Maximal Pattern Matching with Flexible Wildcard Gaps and One-off Constraint

Anu Dahiya
Department of Computer Science and Engineering
Thapar University
Patiala, India -147004
Email: anudahiya0612@gmail.com

Deepak Garg
Department of Computer Science and Engineering
Thapar University
Patiala, India -147004
Email: dgarg@thapar.edu

***Abstract*—Pattern matching is a fundamental operation in finding knowledge from large amount of biosequence data. Finding patterns help in analyzing the property of a sequence. This paper focuses on the problem of maximal pattern matching with flexible wildcard gaps and length constraints under the one-off condition. The problem is to find the maximum number of occurrences of a pattern P with user specified wildcard gap between every two consecutive letters of P in a biological sequence S under the one-off condition and constraint on the overall length of the matching occurrence. To obtain the optimal solution for this problem is difficult. We propose a heuristic algorithm, MOGO, based on the Nettree data structure to solve this problem. Theoretical analysis and experimental results demonstrate that MOGO performs better than the existing algorithms in most of the cases when tested on real world biological sequences.**

Keywords*—Pattern matching, Wildcard, Gap, Length constraints, One-off.

I. Introduction

Pattern matching with wildcard gaps plays a significant role in biological sequence analysis in computational biology. Apart from bioinformatics [1], other applications of pattern matching with wildcard include information retrieval [2], dictionary query [3] etc. Deoxyribonucleic acid (DNA) is the storehouse of all information and genetic instructions used in the development and functioning of a cell. This information is normally encoded by the specific sequence of nucleotide bases i.e. adenine, guanine, cytosine and thymine (A, T, C, G) [4]. It is the linear order in which these bases are arranged that determines the properties of the cell. It is known that in a DNA sequence, common promoter TATA box appears after the CAAT box at a gap of 30-50 characters [5], [6]. Hence pattern matching with wildcard gaps plays a vital role in analyzing such biological sequences.

The concept of pattern matching with wildcard was first introduced in [7] in which the location of wildcard characters in a pattern is fixed. Time efficiency of the matching result was improved in [8], [9]. In [10], any number of wildcard characters were allowed between the two consecutive characters of the pattern. [3] put a restriction on the overall number of wildcard that can occur in a pattern while allowing for any number of wildcard characters in between the two consecutive characters of the pattern. In [4], [5], [11], user was able to specify the range of wildcard gap between consecutive characters of the pattern but that range was fixed for all the consecutive characters of pattern. E.g. A(0,3)T(0,3)A(0,3)C has fixed gap range of (0,3). [12] eliminated the restriction of fixed user specified range, thus allowing variable user specified gap range. E.g. A(0,1)T(0,3)A was allowed in [12]. The algorithm PAIG (Pattern mAtching with Independent wildcard Gaps) [13] deals with the same problem definition as in [12] with global length constraint added. Global length constraint is the constraint on the overall length of each matching substring of sequence with the given pattern [14]. In addition to the global constraint, the concept of one-off condition was taken into consideration in [14]–[17]. One-off condition means every positional index of a character in a sequence can be used at most once while matching with the given pattern [14]. Applying One-off condition makes the solution to satisfy Apriori property and also removes useless information. BPBM algorithm [15] is based on bit-parallel technology. The algorithm SAIL [14] consumes a lot of time for the large pattern length and was extended in [18] to allow for some errors i.e. approximate pattern matching.

In this paper, we propose an algorithm, MOGO (Maximum Occurrences with Global length constraints and One-off condition) based on the Nettree data structure [19] to solve the problem of pattern matching with wildcards with local and global length constraints under the one-off condition.

The paper is organized as follows. Section 2 formally defines the problem statement. Section 3 presents the proposed algorithm in detail along with an illustrative example and an analysis of its time and space complexity. Section 4 shows the experimental results of the proposed algorithm, its comparison with the other peers and effect of different constraints on the completeness of the proposed algorithm. In Section 5 we draw the conclusion with future work suggestions.

II. Problem Statement

Given a biological sequence S, a pattern P along with user defined local and global constraints, our goal is to find the maximum number of substrings of sequence S that matches the pattern P satisfying the local and global constraints under the one-off condition.

Definition 1: A biological sequence S is defined as

$$S = s_0, s_1, s_2, \ldots s_i \ldots s_{n-1}$$

where n is the length of the sequence S and $s_i \in \{a, t, c, g\}$ where $0 \leq i < n$.

Example 1: A sequence $S = aaattcgatgggcat$ is a biological sequence with length, $n = 15$.

978-1-4799-3080-7/14/$31.00

Definition 2: A pattern P is defined as

$$P = p_0[l_0, u_0]p_1[l_1, u_1]p_2[l_2, u_2]\dots[l_{j-1}, u_{j-1}]p_j[l_j, u_j]\dots[l_{m-2}, u_{m-2}]p_{m-1}$$

where m is the length of the pattern P without wildcards and $p_j \in \{a, t, c, g\}$ where $0 \leq j < m$. Here $[l_j, u_j]$ is the range of wildcard gap allowed between the pattern characters p_j and p_{j+1}. l_j depicts the lower limit on the number of wildcard characters and u_j depicts the upper limit on the number of wildcard characters. This wildcard gap specified between every two consecutive characters of P is called local constraint.

Example 2: $P = a[0, 3]t$ is a pattern with length i.e. $m = 2$. Here between the characters a and t, 0 to 3 wildcard characters are allowed.

Definition 3: Global length constraint is defined as the constraint on the overall length of the substring of the sequence that matches the pattern. It is defined as $[min, max]$. min and max depicts the minimum and maximum allowable overall length of the substring of the sequence that matches the pattern respectively.

Definition 4: m and n being the length of the pattern and the sequence respectively, if there exists positional indices $o_0o_1o_2\dots o_{m-1}$ in a sequence $S = s_0s_1s_2\dots s_{n-1}$ such that characters against those positional indices matches the characters of the pattern $P = p_0[l_0, u_0]p_1[l_1, u_1]p_2[l_2, u_2]\dots[l_{m-2}, u_{m-2}]p_{m-1}$, i.e.

$$s_{o_i} = p_i$$

where $0 \leq i \leq m-1$ then $(o_0o_1o_2\dots o_{m-1})$ is called an occurrence of a pattern in a sequence.

Example 3: Suppose sequence $S = atataaa$ and pattern $P = a[0, 3]t[0, 5]a$. All the possible alignments of pattern P with sequence S satisfying the local constraints are given in Table I.

TABLE I. DIFFERENT ALIGNMENTS OF PATTERN WITH SEQUENCE IN EXAMPLE 3

	0	1	2	3	4	5	6	
S	a	t	a	t	a	a	a	
P	a	t	a					(0, 1, 2)
P	a	t	-	-	a			(0, 1, 4)
P	a	t	-	-	-	a		(0, 1, 5)
P	a	t	-	-	-	-	a	(0, 1, 6)
P	a	-	-	t	a			(0, 3, 4)
P	a	-	-	t	-	a		(0, 3, 5)
P	a	-	-	t	-	-	a	(0, 3, 6)
P			a	t	a			(2, 3, 4)
P			a	t	-	a		(2, 3, 5)
P			a	t	-	-	a	(2, 3, 6)

So there are total of 10 possible occurrences of pattern P in sequence S : $\{(0, 1, 2), (0, 1, 4), (0, 1, 5), (0, 1, 6), (0, 3, 4), (0, 3, 5), (0, 3, 6), (2, 3, 4), (2, 3, 5), (2, 3, 6)\}$.

Example 4: Suppose in example 3, global length constraint $[3, 5]$ is given. In this case , we are left with only six possible occurrences- $\{(0, 1, 2), (0, 1, 4), (0, 3, 4), (2, 3, 4), (2, 3, 5), (2, 3, 6)\}$ as the occurrences $(0, 1, 5), (0, 3, 5)$ are having length 6 and the occurrences $(0, 1, 6), (0, 3, 6)$ are having length 7 whereas the maximum possible length allowed is 5.

Definition 5: If every positional index of a character in a sequence can be used at most once while matching with a pattern, then such a set of occurrences is said to follow the one-off condition. The solution $\{occ_1, occ_2, \dots occ_i\}$ is said to follow the one- off condition if and only if

$$occ_1 \bigcap occ_2 \cdots \bigcap occ_i = \varphi \tag{1}$$

where i is the total number of occurrences in the solution.

Example 5: After applying the one-off condition in example 4, the possible solutions are: $\{(0, 1, 2)\}, \{(0, 3, 4)\}, \{(2, 3, 4)\}, \{(0, 1, 4), (2, 3, 5)\}, \{(0, 1, 4), (2, 3, 6)\}$. Since our problem is to find the maximum number of possible occurrences, we should get as a solution either $\{(0, 1, 4), (2, 3, 5)\}$ or $\{(0, 1, 4), (2, 3, 6)\}$ as both solutions contain 2 occurrences whereas rest of the possible solutions contain only single occurrence.

III. ALGORITHM

A. Algorithm Design

Proposed algorithm is based on the Nettree data structure [19]. Nettree is graph cum tree with one or more roots. Nodes can have zero or more parents except those at the root level. Similarly nodes can have zero or more children except those at the leaf level.

Definition 6: $number\_levels[node.data]$ contains number of different levels at which the positional index of node i.e $node.data$ is occurring.

Definition 7: Considering the paths from a particular node nod to the root level, the sum of the value of the $number\_levels$ of the nodes in the path that contains less number of those nodes that occur at various different levels as compared to other possible paths is called min_occ of the node nod.

Property 1: For a node except those at the root level, the minimum value of the min_occ amongst all the parents of the node plus the $number\_levels$ value of the node itself is known as min_occ of the node.For a node at root level,min_occ is the value of the $number\_levels$ of that particular node.

$$Node.min\_occ = min(node.parents[i].min\_occ) + number\_levels[node.data] \tag{2}$$

where $1 \leq i \leq node.num\_parents$

Definition 8: Considering the paths from a particular node nod to the root level, the sum of the value of the $number\_levels$ of the nodes in the path that contains more number of those nodes that occur at various different levels as compared to other possible paths is called max_occ of the node nod.

Property 2: For a node except those at the root level, the maximum value of the max_occ amongst all the parents of the node plus the $number\_levels$ value of the node itself is known

as max_occ of the node. For a node at root level, max_occ is the value of the $number\_levels$ of that particular node.

$$Node.max\_occ = max(mode.parents[i].max\_occ) + number\_levels[node.data] \quad (3)$$

where $1 \leq i \leq node.num\_parents$

Algorithm 1 MOGO

Require: Sequence S, pattern P and global constraint [min, max]
Ensure: Set of occurrences
1: Build the Nettree for the sequence S and the pattern P
2: Remove nodes having no possible path to any leaf from the Nettree
3: **for** $l \in$ number of leaves at m^{th} level down to 1 step -1 **do**
4: **for** $nod \in$ all nodes of the Nettree **do**
5: number_levels[nod.data]+=1
6: **end for**
7: **for** $nod \in$ all nodes of the Nettree **do**
8: calculate $nod.min\_occ$ and $nod.max\_occ$ according to property 1 and property 2
9: **end for**
10: **for** $nod \in$ all nodes of the Nettree **do**
11: calculate the possible roots for nod satisfying global constraints
12: **end for**
13: **if** l satisfies global constraints **then**
14: occ = OOCL(l, Nettree)
15: solution=solution $\cup$ occ
16: Nettree = Nettree $-$ occ
17: **end if**
18: **end for**
19: **return** solution

Algorithm 2 OOCL

Require: Nettree, Leaf l
Ensure: An occurrence containing Leaf l at last position
1: best_parent = l.parent[l.degree_parents]
2: **for** $r \in$ l.degree_parents down to 1 step -1 **do**
3: **if** l.parent[r] satisfies global constraints **then**
4: **if** l.parent[r].min_occ $<$ best_parent.min_occ **then**
5: best_parent = l.parent[r]
6: **else if** l.parent[r].min_occ == best_parent.min_occ **then**
7: **if** l.parent[r].max_occ $\leq$ best_parent.max_occ **then**
8: best_parent = l.parent[r]
9: **end if**
10: **end if**
11: **end if**
12: **end for**
13: **if** best_parent.degree_parents != 0 **then**
14: OOCL(Nettree, best_parent)
15: **end if**
16: **return** occurrence

B. Algorithm Description

1) MOGO (Maximum Occurrences with Global length constraints and One-off condition): In line 1, Nettree is being created according to the sequence S and Pattern P. Sequence S is scanned from left to right. Nodes and relation between nodes will be created according to the following rules [19]:
Rule1. **Creation of nodes of root level**
If $s_i = p_0$, create and add node to the tail of the level one. In case it is the first node of the level, make head point to this node.
Rule2. **Creation of nodes other than root level**
If $s_i = p_j$ where $j \neq 0$ and the distance between i and the j^{th} level nodes satisfies the local constraints , create and add node to the tail of the $j + 1^{th}$ level.
Rule3. **Creating relation between nodes of different levels**
If the distance between the node created at a level and the nodes at one level up satisfies the local constraints, create a parent-child relation between nodes.

In line 2, nodes that cannot be a part of any path from leaf to the root are being removed. From lines 3 to 18, for each node - number_levels, possible roots satisfying global constraint, min_occ and max_occ are being calculated and OOCL is called iteratively for each leaf satisfying the global constraint in order to get an optimal occurrence containing that leaf.

2) OOCL (Optimal Occurrences Containing Leaf): This algorithm works in a recursive manner by finding the best parent of the node for which it is called till it reaches the root level. It chooses the parent having minimum value of min_occ as the best parent. In case of clash of the minimum value of min_occ, it chooses the one having the minimum value of max_occ. This algorithm returns a single occurrence containing the node with which it was called by MOGO at last position.

C. Complexity Analysis

The space complexity of storing the Nettree is $O(W * m * n)$. Hence the space complexity of MOGO is also $O(W*m*n)$ where m is the length of the pattern, n is the length of the sequence and W is the maximum gap between consecutive letters of the pattern.
The time complexity of lines 1 and 2 of MOGO is $O(W * m * n)$. Lines 4 to 12 of MOGO are having time complexity of $O(W * m * n)$.Time complexity of OOCL is $O(W * m)$. Complexity of lines 15 and 16 of MOGO is $O(m)$. Thus complexity of line 3 through 18 is $O((W * m * n) * \frac{n}{m})$ i.e. $O(W * n * n)$.Thus the overall time complexity of MOGO is $O(W * m * n + W * n * n)$ i.e. $O(W * n * (n + m))$.

D. An Illustration Example

For the sequence $S = aatattaat$ and the pattern $P = a[0, 2]t[0, 1]a[0, 3]t$ and the global length constraint of $[4, 10]$, the nettree being created is shown in Figure 1. In Figure 1, solid line and dotted line depicts the parent-child and child-parent relation respectively.

$number\_levels$ calculated from this Nettree is shown in Table II.

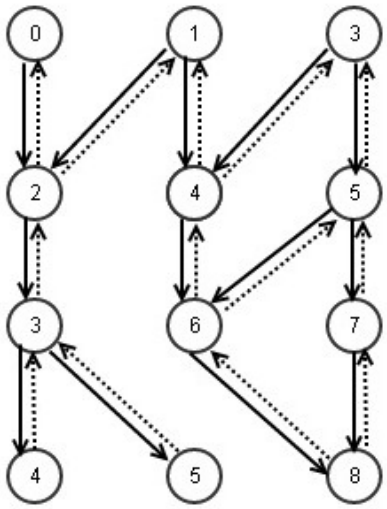

Fig. 1. A Nettree

TABLE II. AN INSTANCE OF NUMBER_LEVELS ARRAY

Array Index	0	1	2	3	4	5	6	7	8
Element	1	1	1	2	2	2	1	1	1

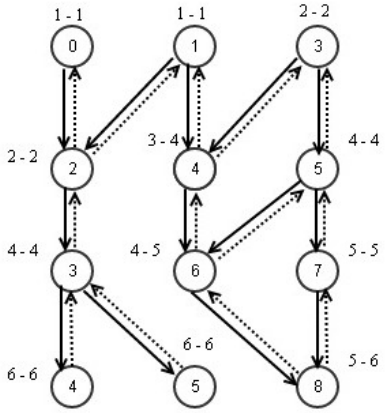

Fig. 2. Nettree with min_occ and max_occ of each node

min_occ and max_occ calculated for each node is shown in Figure 2.

Here the value before '-' represents value of min_occ and the one after '-' represents the value of max_occ. For e.g. if its $3-4$, this means value of min_occ is 3 and that of max_occ is 4.
First rightmost leaf i.e 8 is selected. Amongst all the parents of 8, 6 is having the minimum min_occ value. So 6 is the best parent. Now for 6, 4 is the best parent. Similarly for 4, 1 is the best parent. In this way we get an occurrence $(1, 4, 6, 8)$. These nodes are then removed from the Nettree and same operation is performed on the next leaf node. In this case we get $(0, 2, 3, 5)$ as another occurrence. Then these nodes are again removed from the Nettree and we are left with no other leaf nodes. We get $\{(1, 4, 6, 8), (0, 2, 3, 5)\}$ as the final solution.

IV. EXPERIMENTS

Experiments on the proposed algorithm MOGO are performed using real as well as artificial data. Experiment is performed on machine with configuration Intel(R) Core(TM)2 Duo CPU $T6500@2.10$ GHz 2.10 GHz, 3 GB of RAM and Windows 7 OS. Algorithm has been implemented in Python 3.2.

TABLE III. BIOLOGICAL SEQUENCES

Sequence Number	Locus	Length
S1	CY058563	2286
S2	CY058562	2299
S3	CY058561	2169
S4	CY058556	1720
S5	CY058559	1516
S6	CY058558	1418
S7	CY058557	982
S8	CY058560	844

A. Experiment on Real Data

MOGO has been tested on real world biological data. 8 different segments of the H1N1 (Swine Flu) virus are downloaded from the website of National Center for Biotechnology Information [20]. Locus of the sequences and their length are given in the Table III. Each of these 8 biological sequences have been tested against 4 different patterns with wildcard gaps given by Min et al. [13]. These 4 patterns are shown in Table IV. Table V specifies the minimum and maximum length parameters of global length constraint for each of the pattern given in Table IV.

TABLE IV. PATTERNS WITH WILDCARD GAPS

Pattern Number	Pattern
P1	a[0,3]t[0,3]a[0,3]t[0,3]a[0,3] t[0,3]a[0,3]t[0,3]a[0,3]t[0,3]a
P2	g[1,5]t[0,6]a[2,7]g[3,9]t[2,5]a[4,9]g[1,8]t[2,9]a
P3	g[1,9]t[1,9]a[1,9] g[1,9]t[1,9]a[1,9] g[1,9]t[1,9]a[1,9]g[1,9]t
P4	g[1,5]t[0,6]a[2,7]g[3,9]t[2,5]a[4,9]g[1,8]t[2,9]a[1,9]g[1,9]t

Table VI shows the results obtained by conducting the experiment and its comparison with the results of the existing algorithms SAIL and HSO (Heuristic Search Occurrences) as given in [16].

TABLE V. GLOBAL LENGTH CONSTRAINTS

Pattern Number	Minimum Length	Maximum Length
P1	11	41
P2	24	57
P3	21	101
P4	27	73

TABLE VI. EXPERIMENTAL RESULTS

Pattern	Algorithm	S1	S2	S3	S4	S5	S6	S7	S8
P1	SAIL	13	9	10	15	11	5	3	3
	HSO	13	9	10	15	11	5	3	3
	MOGO	13	9	10	15	11	5	3	3
P2	SAIL	66	69	59	54	42	39	31	27
	HSO	67	71	62	54	42	41	33	28
	MOGO	67	73	65	55	44	44	33	32
P3	SAIL	66	69	66	54	45	42	33	28
	HSO	64	70	68	52	43	43	33	26
	MOGO	68	70	72	52	44	43	32	27
P4	SAIL	49	50	49	40	32	31	24	20
	HSO	51	58	52	46	37	30	26	21
	MOGO	48	58	54	48	37	35	26	22

According to the results, MOGO gives 43.75% better results than the algorithm SAIL and 37.50% better results than the algorithm HSO. Hence MOGO performs better than the existing algorithms SAIL and HSO by searching more number of occurrences of a pattern in a sequence.

B. Experiment on Artificial Data

For the problem taken into consideration, no complete solution has been developed so far. In this subsection, we analyze the effect of different constraints on the completeness of the solution. Some of the parameters that affects the completeness of the solution includes length of the pattern i.e. m and the maximum wildcard gap between consecutive letters of the pattern i.e. W. For the purpose of this analysis, artificial data is used as it is having the complete solution for the problem and is downloaded from [21]. Artificial data is generated by data generator [17]. Input to the data generator is alphabet Σ, pattern, length of sequence n and maximal support sup and output is the sequence with exact support value of pattern in sequence under the one-off condition. So in this way, with artificial data, we have complete solution of the problem. A parameter, $Accuracy$, is used to measure the completeness of the algorithm. Accuracy is defined as:

$$Accuracy = \frac{Num\_occ}{Total\_occ} \tag{4}$$

where Num_occ is number of occurrences of pattern returned by the algorithm MOGO and $Total\_occ$ is the total number of all possible occurrences of the pattern in the sequence. We get $Total\_occ$ from the artificial data set. For the same combination of pattern and sequence in artificial data set, we run the algorithm MOGO in order to obtain Num_occ.

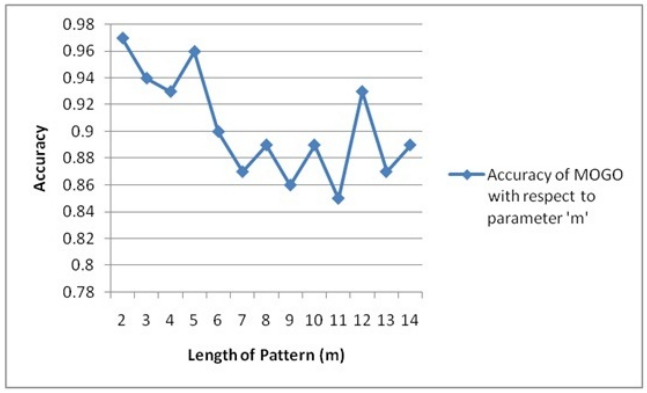

Fig. 3. Effect of length of the pattern on the accuracy of the algorithm MOGO

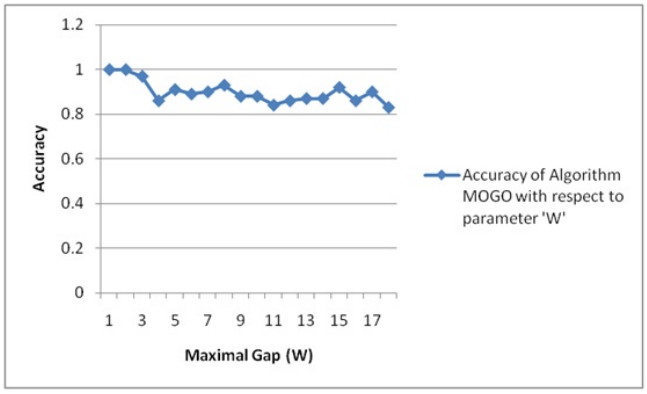

Fig. 4. Effect of maximum wildcard gap on the accuracy of the algorithm MOGO

Figure 3 shows the effect of length of the pattern m on accuracy of the algorithm. Analysis is done for 13 different values of m ranging from 2 to 14. For each value of m, 10 artificial data sets are considered and the accuracy for that particular value of m is equal to the average of accuracies corresponding to those 10 artificial data sets. From the graph, it is clearly visible that with the increase in m i.e. length of the pattern, accuracy of the algorithm is gradually decreasing.

Figure 4 shows the effect of maximum wildcard gap between consecutive letters of the pattern W on accuracy of the algorithm. Analysis is done for 18 different values of W ranging from 1 to 18. For each value of W, 10 artificial data sets are considered and the accuracy for that particular value of W is equal to the average of accuracies corresponding to those 10 artificial data sets. From the graph, it is clearly visible that with the increase in W i.e. gap, accuracy of the algorithm is gradually decreasing. As the gap increases, the probability of overlapping of the possible occurrences becomes higher. Thus there are more chances of losing the occurrences leading to loss of accuracy.

V. Conclusion

In this paper, we considered the problem of pattern matching with flexible wildcard gaps between every two consecutive letters of pattern under the one-off constraint and proposed the algorithm MOGO based on the Nettree which performs better than its peers SAIL and HSO according to theoretical analysis and experimental results. In the future, we will extend our work to allow for some errors while matching the pattern i.e. approximate pattern matching and also support multi-pattern matching with flexible wildcard gaps under one-off condition.

References

[1] N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot, "Bases of motifs for generating repeated patterns with wild cards," *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, vol. 2, no. 1, pp. 40–50, 2005.

[2] R. S. Aygün, "S2s: structural-to-syntactic matching similar documents," *Knowledge and information systems*, vol. 16, no. 3, pp. 303–329, 2008.

[3] R. Cole, L.-A. Gottlieb, and M. Lewenstein, "Dictionary matching and indexing with errors and don't cares," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM, 2004, pp. 91–100.

[4] M. Zhang, B. Kao, D. W. Cheung, and K. Y. Yip, "Mining periodic patterns with gap requirement from sequences," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 2, p. 7, 2007.

[5] U. Manber and R. Baeza-Yates, "An algorithm for string matching with a sequence of don't cares," *Information Processing Letters*, vol. 37, no. 3, pp. 133–136, 1991.

[6] T. Akutsu, "Approximate string matching with variable length don't care characters," *IEICE TRANSACTIONS ON INFORMATION AND SYSTEMS E SERIES D*, vol. 79, pp. 1353–1354, 1996.

[7] M. J. Fischer and M. S. Paterson, "String-matching and other products." DTIC Document, Tech. Rep., 1974.

[8] P. Indyk, "Faster algorithms for string matching problems: Matching the convolution bound," in *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*. IEEE, 1998, pp. 166–173.

[9] A. Kalai, "Efficient pattern-matching with don't cares," in *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2002, pp. 655–656.

[10] G. Kucherov and M. Rusinowitch, "Matching a set of strings with variable length don't cares," *Theoretical Computer Science*, vol. 178, no. 1, pp. 129–154, 1997.

[11] X. Zhu and X. Wu, "Discovering relational patterns across multiple databases," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 726–735.

[12] G. Navarro and M. Raffinot, "Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching," *Journal of Computational Biology*, vol. 10, no. 6, pp. 903–923, 2003.

[13] F. Min, X. Wu, and Z. Lu, "Pattern matching with independent wildcard gaps," in *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*. IEEE, 2009, pp. 194–199.

[14] G. Chen, X. Wu, X. Zhu, A. N. Arslan, and Y. He, "Efficient string matching with wildcards and length constraints," *Knowledge and information systems*, vol. 10, no. 4, pp. 399–419, 2006.

[15] D. Guo, X.-L. Hong, X.-G. Hu, J. Gao, Y.-L. Liu, G.-Q. Wu, and X. Wu, "A bit-parallel algorithm for sequential pattern matching with wildcards," *Cybernetics and Systems*, vol. 42, no. 6, pp. 382–401, 2011.

[16] Y. Wu, X. Wu, H. Jiang, and F. Min, "A nettree for approximate maximal pattern matching with gaps and one-off constraint," in *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, vol. 2. IEEE, 2010, pp. 38–41.

[17] D. Guo, X. Hu, F. Xie, and X. Wu, "Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph," *Applied intelligence*, vol. 39, no. 1, pp. 57–74, 2013.

[18] D. He, X. Wu, and X. Zhu, "Sail-approx: an efficient on-line algorithm for approximate pattern matching with wildcards and length constraints," in *Bioinformatics and Biomedicine, 2007. BIBM 2007. IEEE International Conference on*. IEEE, 2007, pp. 151–158.

[19] Y. Wu, X. Wu, F. Min, and Y. Li, "A nettree for pattern matching with flexible wildcard constraints," in *Information Reuse and Integration (IRI), 2010 IEEE International Conference on*. IEEE, 2010, pp. 109–114.

[20] "National center for biotechnology information website," http://www.ncbi.nlm.nih.gov/, [Online; accessed 06-December-2013].

[21] "Hfut data mining and intelligent computing laboratory website," http://dmic.hfut.edu.cn/HFUT_DMIC/DanGuo/test/, [Online; accessed 10-May-2014].