FastA

An Alternative Strategy For Sequence Alignment

FastA

- Faster than quadratic time
- Alignment isn't always optimal
- Hybrid of dot plot, S-W and N-W strategies

- Strategy

  - Preprocess 2 sequences to determine the most likely offset between them. This essentially a local alignment extending the length of the shorter sequence

  - Use that info to define a restricted alignment space and employ a global alignment within that space

FastA

Preprocessing step-CONCEPTUAL
Runs in **QUADRATIC** time

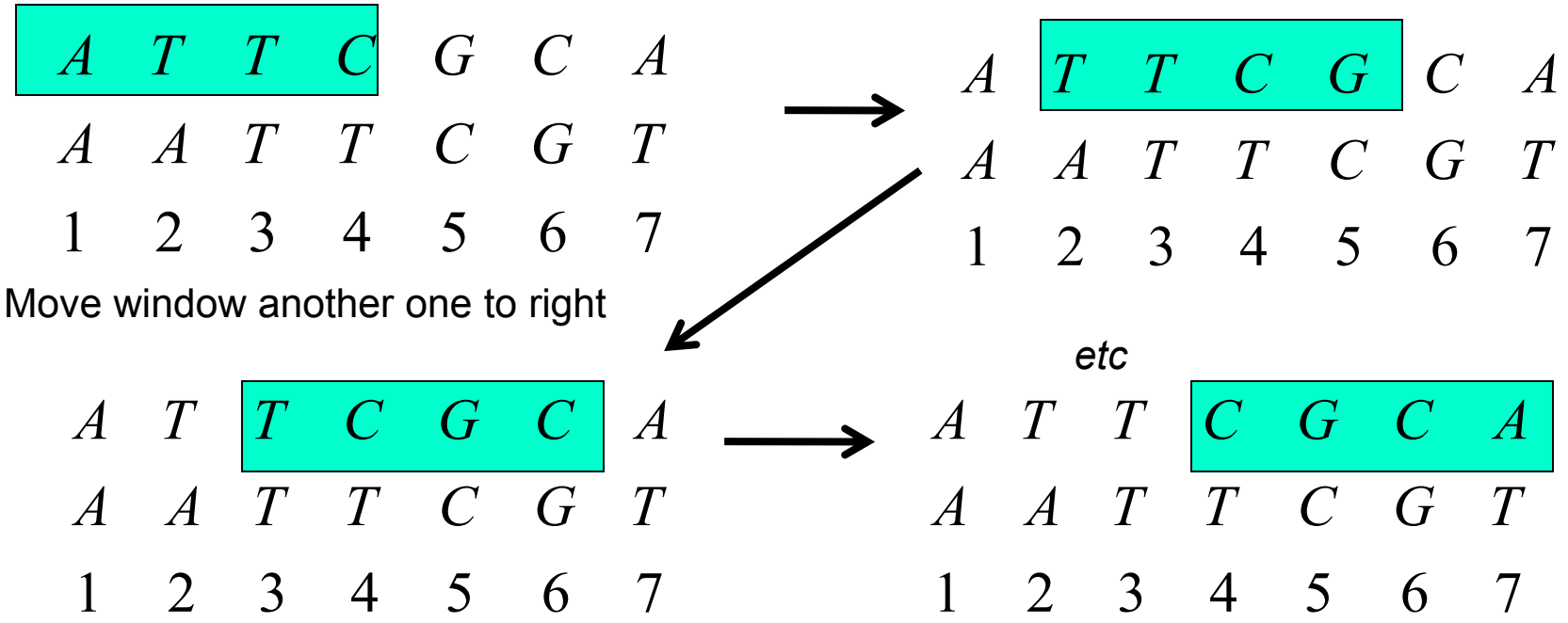
- Pick an arbitrary window size, say, 4 nucleic acid bases (4-mer)
- Note the location of all 4-mers in both the target and the query.
- Compute the most frequent offset distance between corresponding target and query 4-mers

FastA
Preprocessing step-ACTUAL

- There are 256 4-mers of nucleotides (4^4). Create a table and use the 4-mer to index the table
- Slide the 4-wide window along both the query and the target. At each 1 base increment, open up the table cell for the 4-mer in the window and record
  - Whether it came from query or target
  - The increment (*ie* the starting position of the 4-mer in the string)

Start with query in 1st position

Move 4-mer window in the query one to the right



Now start with the target 1st position and do the same thing.

- This process is of the order len(query)+len(target); it is linear!
- The 4-mer itself can be used as an index or a simple hash function can be contrived

Here is the resulting table with the starting offsets in both the query and target indicated for each 4-mer

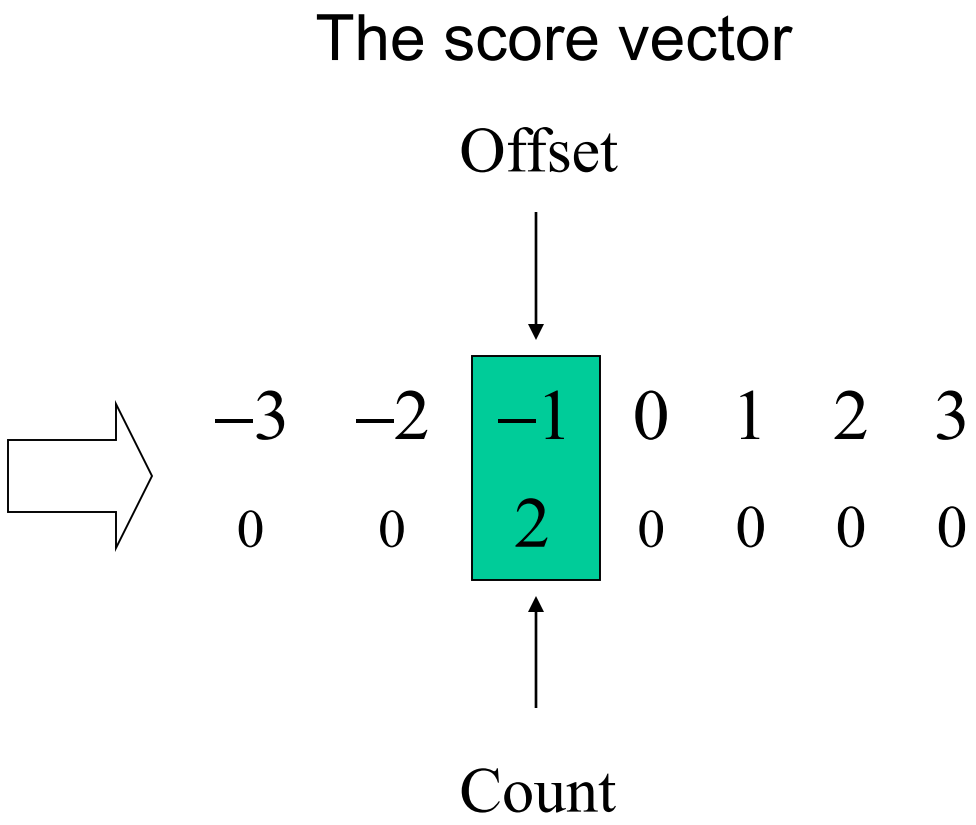
AAAA	
AAAC	
…	
AATT	t:1
ATTC	q:1 t:2
…	
CGCA	q:4
..	
TTCG	q:2 t:3
TCGC	q:3
TCGT	t:4

FastA
Preprocessing step-ACTUAL

- Create a vector of cells, one for each possible offset between the query and the target. Initialize to 0.
- Walk the table. For every entry with a q, if there is a t or t's, compute all the differences (q's-t's) and increment the cells in the score vector indexed by each of those differences. The direction of the distance is critical.
- Select the highest scoring value in the score vector. Its index (may be + or -) tells what the most likely global alignment is, with respect to the main diagonal of the dynamic programming matrix.

This is a catalog of lags

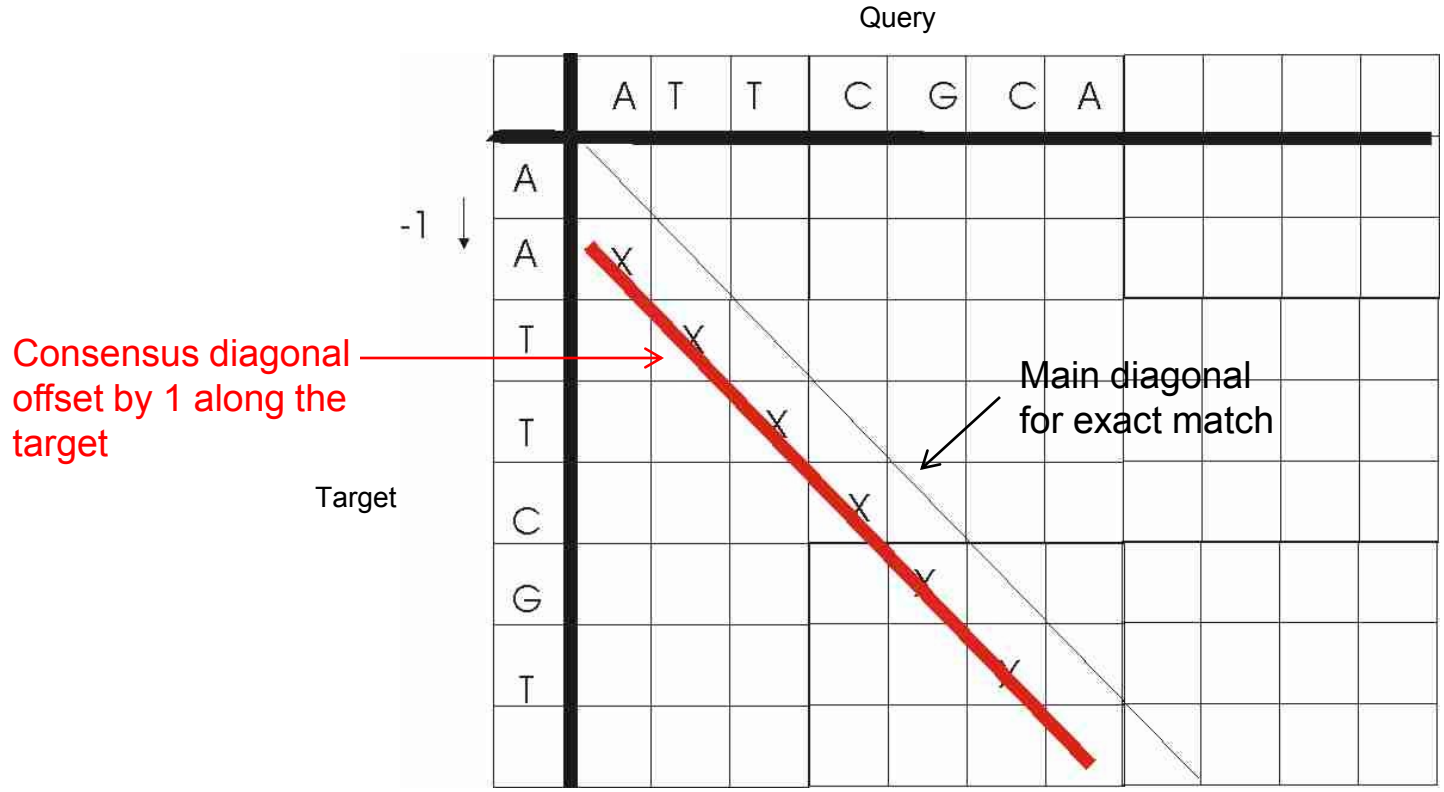
AAAA			
AATT		t:1	
ATTC	q:1	t:2	**-1**
…			
CGCA	q:4		
..			
TTCG	q:2	t:3	**-1**
TCGC	q:3		
TCGT		t:4	



FastA

Processing step

The dynamic programming matrix will then have only one diagonal, offset from the main diagonal by -1.

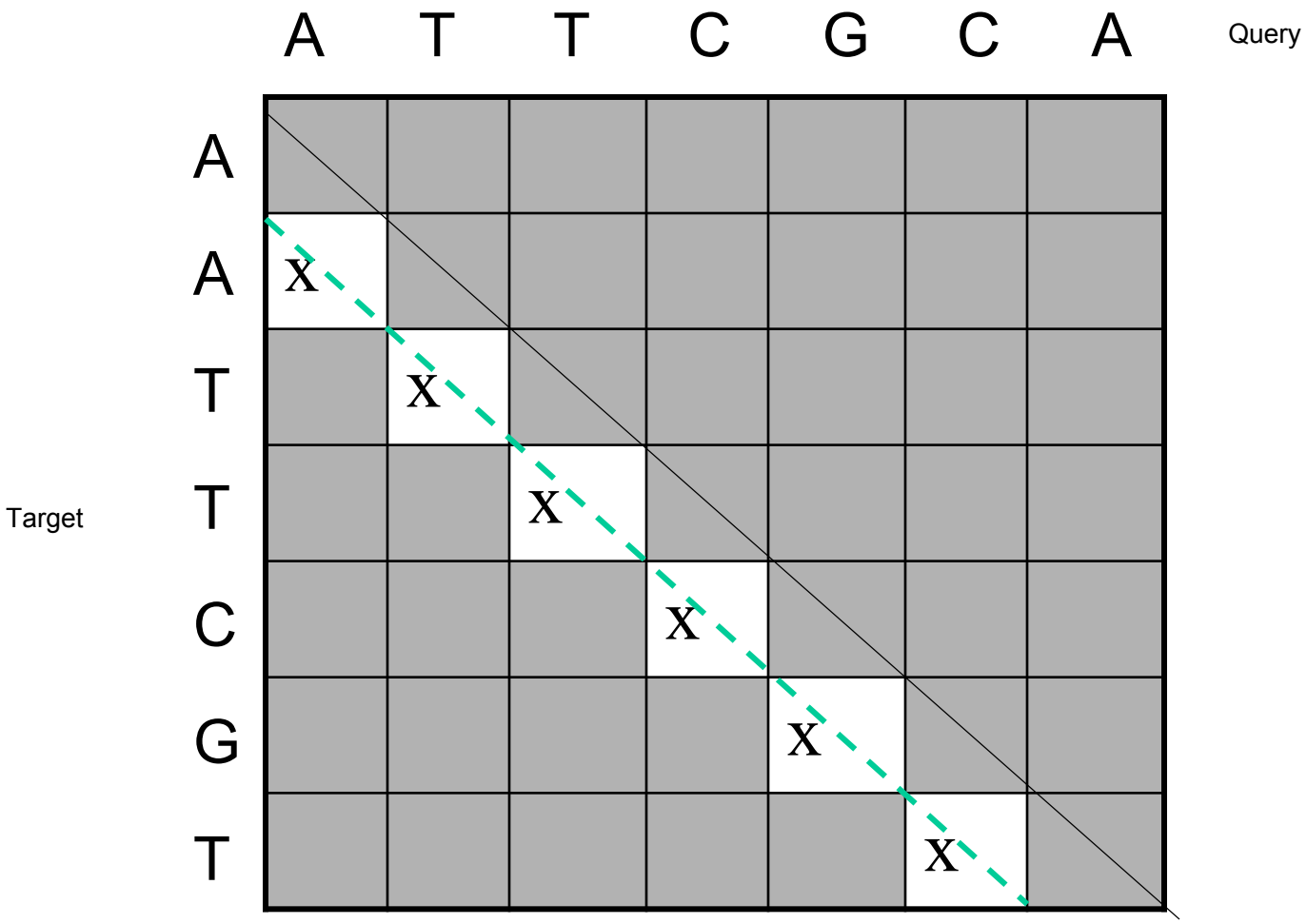


FastA

Processing step

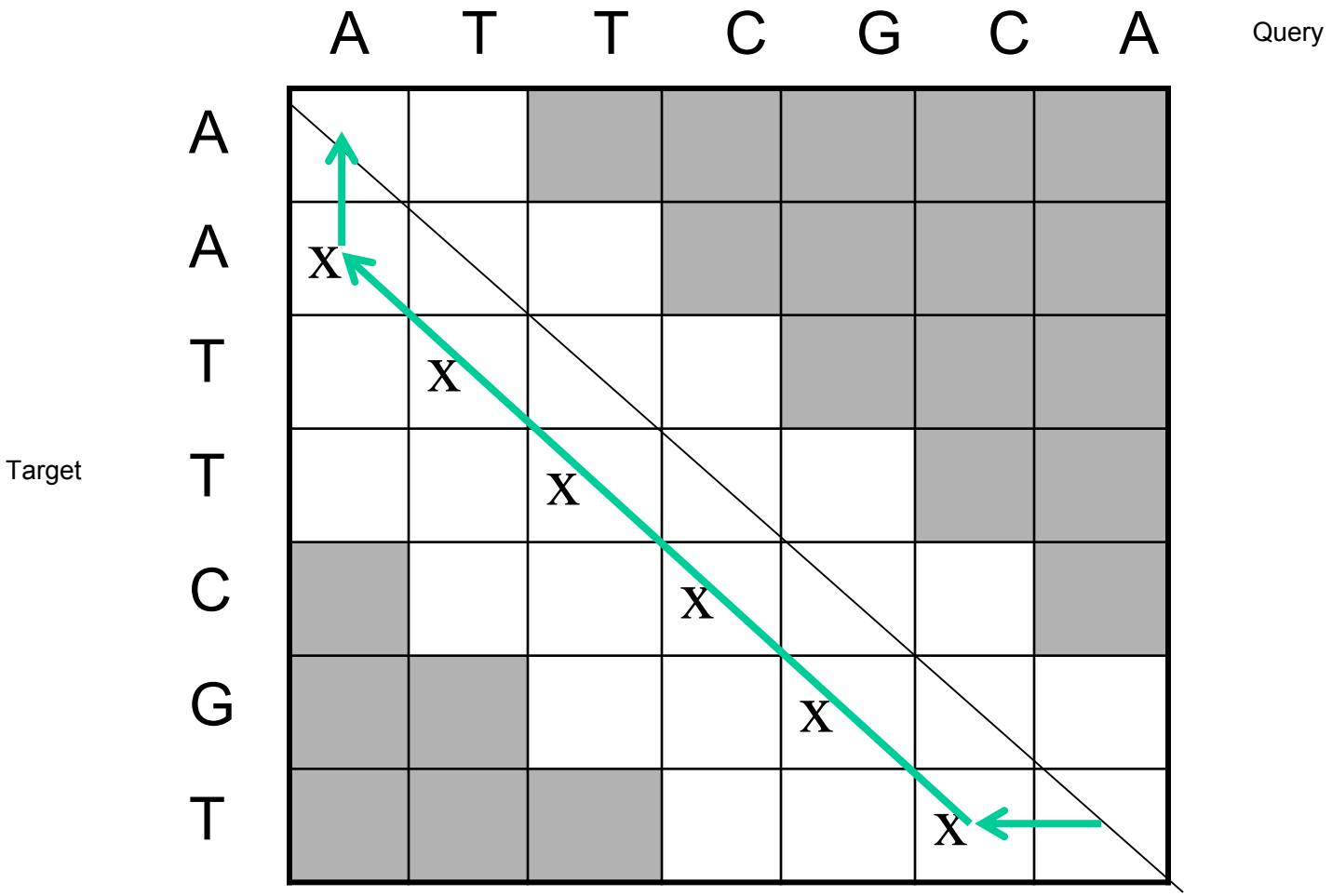
The cells that are shown grey here have infinitely negative values. This forms a ‘channel’ around the offset diagonal. No dynamic programming can proceed.

If there are to be no gaps, we are done. We have a local alignment.



FastA

But…If we *widen* the channel, then the dynamic programming can operate around the cells of the offset diagonal, inserting gaps as needed. The wider the open cells, the more dynamic programming is required. Our local alignment is thus improved, and in this case is a global alignment



FastA
Complexity

- The implementation of the conceptual idea of sliding would be $\mathcal{O}(n^2)$ where n is the length of the longer sequence
- The use of the table lookup in FastA reduces the complexity to $\mathcal{O}(n)$ for an unbiased table
- If the channel around the diagonal for dynamic programming is opened, then the dynamic programming costs increase accordingly