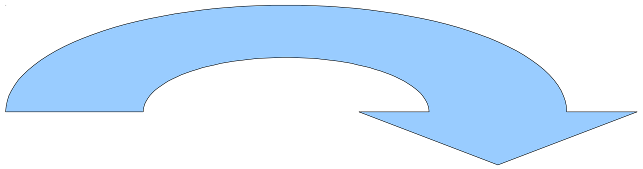
MLiB - Mandatory Project 3

Gene finding using HMMs

Viterbi decoding

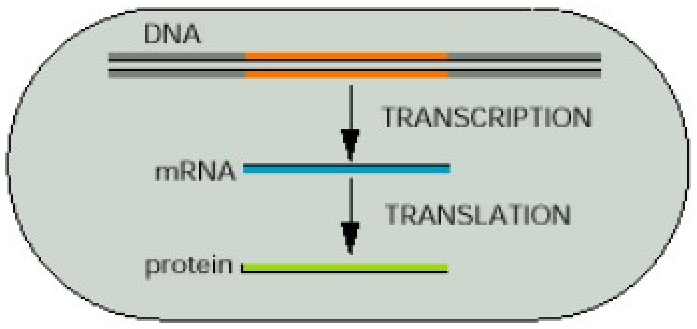


```
>NC_002737.1 Streptococcus pyogenes M1 GAS
TTGTTGATATTCTGTTTTTTCTTTTTTAGTTTTCCACATGAAAAATAGTTGAAAACAATA
GCGGTGTCCCCTTAAAATGGCTTTTCCACAGGTTGTGGAGAACCCAAATTAACAGTGTTA
ATTTATTTTCCACAGGTTGTGGAAAAACTAACTATTATCCATCGTTCTGTGGAAAACTAG
AATAGTTTATGGTAGAATAGTTCTAGAATTATCCACAAGAAGGAACCTAGTATGACTGAA
AATGAACAAATTTTTTGGAACAGGGTCTTGGAATTAGCTCAGAGTCAATTAAAACAGGCA
ACTTATGAATTTTTTGTTCATGATGCCCGTCTATTAAAGGTCGATAAGCATATTGCAACT
ATTTACTTAGATCAAATGAAAGAGCTCTTTTGGGAAAAAAATCTTAAAGATGTTATTCTT
ACTGCTGGTTTTGAAGTTTATAACGCTCAAATTTCTGTTGACTATGTTTTCGAAGAAGAC
CTAATGATTGAGCAAAATCAGACCAAAATCAACCAAAAACCTAAGCAGCAAGCCTTAAAT
TCTTTGCCTACTGTTACTTCAGATTTAAACTCGAAATATAGTTTTGAAAACTTTATTCAA
GGAGATGAAAATCGTTGGGCTGTTGCTGCTTCAATAGCAGTAGCTAATACTCCTGGAACT
ACCTATAATCCTTTGTTTATTTGGGGTGGCCCTGGGCTTGGAAAAACCCATTTATTAAAT
GCTATTGGTAATTCTGTACTATTAGAAAATCCAAATGCTCGAATTAAATATATCACAGCT
GAAAACTTTATTAATGAGTTTGTTATCCATATTCGCCTTGATACCATGGATGAATTGAAA
GAAAAATTTCGTAATTTAGATTTACTCCTTATTGATGATATCCAATCTTTAGCTAAAAAA
ACGCTCTCTGGAACACAAGAAGAGTTCTTTAATACTTTTAATGCACTTCATAATAATAAC
AAACAAATTGTCCTAACAAGCGACCGTACACCAGATCATCTCAATGATTTAGAAGATCGA
TTAGTTACTCGTTTTAAATGGGGATTAACAGTCAATATCACACCTCCTGATTTTGAAACA
CGAGTGGCTATTTTGACAAATAAAATTCAAGAATATAACTTTATTTTTCCTCAAGATACC
ATTGAGTATTTGGCTGGTCAATTTGATTCTAATGTCAGAGATTTAGAAGGTGCCTTAAAA
GATATTAGTCTGGTTGCTAATTTCAAACAAATTGACACGATTACTGTTGACATTGCTGCC
GAAGCTATTCGCGCCAGAAAGCAAGATGGACCTAAAATGACAGTTATTCCCATCGAAGAA
ATTCAAGCGCAAGTTGGAAAATTTTACGGTGTTACCGTCAAAGAAATTAAAGCTACTAAA
CGAACACAAAATATTGTTTTAGCAAGACAAGTAGCTATGTTTTTAGCACGTGAAATGACA
GATAACAGTCTTCCTAAAATTGGAAAAGAATTTGGTGGCAGAGACCATTCAACAGTACTC
CATGCCTATAATAAAATCAAAAACATGATCAGCCAGGACGAAAGCCTTAGGATCGAAATT
GAAACCATAAAAAACAAAATTAAATAACATGTGGAAAAGAATATCTTTTATGAAATAGTT
ATCCACAAGTTGTGAACATCCATTTAGTCTTGGATTCTCTCGTTTATTTAGAGTTATCCA
CTATATACACAAGACCTACTACTACTACTTATTATTATACTTATTAAATAAAGGAGTTCT
```

```
>NC_002737.1 gene annotation Streptococcus pyogenes M1 GAS
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
```

Design a HMM that models the syntax of genes

Gene structure in procaryotes

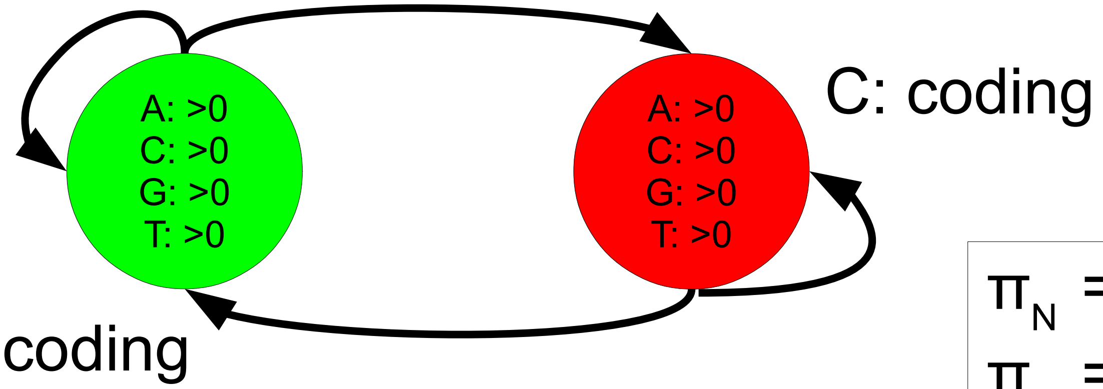


Biological facts

- The gene is a substring of the DNA sequence of A,C,G,T's

Z: NNNCCCCCCCCCNNNNNNNNCCCCCCCCCCCCCCCNNNNNNNNNNN

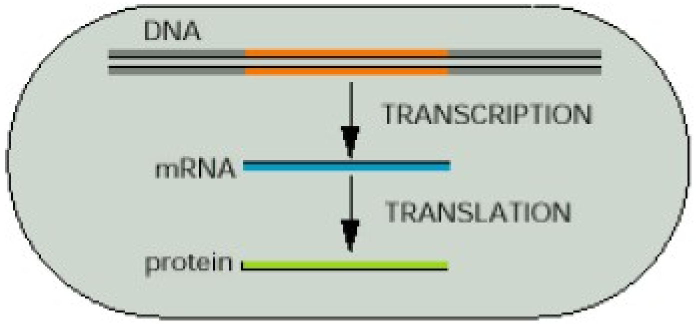
X: acgatgcgctaatatgtccgatgacgtgagcataagcgacatgcag



$\pi_N = 1$

$\pi_C = 0$

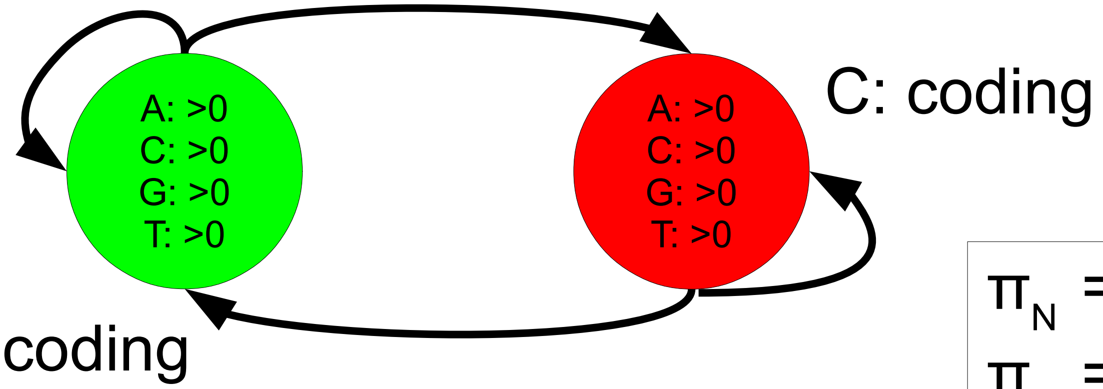
Gene structure in procaryotes



Biological facts

- The gene is a substring of the DNA sequence of A,C,G,T's
- The gene starts with a start-code **atg**

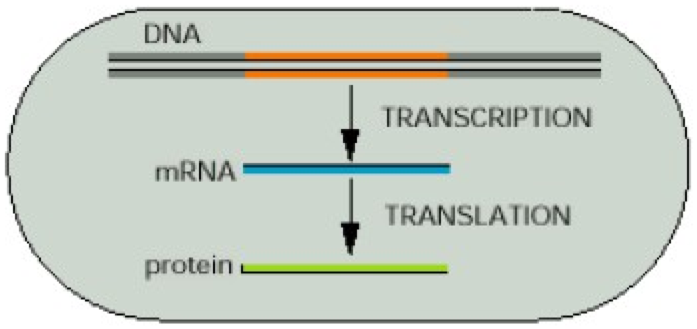
```
Z:  NNNCCCCCCCCCNNNNNNNNCCCCCCCCCCCCCCNNNNNNNNNNN

X:  acgatgcgctaatatgtccgatgacgtgagcataagcgacatgcag
```



$\pi_N = 1$

$\pi_C = 0$

Gene structure in procaryotes



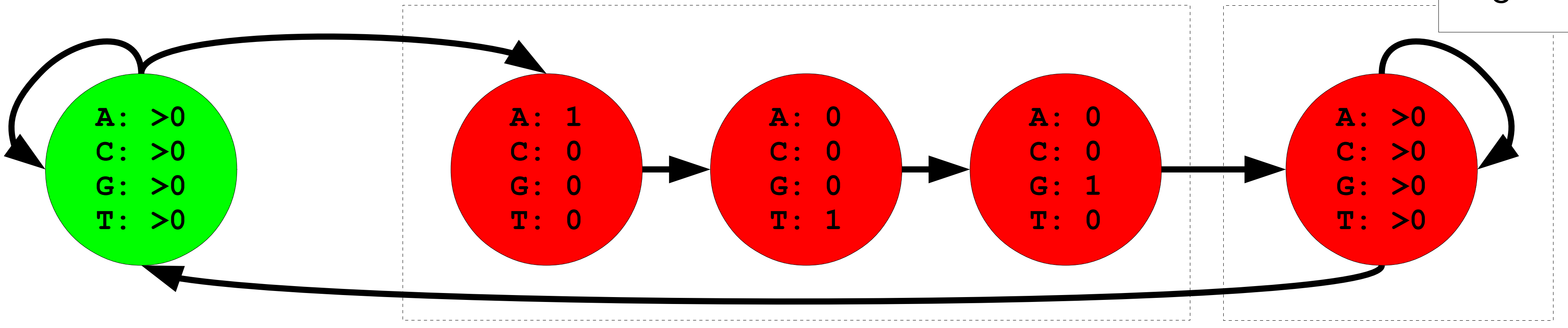
Biological facts

- The gene is a substring of the DNA sequence of A,C,G,T's
- The gene starts with a start-code **atg**

Z: NNNCCCCCCCCCNNNNNNNNCCCCCCCCCCCCCCCNNNNNNNNNNN

X: acgatgcgctaatatgtccgatgacgtgagcataagcgacat

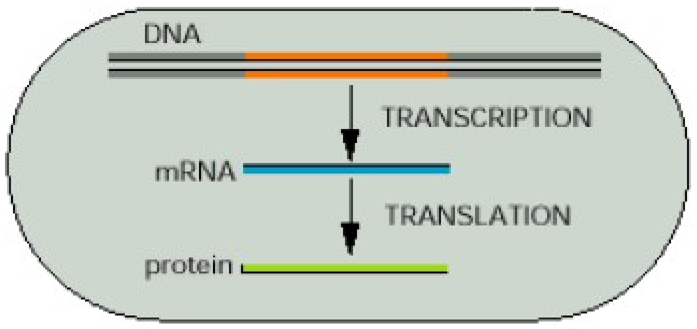
$\pi_N = 1$
$\pi_C = 0$



N: non-coding

C: coding

Gene structure in procaryotes



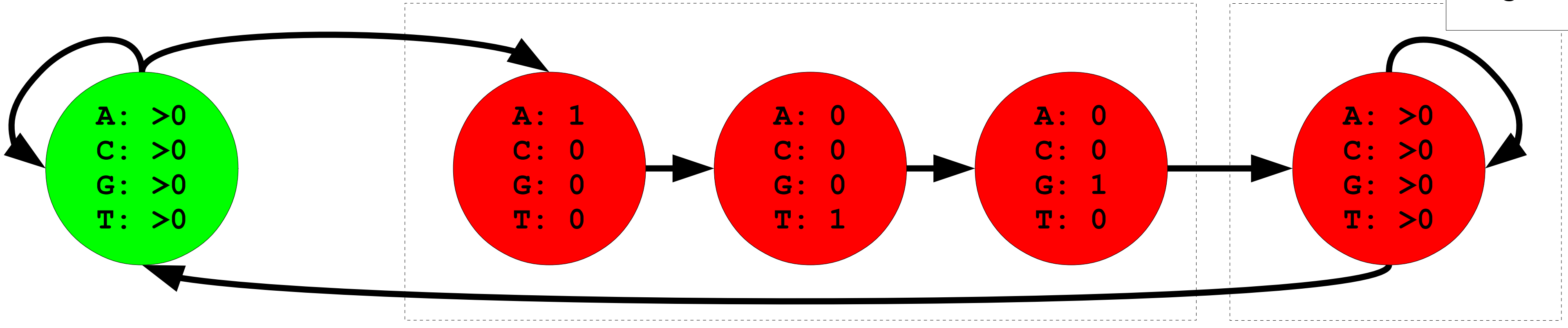
Biological facts

- The gene is a substring of the DNA sequence of A,C,G,T's
- The gene starts with a start-code **atg**
- The gene ends with a stop-codon **taa**, **tag** or **tga**

Z: NNNCCCCCCCCCNNNNNNNNCCCCCCCCCCCCCCCNNNNNNNNNNN

X: acgatgcgctaatatgtccgatgacgtgagcataagcgacat

$\pi_N = 1$
$\pi_C = 0$

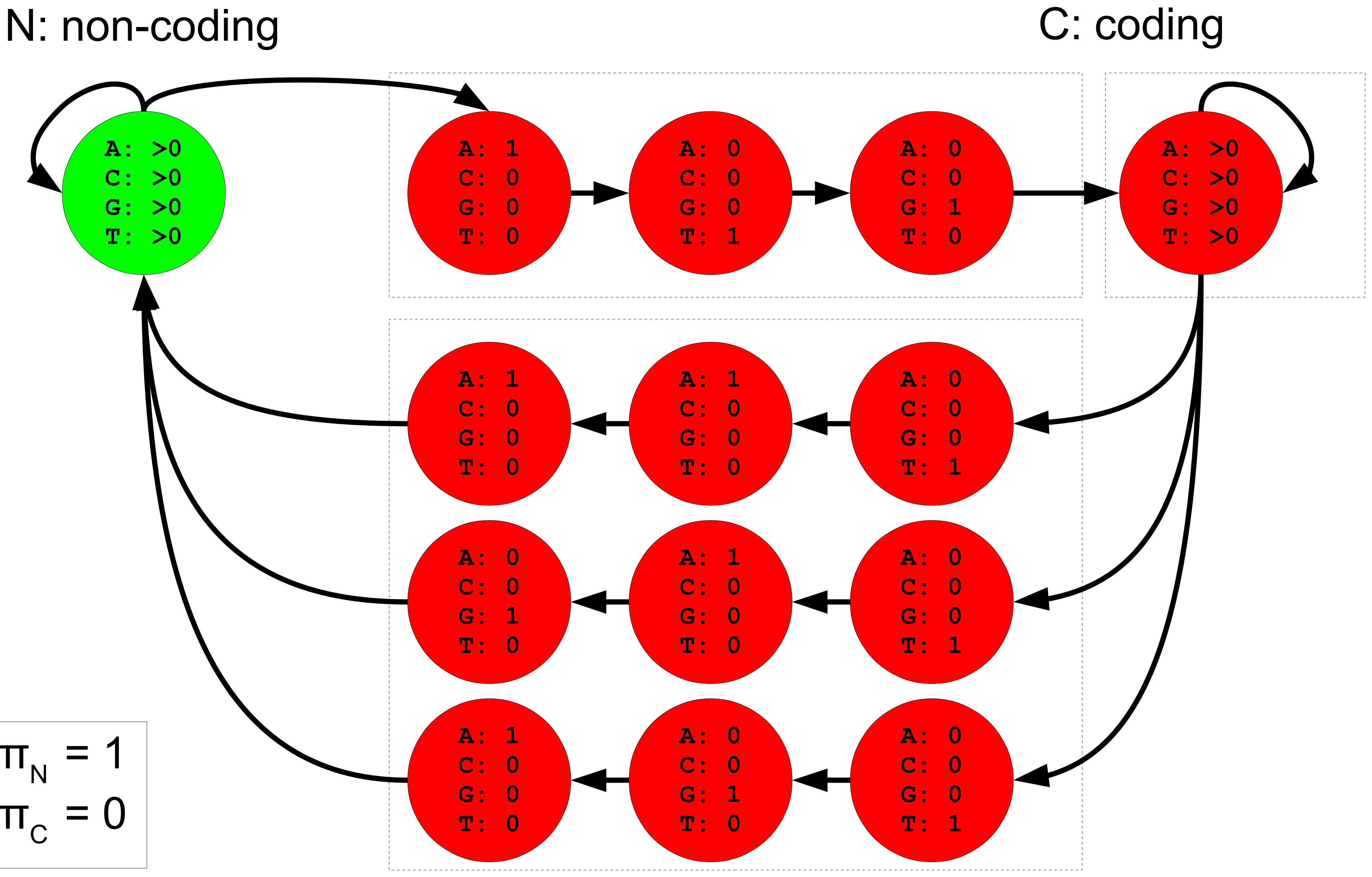


N: non-coding

C: coding

Gene struct

- The gene is a substring of the DNA sequence of A,C,G,T's
- The gene starts with a start-code **atg**
- The gene ends with a stop-codon **taa**, **tag** or **tga**

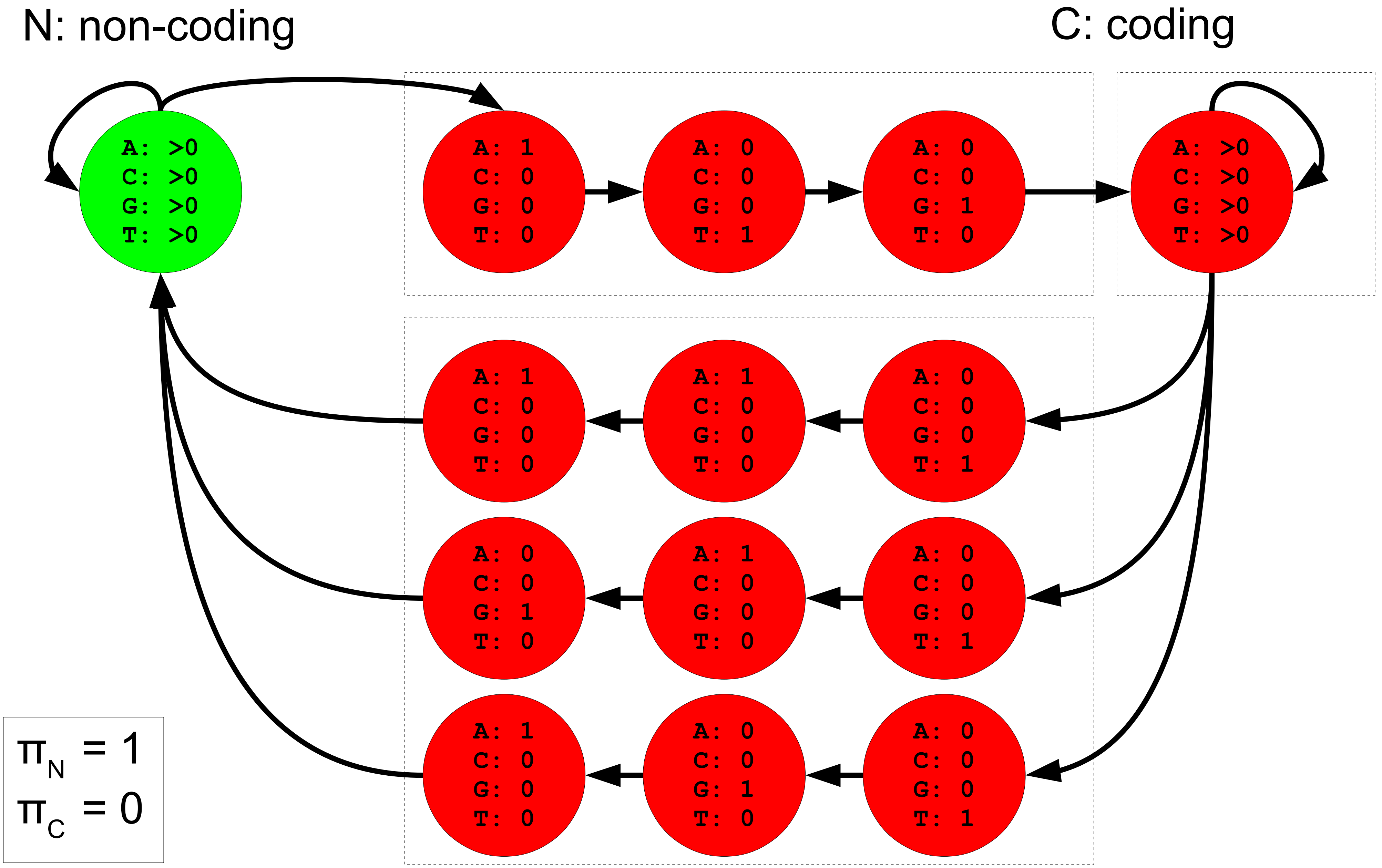


$\pi_N = 1$

$\pi_C = 0$

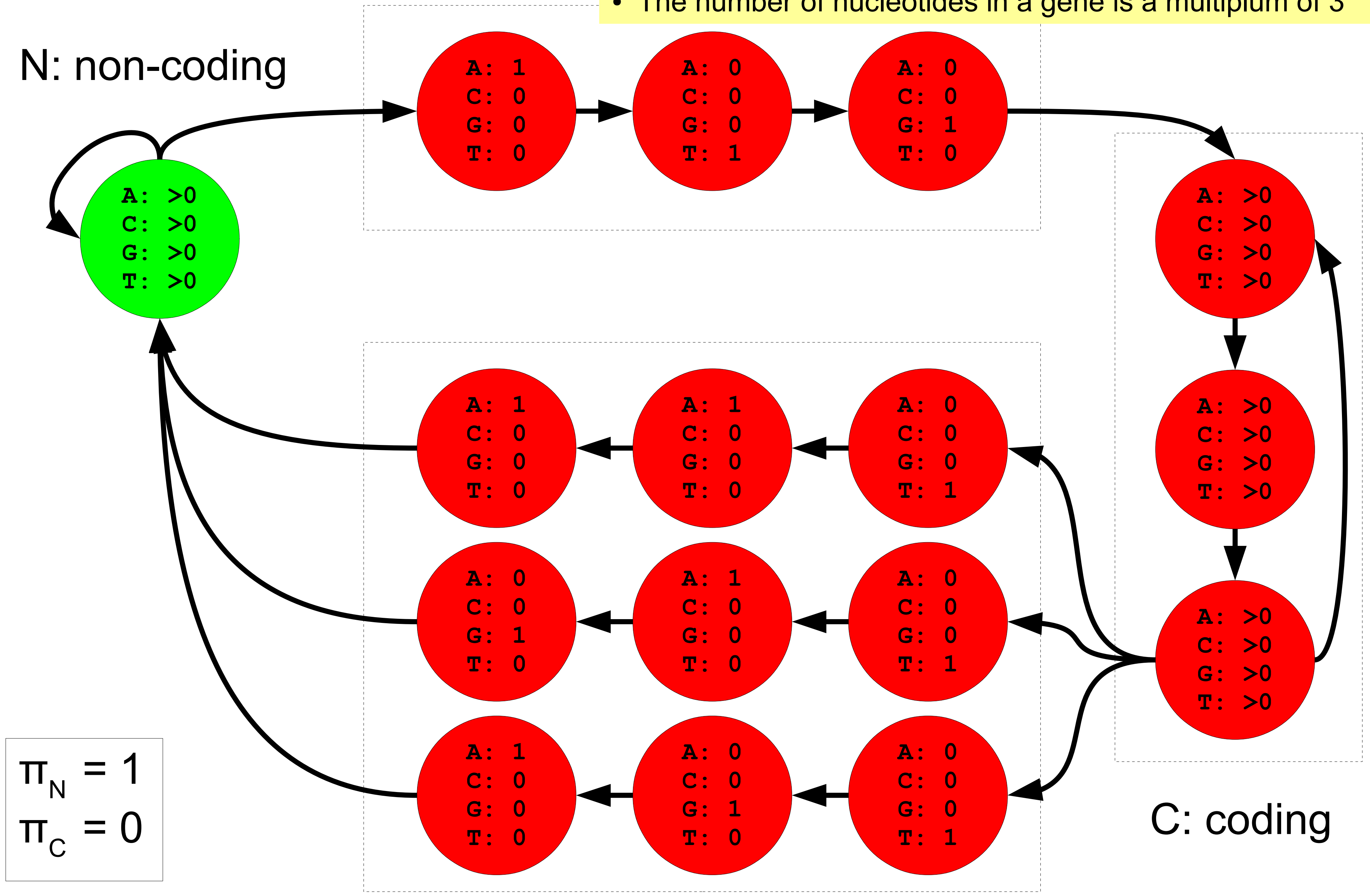
Gene structu

- The gene is a substring of the DNA sequence of A,C,G,T's
- The gene starts with a start-code **atg**
- The gene ends with a stop-codon **taa**, **tag** or **tga**
- The number of nucleotides in a gene is a multiplum of 3



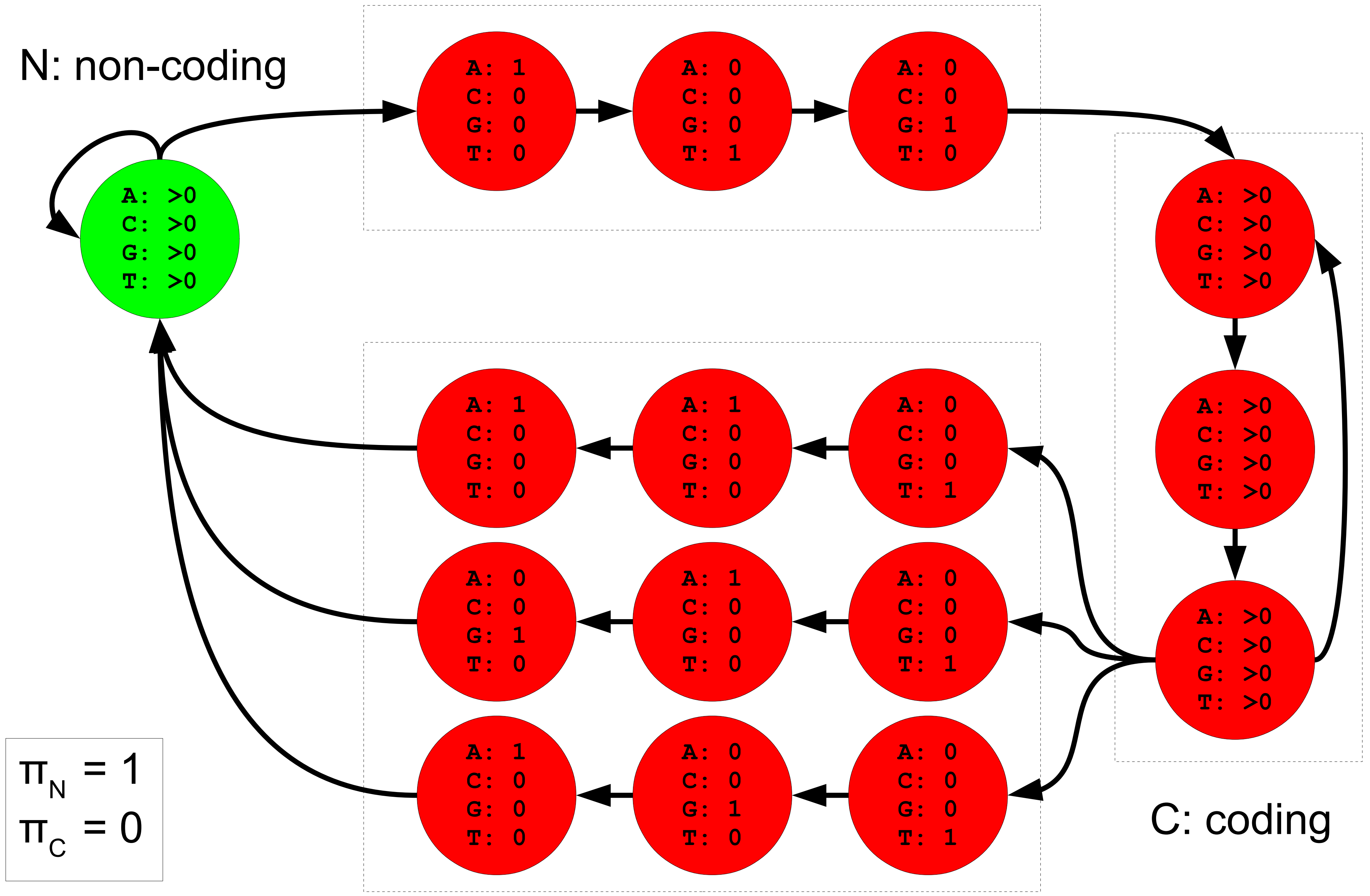
Gene structu

- The gene is a substring of the DNA sequence of A,C,G,T's
- The gene starts with a start-code **atg**
- The gene ends with a stop-codon **taa**, **tag** or **tga**
- The number of nucleotides in a gene is a multiplum of 3



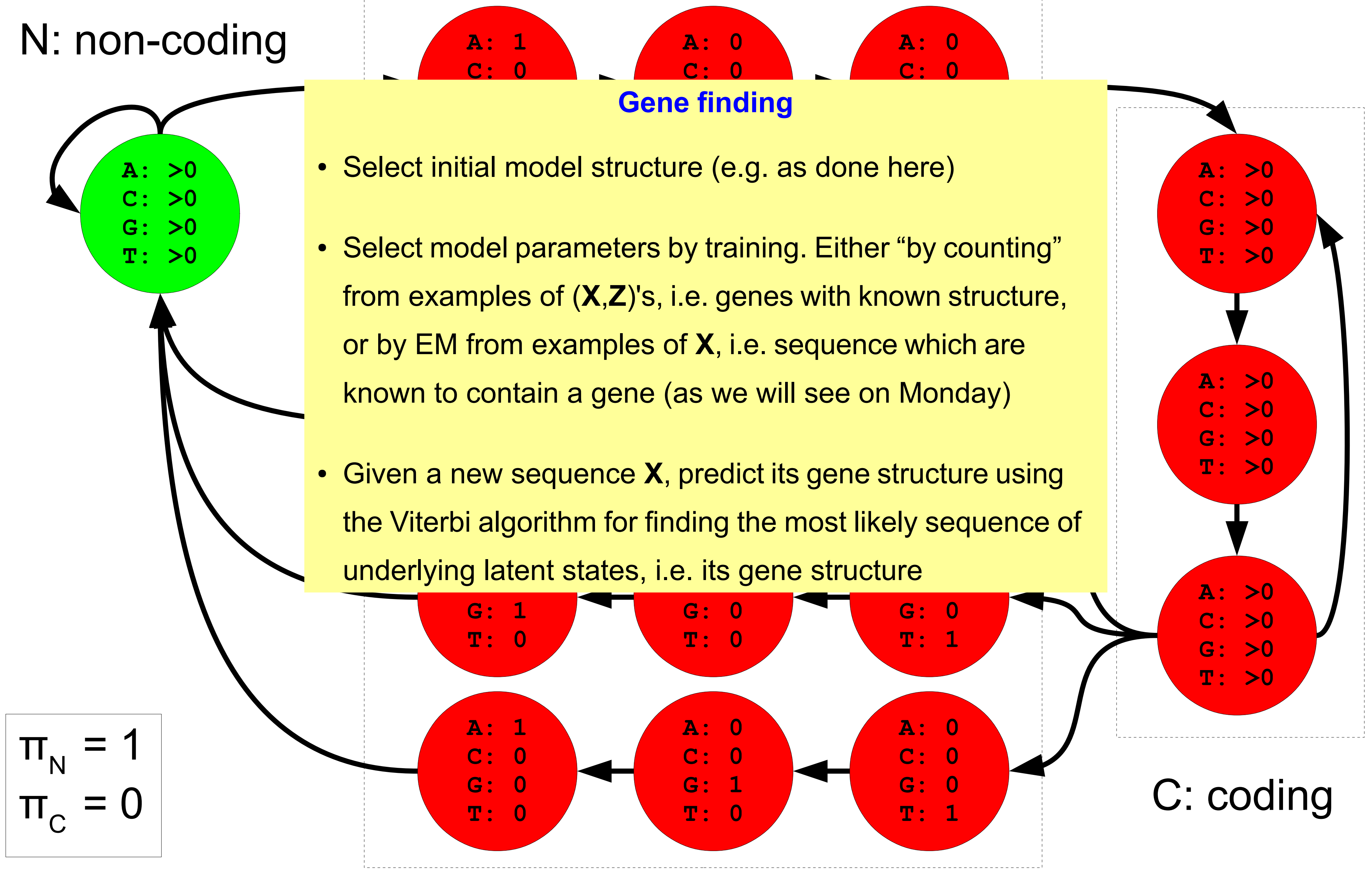
$\pi_N = 1$
$\pi_C = 0$

Gene structure in procaryotes



$\pi_N = 1$
$\pi_C = 0$

Gene structure in procaryotes



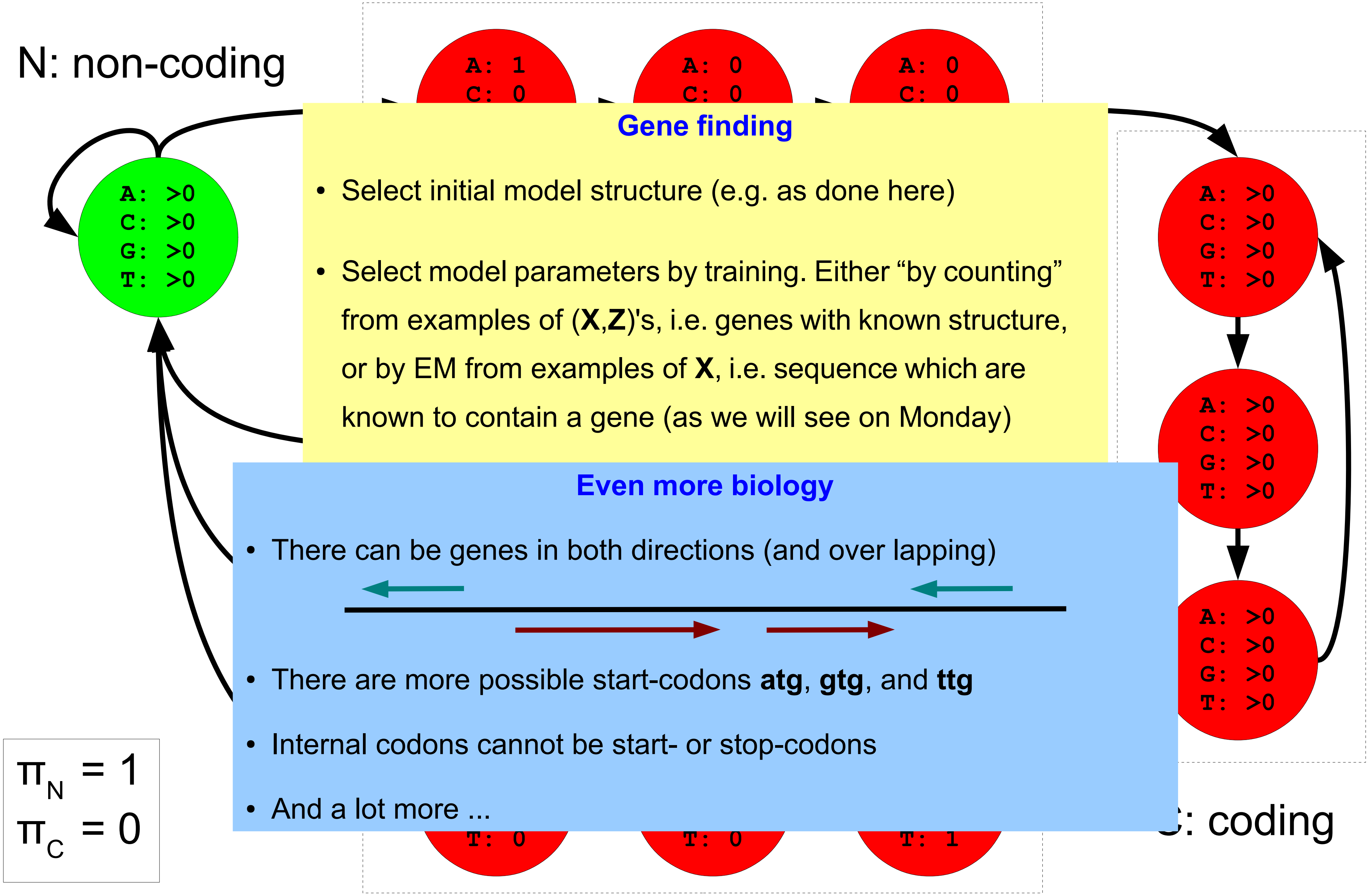
$\pi_N = 1$

$\pi_C = 0$

Gene finding

- Select initial model structure (e.g. as done here)
- Select model parameters by training. Either "by counting" from examples of (**X**,**Z**)'s, i.e. genes with known structure, or by EM from examples of **X**, i.e. sequence which are known to contain a gene (as we will see on Monday)
- Given a new sequence **X**, predict its gene structure using the Viterbi algorithm for finding the most likely sequence of underlying latent states, i.e. its gene structure

Example – Gene finding

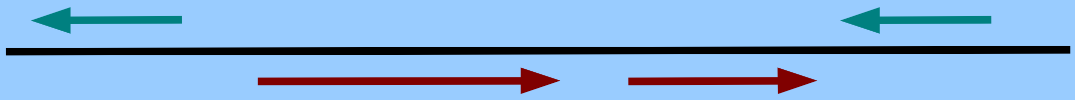


Gene finding

- Select initial model structure (e.g. as done here)
- Select model parameters by training. Either “by counting” from examples of (**X**,**Z**)'s, i.e. genes with known structure, or by EM from examples of **X**, i.e. sequence which are known to contain a gene (as we will see on Monday)

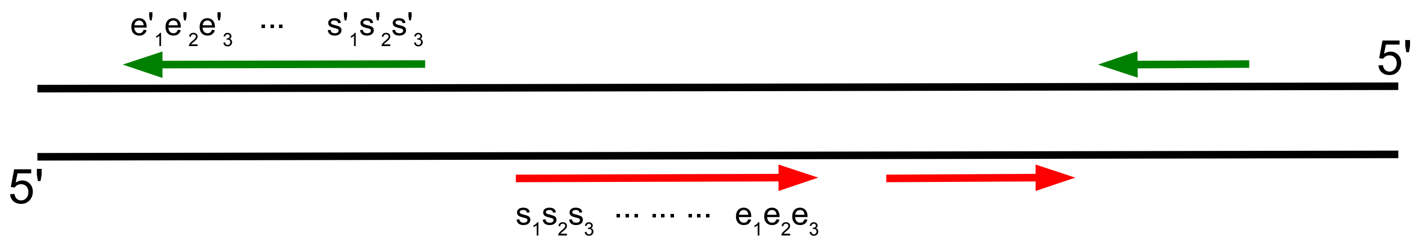
Even more biology

- There can be genes in both directions (and over lapping)

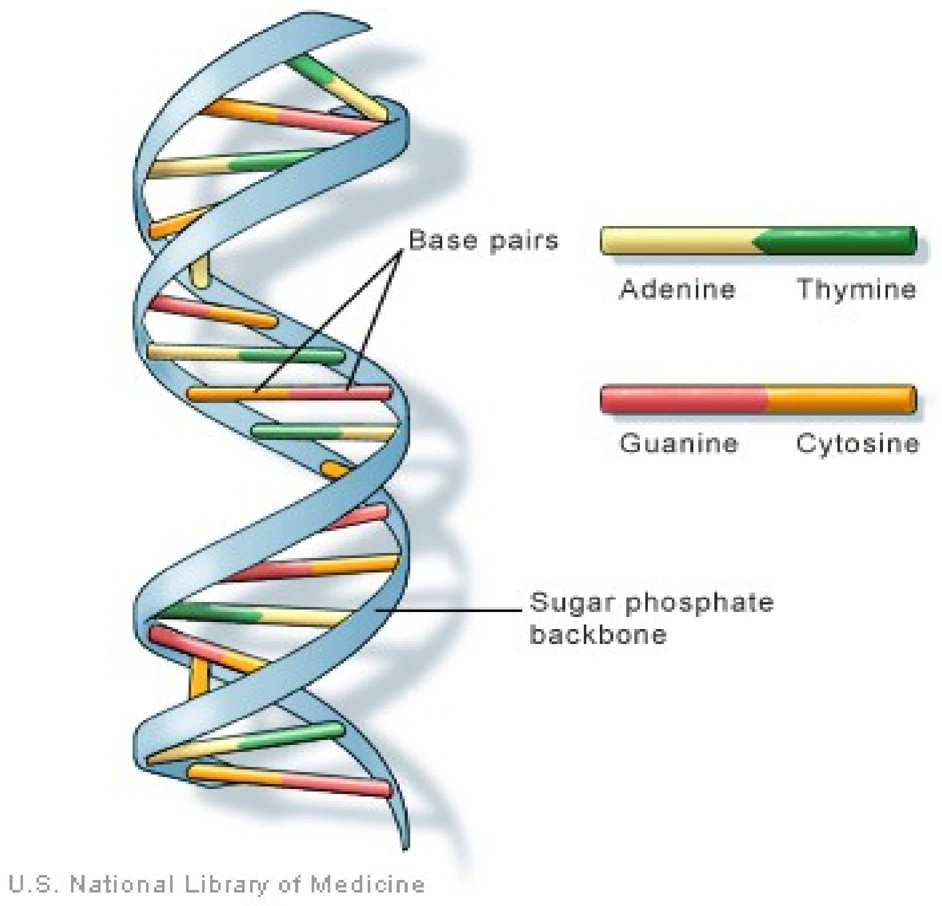


- There are more possible start-codons **atg**, **gtg**, and **ttg**
- Internal codons cannot be start- or stop-codons
- And a lot more ...

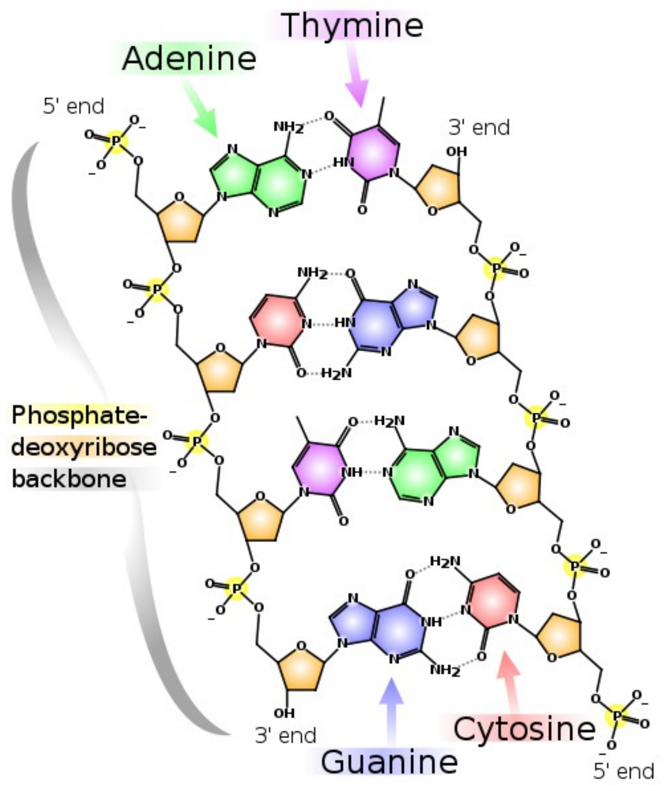
DNA



$e'_1e'_2e'_3$ ··· $s'_1s'_2s'_3$
5'
5'
$s_1s_2s_3$ ··· ··· ··· $e_1e_2e_3$

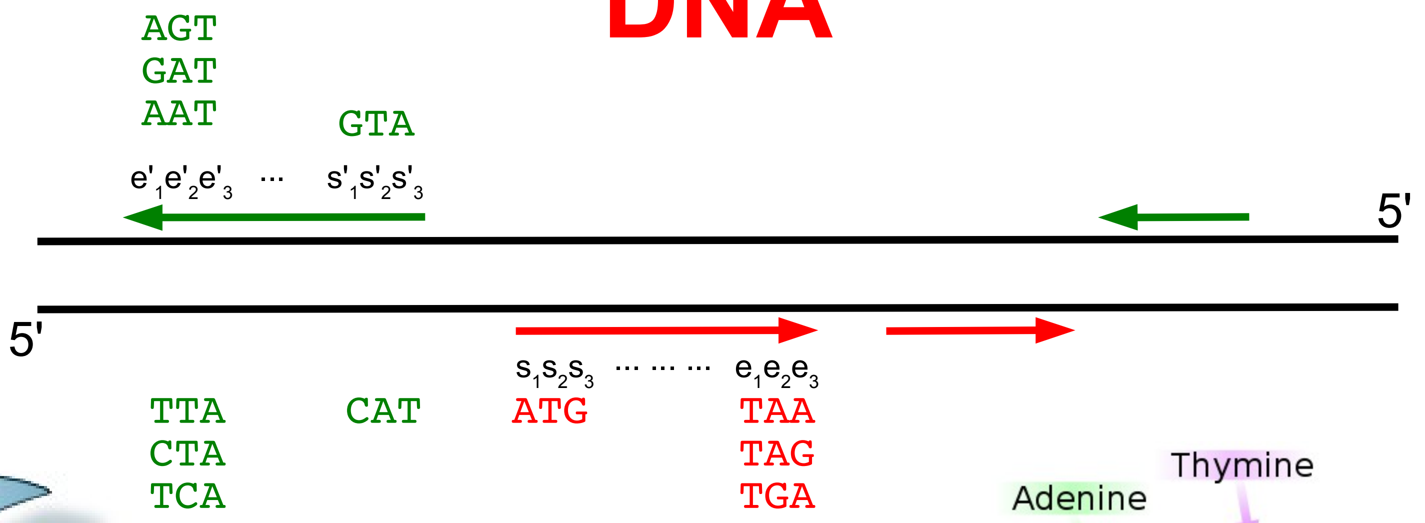


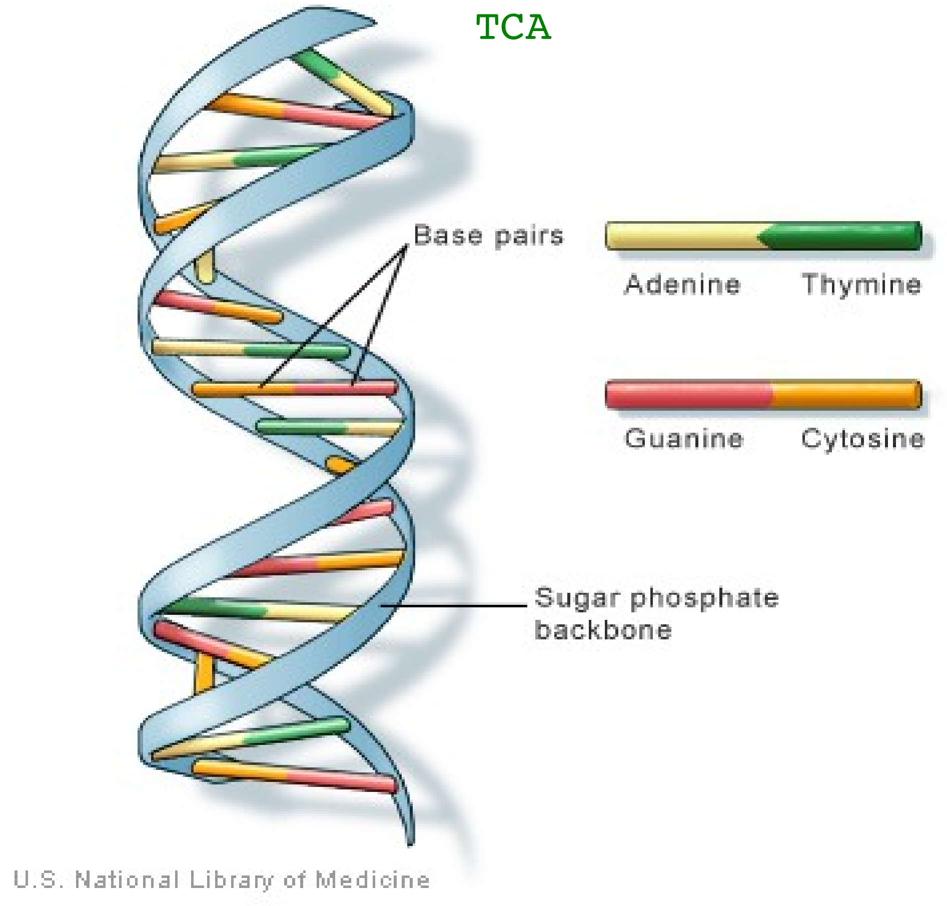
Base pairs
Adenine
Thymine
Guanine
Cytosine
Sugar phosphate backbone
U.S. National Library of Medicine

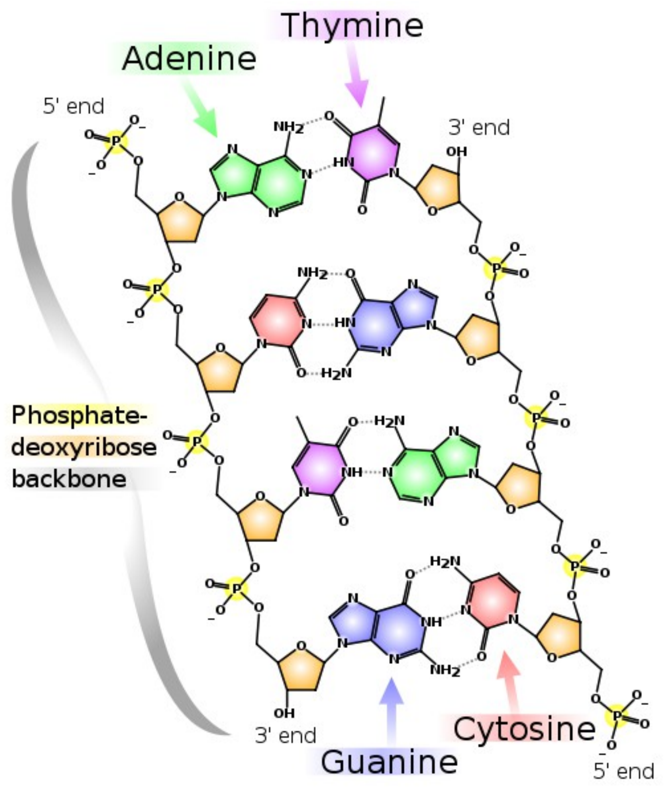


Thymine
Adenine
5' end
3' end
Phosphate-deoxyribose backbone
3' end
Cytosine
Guanine
5' end

DNA


AGT
GAT
AAT
GTA
$e'_1e'_2e'_3$ ··· $s'_1s'_2s'_3$
5'
5'
$s_1s_2s_3$ ··· ··· ··· $e_1e_2e_3$
TTA
CTA
TCA
CAT
ATG
TAA
TAG
TGA


Base pairs
Adenine
Thymine
Guanine
Cytosine
Sugar phosphate backbone
U.S. National Library of Medicine


Thymine
Adenine
5' end
3' end
Phosphate-deoxyribose backbone
3' end
Guanine
Cytosine
5' end

Analysis of the 11 genomes

```
Length of genome1: 1852441 (1852441)
Length of genome2: 2211485 (2211485)
Length of genome3: 2499279 (2499279)
Length of genome4: 1796846 (1796846)
Length of genome5: 2685015 (2685015)
Length of genome6: 2127839 (2127839)
Length of genome7: 2742531 (2742531)
Length of genome8: 2046115 (2046115)
Length of genome9: 2388435 (2388435)
Length of genome10: 1570485 (1570485)
Length of genome11: 2096309 (2096309)
```

```
Start-codon in normal genes:
ATG [8423, 'NCCC']
ATC [3, 'NCCC']
ATA [1, 'RCCC']
GTG [713, 'NCCC']
ATT [3, 'NCCC']
CTG [2, 'NCCC']
GTT [1, 'NCCC']
CTC [1, 'NCCC']
TTA [1, 'NCCC']
TTG [1020, 'NCCC']

Stop-codon in normal genes:
TAG [1949, 'CCCN']
TGA [1531, 'CCCN']
TAA [6686, 'CCCN']

Reversed stop-codon in reversed genes:
TTA (reverse-complement: TAA) [6596, 'NRRR']
CTA (reverse-complement: TAG) [2014, 'NRRR']
TCA (reverse-complement: TGA) [1148, 'NRRR']

Reversed start-codon in reversed genes:
TAT (reverse-complement: ATA) [2, 'RRRN']
ATG (reverse-complement: CAT) [1, 'RRRN']
GAT (reverse-complement: ATC) [1, 'RRRN']
CAT (reverse-complement: ATG) [8077, 'RRRN']
AAT (reverse-complement: ATT) [4, 'RRRN']
TAC (reverse-complement: GTA) [1, 'RRRN']
CAC (reverse-complement: GTG) [715, 'RRRN']
CAA (reverse-complement: TTG) [953, 'RRRN']
CAG (reverse-complement: CTG) [4, 'RRRN']
```

Analysis of the 11 genomes

```
Start-codon in normal genes:
ATG [8423, 'NCCC']
```

It might be an okay idea to ignore, i.e. not to model, the start- and stop-codons which are very rare ...

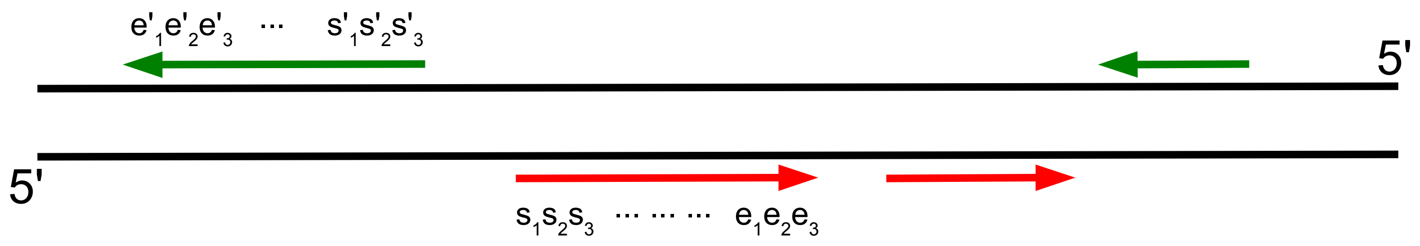
This of course means that there might be genes that you cannot predict ..

You also have to take care when “training by counting” because the data (**X**,**Z**) then contains annotations which are not possible in your model (you can e.g. skip start- and stop-codons which you do not model) ...

```
Leng
Leng
Leng
Leng
Leng
Leng
Leng
Leng
Leng
Length of genome10: [illegible]
Length of genome11: 2096309 (2096309)
```

```
Reversed start-codon in reversed genes:
TAT (reverse-complement: ATA) [2, 'RRRN']
ATG (reverse-complement: CAT) [1, 'RRRN']
GAT (reverse-complement: ATC) [1, 'RRRN']
CAT (reverse-complement: ATG) [8077, 'RRRN']
AAT (reverse-complement: ATT) [4, 'RRRN']
TAC (reverse-complement: GTA) [1, 'RRRN']
CAC (reverse-complement: GTG) [715, 'RRRN']
CAA (reverse-complement: TTG) [953, 'RRRN']
CAG (reverse-complement: CTG) [4, 'RRRN']
```

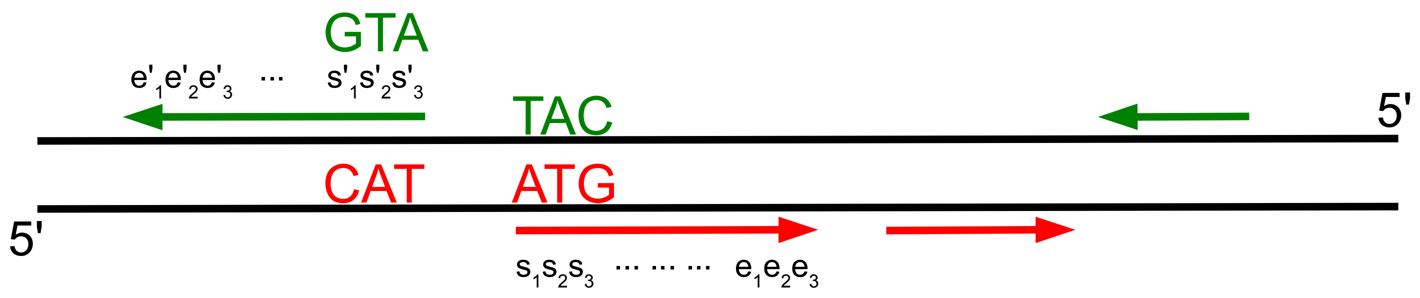
Analysis of the 11 genomes


$e'_1e'_2e'_3$ ··· $s'_1s'_2s'_3$
5'
5'
$s_1s_2s_3$ ··· ··· ··· $e_1e_2e_3$

The genes going from left-to-right (the C's):

- The first 3 symbols $s_1s_2s_3$ (the start-codon) is: **ATG**, ATC, ATA, **GTG**, ATT, CTG, GTT, CTC, TAA, **TTG**
- The last 3 symbols $e_1e_2e_3$ (the stop-codon) is: **TAG**, **TGA**, **TAA**
- The total length is a multiplum of 3.

Analysis of the 11 genomes



The genes going from left-to-right (the C's):

- The first 3 symbols $s_1s_2s_3$ (the start-codon) is: **ATG**, ATC, ATA, **GTG**, ATT, CTG, GTT, CTC, TAA, **TTG**
- The last 3 symbols $e_1e_2e_3$ (the stop-codon) is: **TAG**, **TGA**, **TAA**
- The total length is a multiplum of 3.

The genes going from right-to-left (the R's):

The first 3 symbols $s'_1s'_2s'_3$ (the reversed start-codon) is: TAT, ATG, GAT, **CAT**, AAT, TAC, **CAC**, **CAA**, CAG

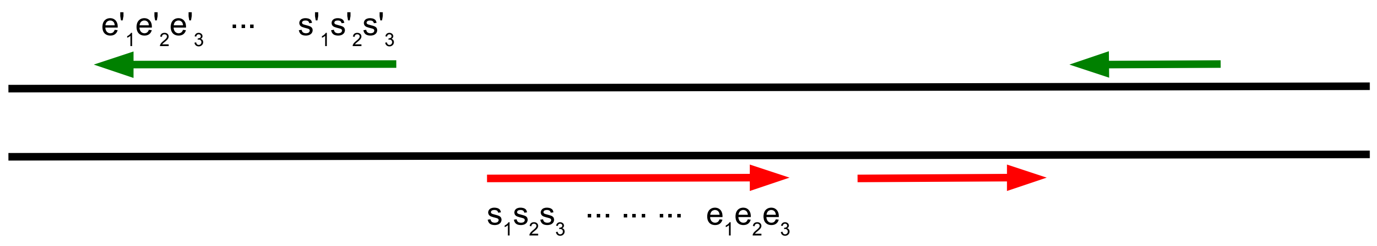
ATA, CAT, ATC, **ATG**, ATT, GTA, **GTG**, **TTG**, CTG

The last 3 symbols $e'_1e'_2e'_3$ (the reversed stop-codon) is: **TTA**, **CTA**, **TCA**

TAA, **TAG**, **TGA**

- The total length is a multiplum of 3.

Predicting the full gene structure

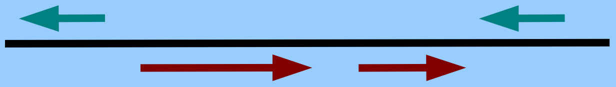

$e'_1e'_2e'_3$ ··· $s'_1s'_2s'_3$
$s_1s_2s_3$ ··· ··· ··· $e_1e_2e_3$

Approach for predicting the full gene structure (Cs and Rs)

- Predict C's and R's using two separate models, or the same model used for the genome (to predict C's) and its reverse complement (to predict R's).
- Afterwards we resolve conflicts, i.e. nucleotides predicted as both Cs and Rs.
- Or make a model which captures everything in one go.

Even more biology

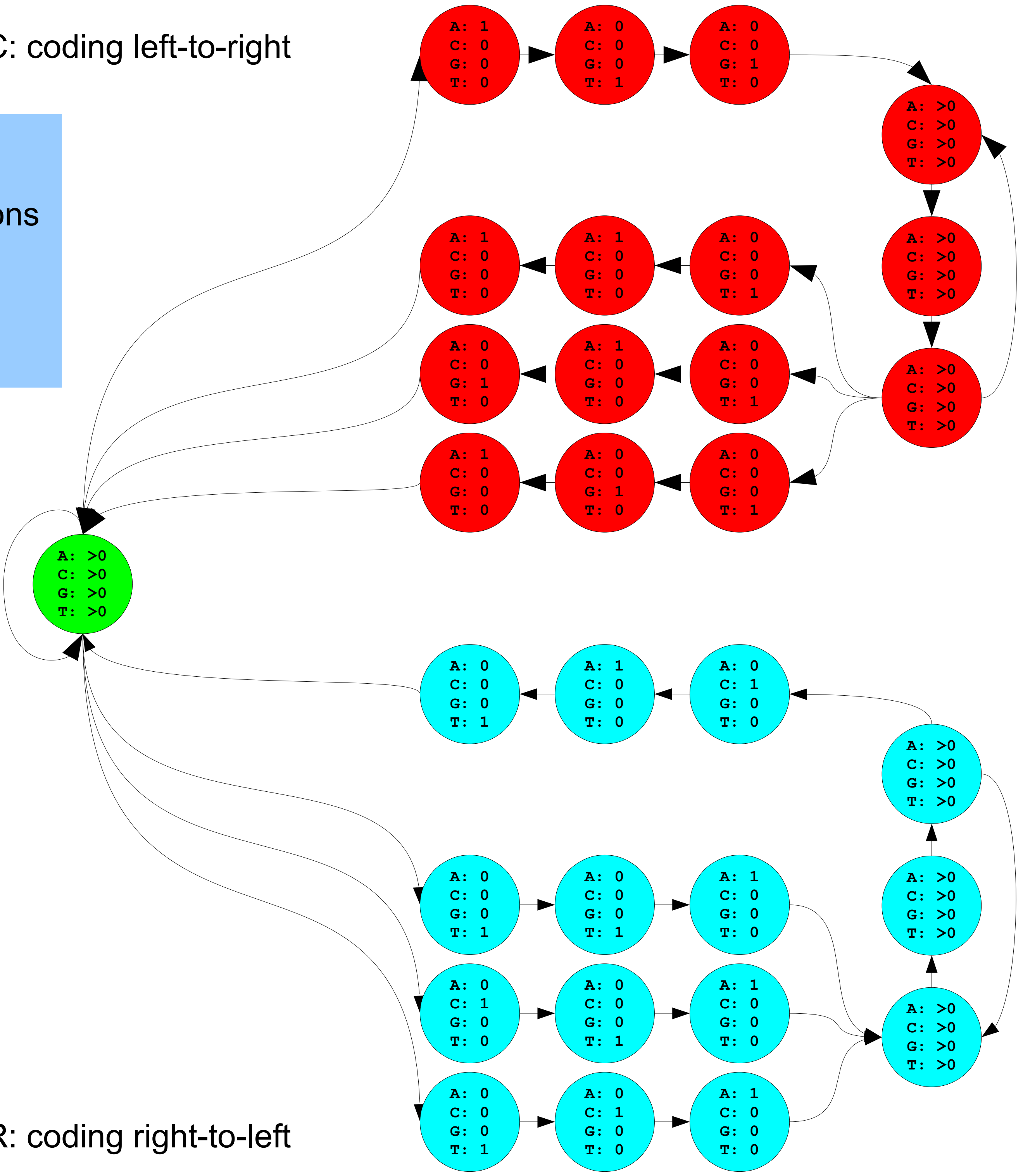
There can be genes in both directions



C: coding left-to-right

N: Non-coding

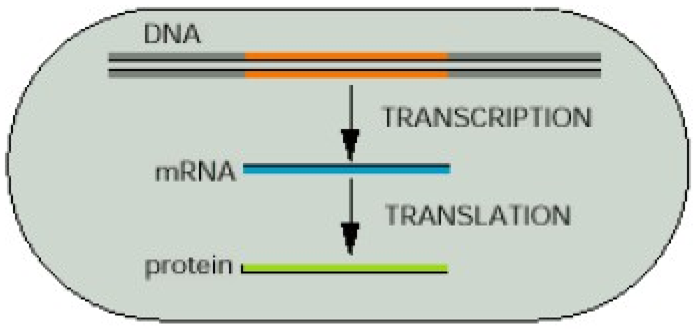
R: coding right-to-left



$\pi_N = 1$

$\pi_C = 0$

Problem: From annotation to Z

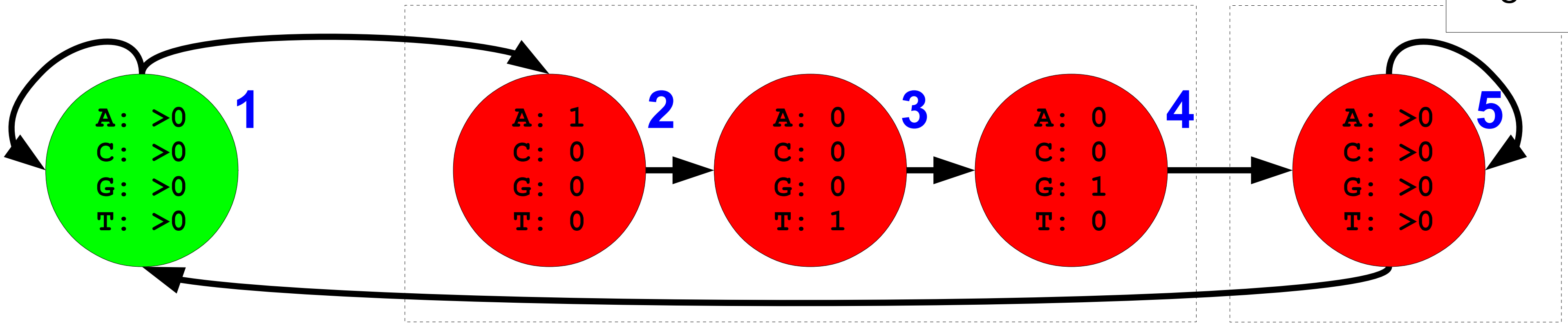


Problem: The string **Z=**NNNCCC.... is not a prober sequence of states in the illustrated HMM, but is can easily be converted into one.

Z: NNNCCCCCCCCCNNNNNNNNCCCCCCCCCCCCCCNNNNNNNNNNNN

X: acgatgcgctaatatgtccgatgacgtgagcataagcgacat

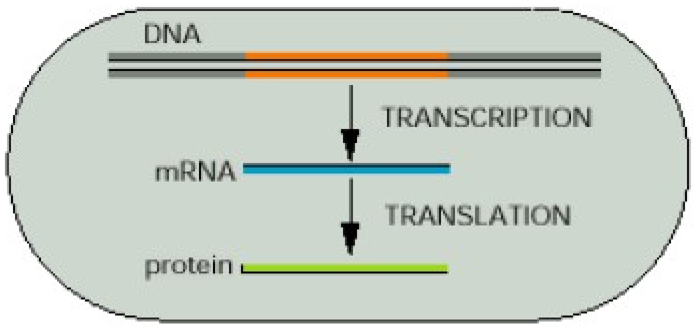
$\pi_N = 1$
$\pi_C = 0$



N: non-coding

C: coding

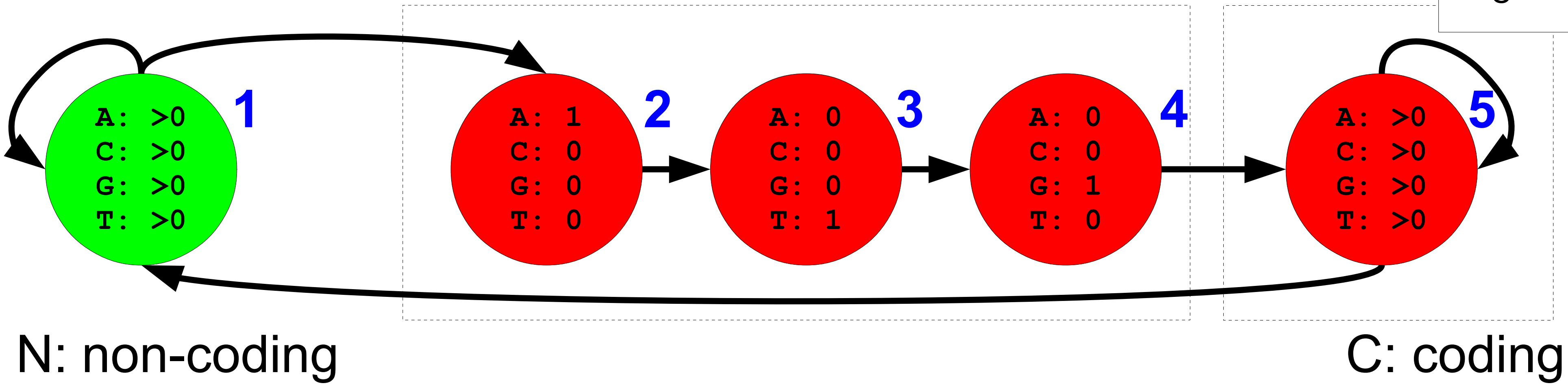
Problem: From annotation to Z



Problem: The string **Z=**NNNCCC.... is not a prober sequence of states in the illustrated HMM, but is can easily be converted into one.

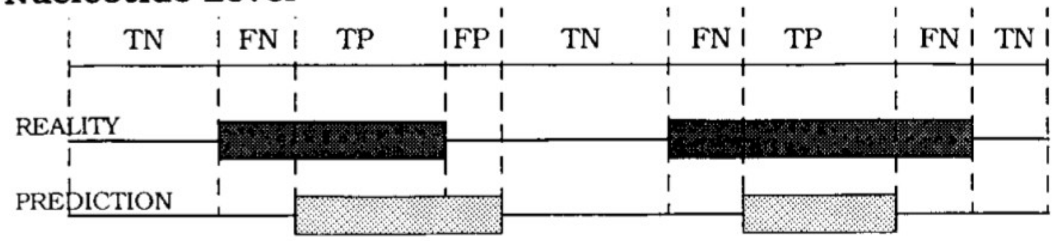
```
    1112345555551111111123455555555555511111111111
Z:  NNNCCCCCCCCCNNNNNNNNCCCCCCCCCCCCCCCNNNNNNNNNNN

X:  acgatgcgctaatatgtccgatgacgtgagcataagcgacatc
```

$\pi_N = 1$
$\pi_C = 0$



Evaluating performance

Nucleotide Level



REALITY

		coding	no coding	
PREDICTION	coding	TP	FP	TP+FP
	no coding	FN	TN	FN+TN
		TP+FN	TF+TN	

$$Sn = \frac{TP}{TP + FN}$$

Sensitivity

$$Sp = \frac{TP}{TP + FP}$$

Specificity

$$CC = \frac{(TP \times TN) - (FN \times FP)}{\sqrt{(TP + FN) \times (TN + FP) \times (TP + FP) \times (TN + FN)}}$$

Correlation Coefficient

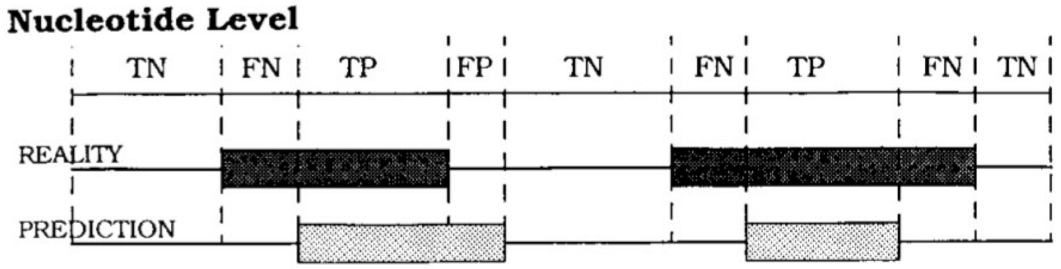
$$ACP = \frac{1}{4}\left[\frac{TP}{TP + FN} + \frac{TP}{TP + FP} + \frac{TN}{TN + FP} + \frac{TN}{TN + FN}\right]$$

$$AC = (ACP - 0.5) \times 2$$

Approximate Correlation

Evaluation of Gene Structure Prediction Programs (Burset and Guigo, 1996)

Evaluating performance



```
def print_stats(tp, fp, tn, fn):
    sn = float(tp) / (tp + fn)
    sp = float(tp) / (tp + fp)
    cc = float((tp*tn - fp*fn)) /
           math.sqrt(float((tp+fn)*(tn+fp)*(tp+fp)*(tn+fn)))
    acp = 0.25 * (float(tp)/(tp+fn) + float(tp)/(tp+fp) +
                  float(tn)/(tn+fp) + float(tn)/(tn+fn))
    ac = (acp - 0.5) * 2
    print("Sn = %.4f, Sp = %.4f, CC = %.4f, AC = %.4f" % (sn, sp, cc, ac))
```

$$CC = \frac{(TP \times TN) - (FN \times FP)}{\sqrt{(TP+FN) \times (TN+FP) \times (TP+FP) \times (TN+FN)}}$$ **Correlation Coefficient**

$$ACP = \frac{1}{4}\left[\frac{TP}{TP+FN} + \frac{TP}{TP+FP} + \frac{TN}{TN+FP} + \frac{TN}{TN+FN}\right]$$

$$AC = (ACP - 0.5) \times 2$$ **Approximate Correlation**

Evaluation of Gene Structure Prediction Programs (Burset and Guigo, 1996)

Counting C's

```
def count_c(true, pred):
    total = tp = fp = tn = fn = 0
    for i in range(len(true)):
        if pred[i] == 'C' or pred[i] == 'c':
            total = total + 1
            if true[i] == 'C' or true[i] == 'c':
                tp = tp + 1
            else:
                fp = fp + 1
        if pred[i] == 'N' or pred[i] == 'n':
            if true[i] == 'N' or true[i] == 'n' or true[i] == 'R' or true[i] == 'r':
                tn = tn + 1
            else:
                fn = fn + 1
    return(total, tp, fp, tn, fn)
```

Counting C's

```
def count_c(true, pred):
    total = tp = fp = tn = fn = 0
    for i in range(len(true)):
        if pred[i] == 'C' or pred[i] == 'c':
            total = total + 1
            if true[i] == 'C' or true[i] == 'c':
                tp = tp + 1
            else:
                fp = fp + 1
        if pred[i] == 'N' or pred[i] == 'n':
            if true[i] == 'N' or true[i] == 'n' or true[i] == 'R' or true[i] == 'r':
                tn = tn + 1
            else:
                fn = fn + 1
    return(total, tp, fp, tn, fn)
```

Counting R's

```
def count_r(true, pred):
    total = tp = fp = tn = fn = 0
    for i in range(len(true)):
        if pred[i] == 'R' or pred[i] == 'r':
            total = total + 1
            if true[i] == 'R' or true[i] == 'r':
                tp = tp + 1
            else:
                fp = fp + 1
        if pred[i] == 'N' or pred[i] == 'n':
            if true[i] == 'N' or true[i] == 'n' or true[i] == 'C' or true[i] == 'c':
                tn = tn + 1
            else:
                fn = fn + 1
    return(total, tp, fp, tn, fn)
```

Counting R's

```
def count_r(true, pred):
    total = tp = fp = tn = fn = 0
    for i in range(len(true)):
        if pred[i] == 'R' or pred[i] == 'r':
            total = total + 1
            if true[i] == 'R' or true[i] == 'r':
                tp = tp + 1
            else:
                fp = fp + 1
        if pred[i] == 'N' or pred[i] == 'n':
            if true[i] == 'N' or true[i] == 'n' or true[i] == 'C' or true[i] == 'c':
                tn = tn + 1
            else:
                fn = fn + 1
    return(total, tp, fp, tn, fn)
```

Counting C's and R's

```
def count_cr(true, pred):
    total = tp = fp = tn = fn = 0
    for i in range(len(true)):
        if pred[i] == 'C' or pred[i] == 'c' or pred[i] == 'R' or pred[i] == 'r':
            total = total + 1
            if (pred[i] == 'C' or pred[i] == 'c') and (true[i] == 'C' or true[i] == 'c'):
                tp = tp + 1
            elif (pred[i] == 'R' or pred[i] == 'r') and (true[i] == 'R' or true[i] == 'r'):
                tp = tp + 1
            else:
                fp = fp + 1
        if pred[i] == 'N' or pred[i] == 'n':
            if true[i] == 'N' or true[i] == 'n':
                tn = tn + 1
            else:
                fn = fn + 1
    return(total, tp, fp, tn, fn)
```