Compact spectrum representation

When a whole spectrum needs to be retrieved and processed, it is useful to treat it as a bundle of data as opposed to a collection of data items (i.e. peaks). The following assumptions apply to an individual spectrum: (1) the range of *x*-values is known, and (2) all *x*-values are recorded with the same precision *p* (the number of digits to be recorded after the decimal point). The compact format takes advantage of the fact that all *x*-values can be generated from the initial value using 10^{-p} as an increment: $x_\mathrm{n} = x_0 + n \cdot 10^{-p}$, and therefore need not be explicitly stored. Instead, only the initial value and the increment are stored, while the *y*-values are stored in correspondence to the ascending order of *x*-values. In addition, when a spectrum is sparse (e.g. mass spectra obtained through GCMS), instead of storing the *NULL* values explicitly, they can simply be skipped by recording the number of consecutive *NULL* values. We define a compact spectrum format by the following regular expression:

(<separator> <value>)$^+$ <separator>

where <value> is a numerical value and <separator> stands for any of the following characters: |, \#, + and -. The separators indicate how the *y*-values can be constructed from the given numerical values. The vertical bar (|) is used to separate the actual *y*-value from the previously stored value. For n consecutive *NULL* values, the hash sign (#) is used as a separator followed by *n*. The plus (+) and minus (-) signs are used when the difference between the *y*-value to be stored and the last stored *y*-value is shorter (in the number of digits) than the actual *y*-value to be stored. In such case, + or - are followed by the absolute value of the difference (so that additional space can be saved) and they indicate that the current *y*-value can be obtained by adding or subtracting the current value from the last recorded *y*-value. When decimal numbers are used, they should be multiplied by 10^p (where *p* is the number of digits recorded after the decimal point) in order to remove the decimal point and, more importantly, the leading zeros, thus reducing the number of digits to be stored. Additional space can be saved by recording the numbers using a bigger base (e.g. 16 to record *x*- and *y*-values as hexadecimal numbers), since fewer characters (i.e. bytes) are needed to represent individual numbers.

We use an example to illustrate the compact format. Given a mass spectrum:

x	y		
30	*NULL*	→	#1
31	1020	→	1020
32	2847	→	2847
33	*NULL*	} →	#6
…	…		
38	*NULL*		
39	3632	→	+785

…	…		…
271	1661	→	1661
272	1614	→	–47
273	402	→	402
274	*NULL*	} →	#327
…	…		
600	*NULL*		

where *m/z* values are integers (which implies that the increment to be used is 1) ranging from 30 to 600, the compact representation is as follows:

#1 | 1020 | 2847#6+785 … | 1661-47 | 402#327 |

A more compact representation is obtained using 36 as a base:

#1 | SC | 273#6+LT … | 1A5-1B | B6#93 |

Let us contrast our compact spectrum representation with the binary representation, which is similarly used to save space when storing spectra. For example, the mzData (http://psidev.sourceforge.net/ms/) format supports storing mass spectra in the binary format. The *x*- and *y*-values are stored separately as byte arrays for the recorded peaks. Depending on the precision (single or double), 4 or 8 bytes are used for each *x*- or *y*-value, which is 8, 12 or 16 bytes per peak. In this format, the number of bytes used per peak is constant, while in our format it varies depending on the number of digits (each represented as an ASCII character) in the y-value of the peak. Therefore, none of the formats generally outperforms the other in terms of the space requirements. In general, it is expected that the binary format will do better when the *y*-values are represented as floating point numbers with high precision. Therefore, both formats should be used sensibly, depending on the nature of the data and specific applications.