# Computational tricks for tied score detection in global pairwise sequence alignment algorithms

Detecting ties in global pairwise sequence alignments is not an interesting or important problem, but here are some notes regarding some computational considerations.

The computational concerns about equality detection are completely uninteresting from a mathematics research perspective. Even from an applied computational perspective, equality detection for the kinds of expressions mentioned in this note has already been implemented by widely available computer algebra systems.

All schemes considered here are assumed to score a partial pairwise sequence alignment according to a finite dimensional vector of 'move type' counts. The finite set of 'move types' depends on details of the scoring algorithm. For example the set of move types for a basic scheme could be {gap, mismatch, match}, or for more complicated schemes it could be more exotic like {Seq1GapInit, Seq1GapExtend, matchAA, mismatchAG, . . . }.

In this framework the sufficiency of the count vector for computing the score immediately gives a condition indicating a tie: if two partial alignments share the same count vector then they must also share the same score. On the other hand, two different count vectors may sometimes also share the same score – this mathematical coincidence is annoying if you are trying to determine which partial alignment should be preferred, and the following notes attempt to explain the nature of these coincidences and how they can be detected efficiently within the context of a dynamic programming alignment algorithm.

## Examples and analyses under various scoring schemes

The following examples assume that we have a scoring function with a set of $k = 3$ move types and that we want to evaluate a partial alignment whose move type count vector is $n = (n_1, n_2, n_3)^T$.

### Additive integer scores

In this simple scoring scheme, move type $i$ is associated with an integer value $a_i$ and the partial alignment score is $\sum_i^k a_i n_i$. Tie detection is simple in this case because you can compare the integer scores directly.

As an example, let the vector of move type values be $a = (1, 2, 3)^T$. If we let two different partial alignments have move type counts $u = (1, 1, 0)^T$ and $v = (0, 0, 1)^T$, then the partial alignments both have score 3. Similarly, if two different partial alignments have move type count vectors $u = (5, 0, 0)^T$ and $v = (0, 1, 1)^T$ then the partial alignments will both have score 5.

## Multiplicative rational number scores

This scoring scheme is nearly as simple. If move type $i$ is associated with a rational number $b_i$ then the partial alignment score for move type count vector $n$ is $\prod_i^k {b_i}^{n_i}$. The logarithm of the score is $\sum_i^k n_i \log b_i$. As an example, let $b = (10/3, 4/5, 8/3)^T$ be the vector of move type values. Continuing the example, if two different partial alignments have move type counts $u = (1, 1, 0)^T$ and $v = (0, 0, 1)^T$, then the partial alignments both have score 8/3. On the other hand, if two different partial alignments have move type count vectors $u = (5, 0, 0)^T$ and $v = (0, 1, 1)^T$, then the partial alignment scores will differ – the score of $u$ would be $(10/3)^5 = 100000/243$ and the score of $v$ would be $(4/5) * (8/3) = 32/15$.

As was the case for the additive scoring scheme, the rational number scores could also be directly compared. If a naive internal representation of these scores is used, for example storing the numerator and denominator of the reduced fraction using base-2 integers, then the comparison time and storage requirement will grow at least linearly with the length of the partial alignment, which is undesirably slow.

It is possible to do better by representing and comparing the partial scores in 'log space'. We want to enable exact score comparisons, so we do not use floating point representations which approximate rational numbers as integer multiples of powers of 2. Instead, the partial score can be represented by a vector of integer powers of pairwise coprime factors appearing in the numerators and denominators of $b$. Continuing the example above, the factors $\{2, 3, 5\}$ could be used, and the entrywise logarithm of $b$ could be written as

$$\begin{pmatrix} \log b_1 \\ \log b_2 \\ \log b_3 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 2 & 0 & -1 \\ 3 & -1 & 0 \end{pmatrix} \begin{pmatrix} \log 2 \\ \log 3 \\ \log 5 \end{pmatrix}.$$

and the score would be

$$\exp\left( \begin{pmatrix} n_1 & n_2 & n_3 \end{pmatrix} \begin{pmatrix} 1 & -1 & 1 \\ 2 & 0 & -1 \\ 3 & -1 & 0 \end{pmatrix} \begin{pmatrix} \log 2 \\ \log 3 \\ \log 5 \end{pmatrix} \right).$$

Note that the score depends on the integer count vector $n$ only through the transformed vector $n^T G$ where

$$G = \begin{pmatrix} 1 & -1 & 1 \\ 2 & 0 & -1 \\ 3 & -1 & 0 \end{pmatrix}.$$

Because $\{2, 3, 5\}$ are pairwise coprime, two count vectors will give the same scores if and only if the transformed vectors are the same. In the first example above with count vectors $u = (1, 1, 0)^T$ and $v = (0, 0, 1)^T$, the transformed vectors $u^T G$ and $v^T G$ are both $(3, -1, 0)^T$. In the second example above with count vectors $u = (5, 0, 0)^T$ and $v = (0, 1, 1)^T$, the transformed vectors are $u^T G = (5, -5, 5)$ and $v^T G = (5, -1, -1)$.

In less abstract terms, the idea is to track a vector of integer powers of pairwise coprime factors instead of tracking the vector of move type counts. Instead of incrementing the $i$th entry of the count vector when move type $i$ is encountered, row $i$ of $G$ is added (in the sense of vector addition) to the vector that tracks the integer powers of factors. For detecting tied scores, the benefit of tracking integer powers instead of move type counts is that two power vectors are identical if and only if the scores are identical, whereas it is possible (as seen in the example above) that two move type count vectors may be different from each other while still sharing the same score evaluation.

### a digression on optional computational tricks

In closing this section on multiplicative rational number scores, I'll mention a couple small computational tricks that can be used in combination with the idea of transforming the count vector.

The first trick involves noticing that although computing the prime factorization of the numerators and denominators of $b$ may be the most obvious way to compute a pairwise coprime factorization, it is not always necessary. Any pairwise coprime factorization is sufficent, and this factorization can be computed by "factor refinement" more efficiently than by computing the prime factorization. As an extreme example of the efficiency improvement, if someone concocts a scoring scheme that involves a large integer like RSA-1024 then a method based on prime factorization will fall down that rabbit hole whereas a method based on factor refinement would compute a coprime factorization without trying to solve the RSA challenge.

The second trick involves decomposing $G$ to yield a transformation with the desirable properties mentioned above but without having to track as many integers. The number of entries in the transformed vector explained earlier in this section is equal to the number of columns of $G$, but linear algebra tricks can be used to find a different transformation with the same desirable properties and which tracks a vector with only $\text{rank}\,(G)$ entries. This trick involves a matrix decomposition of $G$ which needs to be computed only once. For the example in this section, a vector that tracks only $\text{rank}\,(G) = 2$ integers instead of 3 integers could be used, and it would involve a decomposition like

$$G = \begin{pmatrix} 1 & -1 \\ 2 & -1 \\ 3 & -2 \end{pmatrix} \begin{pmatrix} 1 & 1 & -2 \\ 0 & 2 & -3 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 2 & 0 & -1 \\ 3 & -1 & 0 \end{pmatrix}.$$

## Multiplicative univariate rational function scores

The algebraic structure of this scoring scheme is similar to that of multiplicative rational number scoring. If move type $i$ is associated with a univariate rational function $c_i$ then the partial alignment score for move type count vector $n$ is $\prod_i^k {c_i}^{n_i}$. We will reuse the structure of the example from the previous section, but instead of prime integer factors $\{2, 3, 5\}$ we will use irreducible univariate polynomial factors $\{x, x-1, x+1\}$. Following that analogy, for this example we have

$$c = \left( \frac{x\,(x+1)}{x-1}, \frac{x^2}{x+1}, \frac{x^3}{x-1} \right)^T .$$

Because the structure is like that of multiplicative rational number scoring I think that similar tricks apply, but a practical complication is that software library support for rational functions is less readily available than for rational numbers. Just for comparison with the previous section, here is the log space form:

$$\begin{pmatrix} \log c_1 \\ \log c_2 \\ \log c_3 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 2 & 0 & -1 \\ 3 & -1 & 0 \end{pmatrix} \begin{pmatrix} \log x \\ \log(x-1) \\ \log(x+1) \end{pmatrix}$$

The scoring method in this section may seem strange because the score is a function of an apparently useless placeholder value $x$, but nevertheless it is meaningful to detect equality of two rational functions. In the next section we look at what happens when a particular value of $x$ is chosen, and we see that for a certain class of choices the equality of two rational functions is equivalent to the equality of the evaluations of the functions at that point.

## Multiplicative univariate rational functions evaluated at transcendental points

It may be clear that comparing two rational functions for equality is not equivalent to comparing the equality of the evaluations of the two rational functions at a given point. For example $x$ and $x^2$ are distinct polynomials with integer coefficients but they both evaluate to 1 at the point $x = 1$.

On the other hand, this kind of coincidence cannot occur when rational functions are evaluated at a transcendental point. Therefore if the scoring scheme specifies that the multiplicative univariate rational functions are to be evaluated only at transcendental points, then any technique for detecting equality of rational functions 'goes through' for detecting equality of transcendental evaluations of those functions.

Continuing the example from the previous section, if we let $x$ take the transcendental value $e^{1/5}$ then we have

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} \frac{e^{1/5}\left(e^{1/5}+1\right)}{e^{1/5}-1} \\ \frac{e^{2/5}}{e^{1/5}+1} \\ \frac{e^{3/5}}{e^{1/5}-1} \end{pmatrix} .$$

Even for such complicated-looking scoring schemes, I think that with a suitable library for manipulating (large sparse) univariate polynomials with integer coefficients it would be possible to precompute a basis that allows an integer vector to be tracked efficiently in a way that makes tied score detection equivalent to detecting equality of integer vectors. This particular example would not require working with large sparse polynomials, but I have in mind an application that would require working with polynomials of degree $\operatorname{lcm}\{23^2, 24^2, 26^2, 27^2\}$ and with fewer than a dozen nonzero coefficients.

Without such an implementation, univariate rational functions can be partially supported by treating their numerators and denominators analogously to prime integers within the framework of multiplicative rational number scoring. This gives an incomplete tie detection method whose ignorance of polynomial factors shared between polynomials may cause it to miss ties. But maybe this detection method would miss fewer ties than the method that directly compares move type count vectors.