Match
g g Match
a a Match
c c Match
g g Match
a a Match
c c Match
a a Match
t t Match
g g Match
t t Match
a a Match
t t Match
t t Match
g g Match
g g Match
t t Match

a a Match
g g Match
c t Subst
g g Match
t t Match
t t Match
a a Match
g g Match

Source Code

```
private int Score(GeneSequence seqA, GeneSequence seqB)
{

   int maxA = Math.Min(seqA.Sequence.Length, MaxCharactersToAlign);
   int maxB = Math.Min(seqB.Sequence.Length, MaxCharactersToAlign);
   int a, b;
   int[] cost = new int[maxA];
   int costBelow = 0, costHere = 0;

   // Cost of first cell depends on match or not
   cost[0] = seqA.Sequence[0] == seqB.Sequence[0] ? MatchCost : SubstCost;
   // Initialize the cost array by computing the first column
   for (a = 1; a < maxA; a++) cost[a] = cost[a - 1] + IndelCost;

   for (b = 1; b < maxB; b++)
   {
      // Before the inner loop, calculate the bottom row's cost
      costBelow = cost[0] + IndelCost;
      for (a = 1; a < maxA; a++)
      {
         bool isMatching = seqA.Sequence[a] == seqB.Sequence[b];
         int diagonal = cost[a - 1] + (isMatching ? MatchCost : SubstCost);
         int left = cost[a] + IndelCost;
         int below = costBelow + IndelCost;

         costHere = Math.Min(diagonal, Math.Min(left, below));

         // Prepare for the next column, using an N-space cost array
         cost[a - 1] = costBelow;
         // The next row up will use a "cost below" of whatever the cost is here
         costBelow = costHere;
      }
   }

   //Console.WriteLine("Cost is " + costHere);
   return costHere;
}

private void showCost(int[] cost, int max, int column)
{
   Console.WriteLine("Cost at column " + column.ToString());
   for (int i = 0; i < max; i++)
   {
      Console.WriteLine(i.ToString() + ": " + cost[i].ToString());
   }
}

private void Extract(GeneSequence seqA, GeneSequence seqB)
{
   int maxA = Math.Min(seqA.Sequence.Length, MaxCharactersToAlign);
   int maxB = Math.Min(seqB.Sequence.Length, MaxCharactersToAlign);
   int a, b;
   int[][] cost = new int[maxA + 1][];;
   for (a = 0; a < maxA + 1; a++)
      cost[a] = new int[maxB + 1];

   // Initialize sequence A column of insertions
```

```
    for (a = 0; a < maxA + 1; a++)
      cost[a][0] = a * IndelCost;

    // Initialize sequence B row of insertions
    for (b = 0; b < maxB + 1; b++)
      cost[0][b] = b * IndelCost;

    // Calculate the Cost matrix
    for (b = 1; b < maxB + 1; b++)
    {
      for (a = 1; a < maxA + 1; a++)
      {
        bool isMatching = seqA.Sequence[a - 1] == seqB.Sequence[b - 1];
        int diagonal = cost[a - 1][b - 1] + (isMatching ? MatchCost : SubstCost);
        int left = cost[a][b - 1] + IndelCost;
        int below = cost[a - 1][b] + IndelCost;
        // Use the min path, wherever it leads us
        cost[a][b] = Math.Min(diagonal, Math.Min(left, below));
      }
    }

    Console.WriteLine("{0} x {0} cost array", maxA + 1, maxB + 1);

    // Display the results of the alignment in the console
    int pathA = maxA, pathB = maxB;
    while (pathA > 0 && pathB > 0)
    {
      int diagonal = cost[pathA - 1][pathB - 1];
      int left = cost[pathA][pathB - 1];
      int below = cost[pathA - 1][pathB];
      char charA = seqA.Sequence[pathA - 1];
      char charB = seqB.Sequence[pathB - 1];

      if (diagonal <= left && diagonal <= below)
      {
        Console.Write(charA.ToString() + " " + charB.ToString());
        if (charA == charB)
        {
          Console.WriteLine(" Match");
        }
        else
        {
          Console.WriteLine(" Subst");
        }
        pathA--;
        pathB--;
      }
      else if (left <= diagonal && left <= below)
      {
        Console.WriteLine(charA.ToString() + " - Indel");
        pathB--;
      }
      else // min is below
      {
        Console.WriteLine("- " + charB.ToString() + " Indel");
        pathA--;
      }
    }
  }
}
```