Supplementary Information:

0.1 Formal proof regarding Hamming Distance Property

Let hamming distance between strings x and y be $d(x, y)$. Assuming both x and y are composed of n characters, then hamming distance is formally defined as [2]:

$$d(x, y) = \Sigma_{i=0}^{n} neq(x_i, y_i) \tag{10}$$

where, if a and b are two characters,

$$neq(a, b) = \begin{cases} 0, & \text{if } a = b \\ 1, & \text{otherwise} \end{cases} \tag{11}$$

Property: Given, there are two strings x and y (composed of n characters each) and characters from p positions are removed to obtain strings x' and y' with $(n - p)$ characters. If the hamming distance between x' and y', $d(x', y') = 0$ then the hamming distance between original x and y, $d(x, y) \leq p$.

Proof by example:

Let $p = 2$. We first re-write Eq. 10 as:

$$d(x, y) = \Sigma_{i=0}^{n-2} neq(x_i, y_i) + neq(x_{n-1}, y_{n-1}) + neq(x_n, y_n) \tag{12}$$

That is, we split the summation of $neq(.)$ function as summation of $neq(.)$ for $(n-2)$ characters plus the sum of $neq(.)$ for the $(n-1)^{th}$ and last n^{th} character for x and y.

The term $\Sigma_{i=0}^{n-2} neq(x_i, y_i)$ represents the hamming distance $d(x', y')$ for $p = 2$ positions removed. Therefore:

$$d(x, y) = d(x', y') + neq(x_{n-1}, y_{n-1}) + neq(x_n, y_n) \tag{13}$$

Now if $d(x', y') = 0$ then

$$d(x, y) = neq(x_{n-1}, y_{n-1}) + neq(x_n, y_n) \tag{14}$$

Based on Eq. 11, $d(x, y) = \{(0 + 0), (0 + 1), (1 + 0), (1 + 1)\}$ as these are all the possible values of $neq(.)$ function.

Therefore, $d(x, y) \leq 2$ if $d(x', y') = 0$ where x' and y' are x and y (respectively) with characters removed from $p = 2$ positions.

0.2 Time profiles for different functions of GaKCo

Figure 8 shows the GaKCo average time profiled for sorting vs. count and update function when calculating cumulative mismatch profile for EP300 DNA dataset ($m = 7$).

0.3 AUC scores for best performing parameters

Different handling of dictionaries Table 3 summarizes the AUC scores for all datasets. The current gkSVM implementation [11] while reading the input ignores an unknown character. GaKCo maps it to another 'UNK' character. For example, the dataset may contain an extra 'X' character, which is not a part of the dictionary. To ensure consistent empirical performance between GaKCo and gk-SVM, the user will have to add the extra 'X' character to the dictionary of gk-SVM.

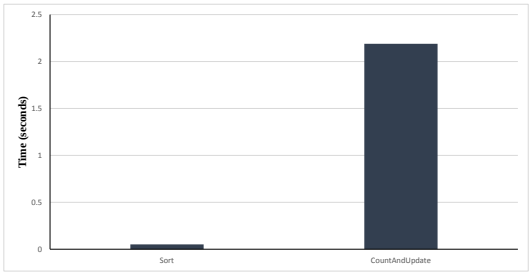

Fig. 8. The GaKCo sorting time vs Count and Update time averaged over all iterations for EP300 DNA dataset (g=10 and k=3).

Table 3. Summary of GaKCo-SVM, gk-SVM and CNN-AUC scores for all datasets. For Web-KB we report the micro-averaged F1-Score since it is a multi classification task with four classes: student, faculty, project and course.

Prediction Task	**Sample properties**			**Best Parameters**			**AUC**		
Datasets	N	Σ	Max(l)	g	k	c	GaKCO-AUC	gk-SVM-AUC	NN-AUC
1.1	3574	20	905	7	5	0.01	0.7453	0.7448	0.7484
1.34	3312			10	1	0.1	0.9903	0.9903	0.9858
2.19	2560			7	1	100	0.8951	0.8951	0.822
2.31	3500			10	7	10	0.9484	0.9497	0.5317
2.1	7062			10	3	10	0.979	0.9895	0.7970
2.34	2738			7	6	0.01	0.8664	0.8660	0.7477
2.41	2646			10	6	0.01	0.7925	0.7925	0.6484
2.8	2480			10	1	10	0.6367	0.6367	0.6801
3.19	3341			8	1	0.1	0.9326	0.9326	0.7050
3.25	3637			10	8	1	0.7967	0.7962	0.5848
3.33	2918			10	5	0.01	0.9018	0.9018	0.8843
3.50	2549			10	7	0.01	0.7768	0.7772	0.8265
CTCF	4000	5	100	10	5	1	0.902	0.902	0.7834
EP300				10	5	1	0.942	0.942	0.6138
JUND				10	7	1	0.91	0.91	0.8317
RAD21				10	5	1	0.901	0.901	0.7937
SIN3A				10	7	1	0.834	0.834	0.8309
Sentiment	9217	36	260	8	4	1	0.8154	0.81	0.5303
WebKB (F1-score)	4163	36	14218	8	5	1	0.9153	0.9116	0.9147

0.4 Comparison of GaKCo with one level of parallelization versus GaKCo with two levels of parallelization

As explained earlier, our GaKCo implementation utilizes the parallelizability of GaKCo over iterations over m mismatches. It is possible to parallelize GaKCo on another level as the calculation of the cumulative mismatch profile C_i is also independent for the $\binom{g}{i}$ iterations. We also performed similar experiments for a two-level parallelization implementation of GaKCo. As summarized in Table 4 and Figure 9, the kernel calculation time decreases manifold, but this speed-up comes at the cost of increased memory usage. If memory is not a constraint, our GaKCo-Parallel+ implementation can be used to further speed up kernel calculation.

[7] * indicates memory issues

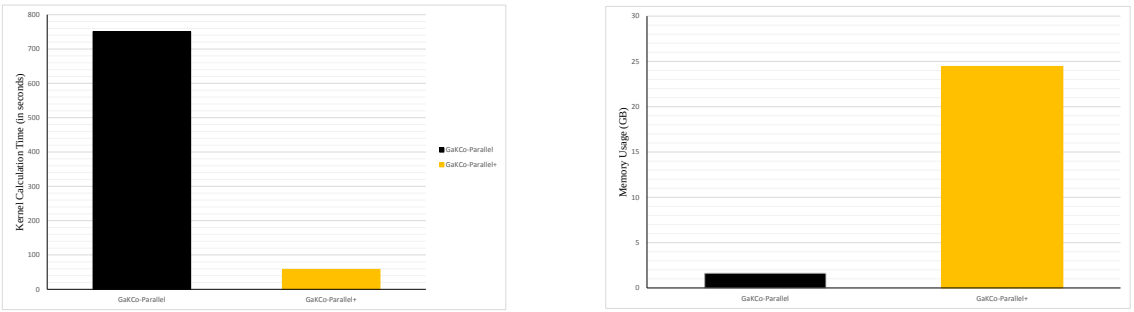

Fig. 9. GaKCo-Parallel (Single Level Parallelization) vs GaKCo-Parallel+ (two levels of parallelization). (a) Kernel Calculation Time (b)Memory Usage

Table 4. GaKCo-Parallel(Single Level Parallelization) vs GaKCo-Parallel+ (two levels of parallelization). The time is in seconds[7]

Dataset	GaKCo-Parallel	GaKCo-Parallel+
1.1	31	13
1.34	266	63
2.19	79	28
2.31	120	17
2.1	974	*
2.34	10	7
2.41	90	19
2.8	184	43
3.19	175	35
3.25	54	15
3.33	151	32
3.50	58	10
WebKB	751	58
Sentiment	522	137