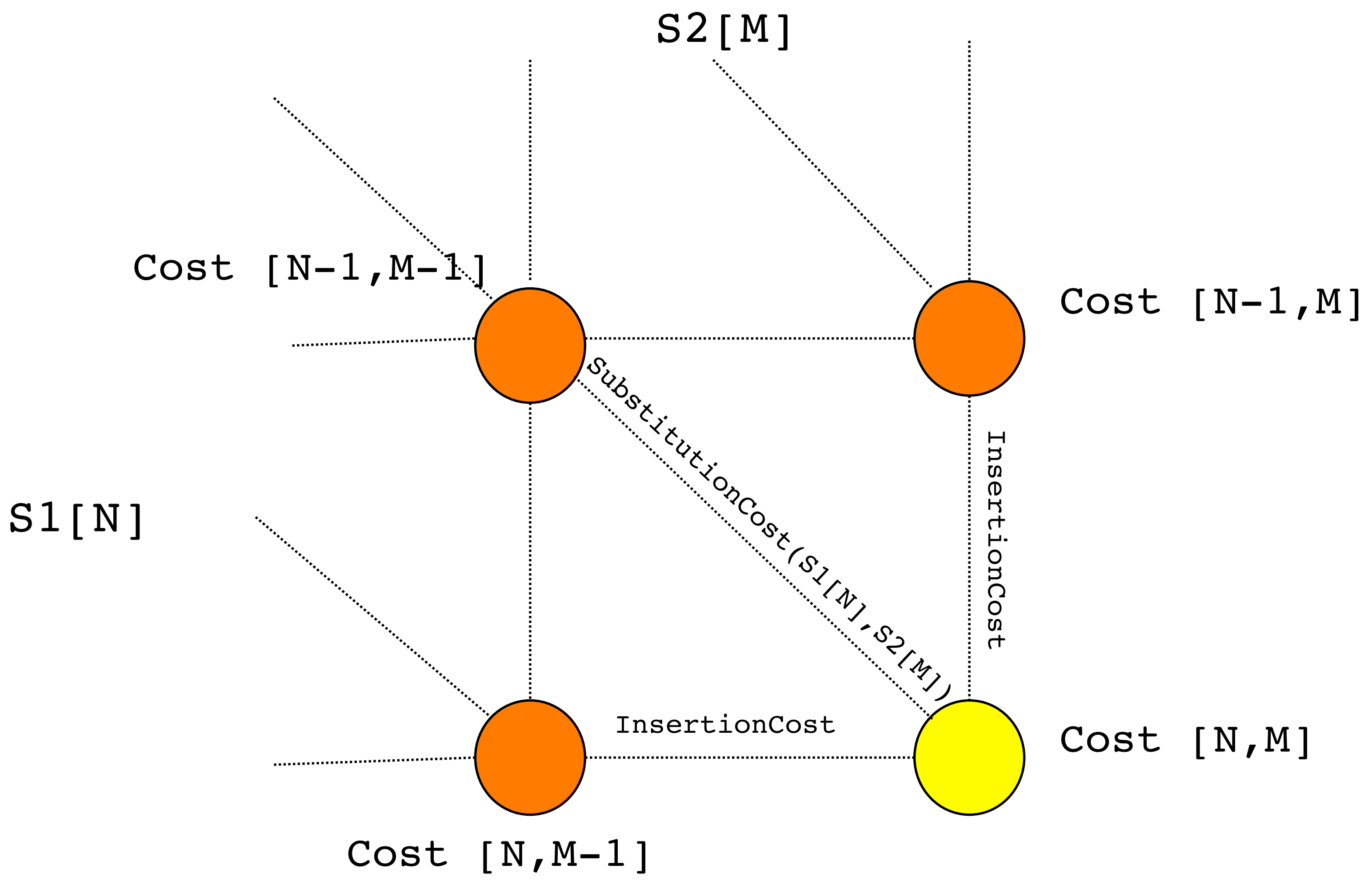
4. Sequence comparison

- How to predict gene/protein functions?
- Why gene/protein sequences may be similar?
- Measuring sequence similarity
- Aligning sequences is an optimization problem
- A sequence alignment as a path
- A path is optimal if all its sub-paths are optimal
- Alignment costs
- **A recursive algorithm**
- Recursion can be avoided: an iterative version
- How efficient is this algorithm?


S2[M]
Cost [N-1,M-1]
Cost [N-1,M]
SubstitutionCost(S1[N],S2[M])
InsertionCost
S1[N]
InsertionCost
Cost [N,M]
Cost [N,M-1]

Recursive functions

- A recursive function is called during its current execution

- Two conditions have to be satisfied when writing a recursive function
  1. A termination condition must be checked as soon as the function is entered
  2. The inside call(s) to the function must apply on a sub-problem, i.e. a “smaller” problem

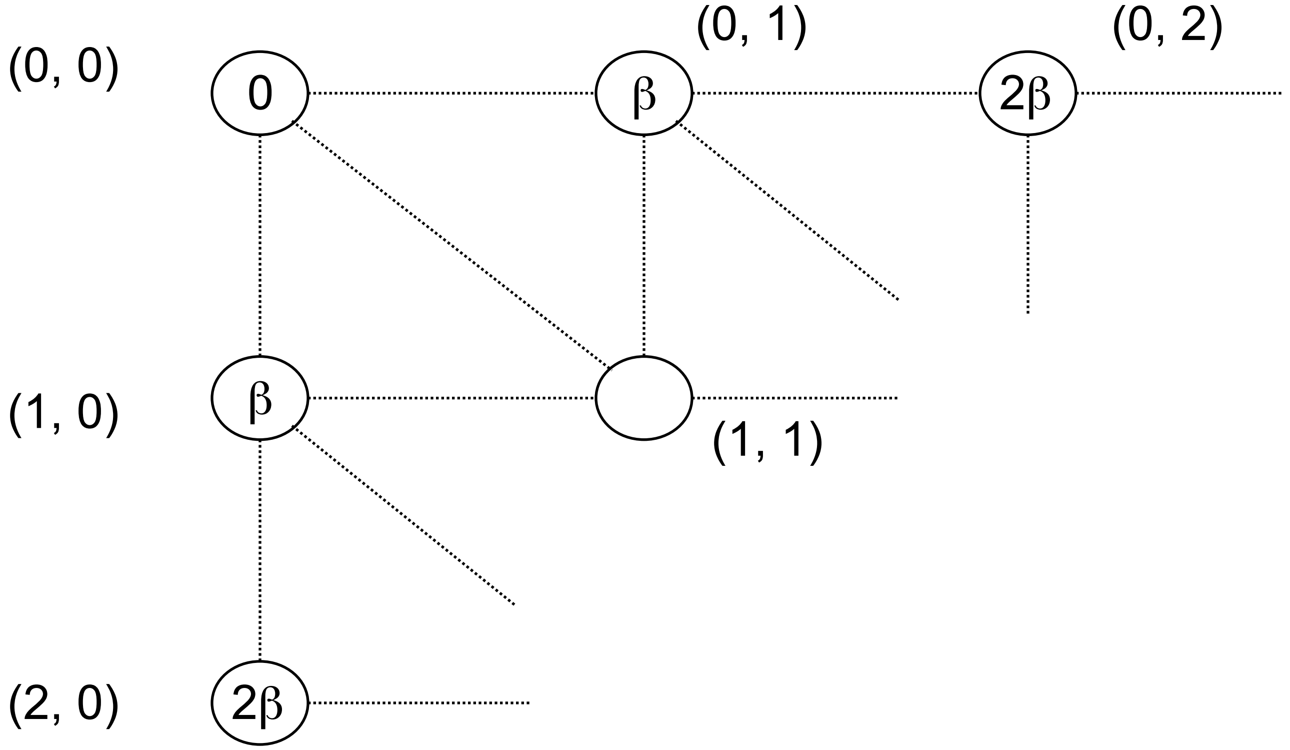
Computing the costs recursively

```
function ComputeCost (I, J: integer) returns integer



        ComputeCost(I-1, J-1) +
          substitutionCost(Sequence1[I], Sequence2[J])
        ComputeCost(I, J-1) + InsertionCost
        ComputeCost(I-1, J) + InsertionCost
```

Computing the costs recursively

```
function ComputeCost (I, J: integer) returns integer
 if I = 0 and J =0 then return 0
   else
      return Min (
         ComputeCost(I-1, J-1) +
           substitutionCost(Sequence1[I], Sequence2[J])
         ComputeCost(I, J-1) + InsertionCost
         ComputeCost(I-1, J) + InsertionCost
         )
```

β: `InsertionCost`



Computing the costs recursively

```
function ComputeCost (I, J: integer) returns integer
 if I = 0 and J =0 then return 0
   else if I = 0 and J ≥ 1 then return J*InsertionCost
   else if I ≥ 1 and J = 0 then return I*InsertionCost
   else
      return Min (
         ComputeCost(I-1, J-1) +
           substitutionCost(Sequence1[I], Sequence2[J])
         ComputeCost(I, J-1) + InsertionCost
         ComputeCost(I-1, J) + InsertionCost
         )
```