Online Recognition of Dictionary with One Gap

Amihood Amir[1,2], Avivit Levy[3], Ely Porat[1], and B. Riva Shalom[3]

[1] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel.
E-mail: {amir, porately}@cs.biu.ac.il
[2] Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218.
[3] Department of Software Engineering, Shenkar College, Ramat-Gan 52526, Israel.
E-mail: {avivitlevy, rivash}@shenkar.ac.il

Abstract. We formalize and examine the online Dictionary Recognition with One Gap problem (DROG) which is the following. Preprocess a dictionary D of d patterns, where each pattern contains a special *gap* symbol that can match any string, so that given a text that arrives online, a character at a time, we can report all the patterns from D that have not been reported yet and are suffixes of the text that has arrived so far, before the next character arrives. The gap symbols are associated with *bounds* determining the possible lengths of matching strings. Online DROG captures the difficulty in a bottleneck procedure for cyber-security, as many digital signatures of viruses manifest themselves as patterns with a single gap.
Following the work of [4] on the closely related online Dictionary Matching with One Gap problem (DMOG), we provide algorithms whose time cost depends linearly on $\delta(G_D)$, where G_D is a bipartite graph that captures the structure of D and $\delta(G_D)$ is the *degeneracy* of this graph. These algorithms are of practical interest since although $\delta(G_D)$ can be as large as $\sqrt{d}$, and even larger if G_D is a multi-graph, it is typically a very small constant in practice. Finally, when $\delta(G_D)$ is large we describe other efficient solutions.

1 Introduction

Cyber-security is a critical modern challenge. Network intrusion detection systems (NIDS) perform protocol analysis, content searching, recognizing and matching, in order to detect harmful software. Such malware may appear non-contiguously, scattered across several packets, which necessitates matching *gapped* patterns.

A *gapped pattern* P is one of the form $P_1\ \{\alpha, \beta\}\ P_2$, where each subpattern P_1, P_2 is a string over alphabet Σ, and $\{\alpha, \beta\}$ matches any substring of length at least α and at most β, which are called the *gap bounds*. Gapped patterns may contain more than one gap, however, those considered in NIDS systems typically have at most *one* gap, and are a serious bottleneck in such applications [22,4]. Therefore, an efficient solution for this case is of special interest.

Though the gapped pattern matching problem arose over 20 years ago in computational biology applications [20,14] and has been revisited many times in the intervening years (e.g. [19,8,17,7,12,21,23]), network intrusion detection systems applications necessitate a different generalization of the problem. These applications motivate the *dictionary matching with one gap* (DMOG) problem defined by [4], which is a variant of the well-studied dictionary matching problem (see, e.g. [1,2,9,3,11]). The dictionary, which is the set of d gapped patterns to be detected, could be quite large.

The DMOG problem was, therefore, studied [6,15,4] for both the offline and the online settings. Lower bounds on the complexity of this problem as well as (almost) matching upper bounds were described in [4]. These lower bounds expose a hidden parameter of input dictionary that sheds light on the reason why this problem has

Amihood Amir, Avivit Levy, Ely Porat, B. Riva Shalom: Online Recognition of Dictionary with One Gap, pp. 3–17.
Proceedings of PSC 2017, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-06193-0 © Czech Technical University in Prague, Czech Republic

resisted many researcher's attempts at finding a definitive efficient solution on the one hand, while on the other hand, enables describing the solutions in terms of this parameter. We elaborate on this issue in Section 2.

The definition of the DMOG problem requires reporting all occurrences of the dictionary patterns. This is a necessary requirement in order to remove all viruses from a given source. However, the size of the input may be quite large if dictionary patterns occur many times in the source. The process of malware detection is required to be very fast, and in many cases we would prefer a faster scan in order to determine whether the source stream is infected by viruses or not. We would also like to know which viruses attacked the source in case it is affected, so that an appropriate (slower) exhaustive infection recovery procedure can be applied on the source. Motivated by this need of NIDS applications, we focus in this paper on the recognition of the set of viruses that exists in the source, and formally define the *Dictionary Recognition with One Gap problem (DROG)* as follows:

Definition 1. The Dictionary Recognition with One Gap problem (DROG) *is:*
Input: *A text T of length $|T|$ over alphabet Σ, and a dictionary D of d gapped patterns $P_1, \ldots, P_d$ over alphabet Σ, where each pattern has at most one gap.*
Output: The maximal subset $S \subseteq D$, where pattern $P_i \in S$ appears at least once in T.

We study the more practical *online* DROG problem. The dictionary D can be preprocessed in advance, resulting in a data structure. Given this data structure the text T is presented one character at a time, and when a character arrives only the subset of patterns with a match ending at this character *that were not previously reported* should be reported before the next character arrives. Two cost measures are of interest: a preprocessing time and a time per character.

	Preprocess Time	Total Query Time	Algorithm Type	Remark
[16]	none	$\tilde{O}(\lvert T\rvert + \lvert D\rvert)$	online	reports only first occurrence
[23]	$O(\lvert D\rvert)$	$\tilde{O}(\lvert T\rvert + d)$	online	reports only first occurrence
[13]	$O(\lvert D\rvert)$	$O(\lvert T\rvert \cdot lsc + socc)$	online	reports one occurrence per pattern and location
[5]	$\tilde{O}(\lvert D\rvert)$	$\tilde{O}(\lvert T\rvert(\beta - \alpha) + op)$	offline	DMOG
[15]	$O(\lvert D\rvert)$	$\tilde{O}(\lvert T\rvert(\beta^* - \alpha^*) + op)$	offline	DMOG
[4]	$O(\lvert D\rvert)$	$\tilde{O}(\lvert T\rvert \cdot \delta(G_D) \cdot lsc + op)$	online	DMOG
[4]	$O(\lvert D\rvert)$ $O(\lvert D\rvert)$	$\Omega(\lvert T\rvert \cdot \delta(G_D)^{1-o(1)} + op)$ $\Omega(\lvert T\rvert \cdot (\beta - \alpha)^{1-o(1)} + op)$	online or offline	DMOG
This paper	$O(\lvert D\rvert)$	$\tilde{O}(\lvert T\rvert \cdot \delta(G_D) \cdot lsc + d)$	online	DROG

Table 1. Comparison of previous work and some new results. The parameters: lsc is the longest suffix chain of subpatterns in D, $socc$ is the number of subpatterns occurrences in T, op is the number of pattern occurrences in T, α^* and β^* are the minimum left and maximum right gap borders in the non-uniformly bounded case, $\delta(G_D)$ is the degeneracy of the graph G_D representing dictionary D.

Previous Work. Finding efficient solutions for the problem has proven to be a difficult algorithmic challenge as little progress has been obtained even though many researchers in the pattern matching community and the industry have tackled it. Table 1 describes a summary and comparison of previous work. It illustrates that previous formalizations of the problem until that of [4], either do not enable detection of all intrusions or are incapable of detecting them in an online setting, and therefore, are inadequate for NIDS applications. Table 1 also demonstrates that the upper bounds of [4] for the DMOG problem are essentially optimal (assuming some popular conjectures). Most importantly, Table 1 demonstrates that no previous work has been done on the DROG problem as formalized in this paper.

Our Results. Our goal in this paper is that the time per character cost would be independent of the number of occurrences of dictionary patterns in the text. This is a nontrivial requirement as we can no longer afford costly operations that were accounted for the size of the output for the detection of dictionary patterns at query time in the online DMOG solutions of [4]. In our case such costly operations can be afforded for newly detected patterns only. This raises the difficulty of limiting the detection process to the dynamically changing set of yet undetected dictionary patterns.

Paper Organization. In Section 2 we give a brief review of the solutions to the online Dictionary Matching with One Gap (DMOG) problem suggested by [4]. Section 3 describes our solution for the online Dictionary Recognition with One Gap (DROG) problem, which is based on the solutions described in Section 2 for the DMOG problem with changes and adoptions in order to fulfill the requirement of the DROG problem. Section 4 concludes the paper and poses some open problems.

2 An Overview of the DMOG Solutions

In this section we give a brief description of the DMOG solutions of [4]. The reader who is familiar with their ideas and techniques can skip this section.

The Bipartite Graph G_D. The first baseline idea of their solutions is to represent the dictionary as a graph $G_D = (V, E)$, where the subpatterns are the vertices, and there is an edge $(u, v) \in E$ if and only if there is a pattern $P \in D$, where P^1 is associated with node u and P^2 is associated with v. Moreover, the graph $G_D = (V, E)$ is converted to a bipartite graph by creating two copies of V called L (the left vertices) and R (the right vertices) in the following way. For every edge $(u, v) \in E$, an edge is added to the bipartite graph from $u_L \in L$ to $v_R \in R$, where u_L is a copy of u and v_R is a copy of v.

Graph Orientations. The next baseline idea is to preprocess G_D using linear time greedy algorithm suggested by Chiba and Nishizeki [10] to obtain a $\delta(G_D)$-orientation of the graph G_D, where an orientation of an undirected graph $G = (V, E)$ is called a *c-orientation* if every vertex has out-degree at most $c \geq 1$. The orientation is viewed as assigning "responsibility" for all data transfers occurring on an edge to one of its endpoints, depending on the direction of the edge in the orientation (regardless of the actual direction of the edge in the input graph G_D). The notation of an edge $e = (u, v)$ is as oriented from u to v, while e could be directed either from u to v or from v to u. The vertex u is called a *responsible-neighbour* of v and v an *assigned-neighbour* of u. The notion of graph *degeneracy* $\delta(G_D)$ is defined as follows. The degeneracy of an undirected graph $G = (V, E)$ is $\delta(G) = \max_{U \subseteq V} \min_{u \in U} d_{G_U}(u)$, where d_{G_U} is the degree of u in the subgraph of G induced by U. In words, the degeneracy of G is the largest minimum degree of any subgraph of G. A non-multi graph G with m edges has $\delta(G) = O(\sqrt{m})$, and a clique has $\delta(G) = \Theta(\sqrt{m})$. The degeneracy of a multi-graph can be much higher.

Subpatterns Detection Mechanism. An Aho-Corasick (AC) Automaton [1] is used for determining when a subpattern arrives using a standard binary encoding technique, so that each character arrival costs $O(\log |\Sigma|)$ worst-case time for recognizing the arrival of a dictionary subpattern. For simplicity of exposition, $|\Sigma|$ is assumed to be constant. Since each arriving character may correspond to the arrival of several subpatterns when a subpattern is a proper suffix of another, the complexities are phrased in terms of lsc, which is the maximum number of vertices in the graph that arrive due to a character arrival. The lsc factor was used even in solutions for simplified relaxations of the DMOG problem [13]. Another issue is that, since subpatterns may be long, a delay must be accommodated in the time a vertex corresponding to a second subpattern is treated as if it has arrived, thus inducing a minor additive space usage.

Two variants of the gapped dictionary are considered having either uniformly bounded gap borders or non-uniformly bounded gap borders. In the former case, all gapped patterns of the dictionary have the same gaps borders $\{\alpha, \beta\}$, whereas in the latter, every pattern P_i has its own gap borders $\{\alpha_i, \beta_i\}$. Two sets of solutions are described: for sparse graphs, where $\delta(G_D) = o(\sqrt{d})$, and for dense graphs. The solutions for these four cases are described hereafter.

2.1 DMOG for Sparse Graphs

Uniformly Bounded Gaps. The data structures used in this case are:

1. For each vertex $v \in R$, a list $\mathcal{L}_v$ maintaining all responsible-neighbours of v, $u \in L$, that arrived at least α and at most β time units ago.
2. For each vertex $u \in L$, an ordered list of time stamps τ_u of the times u arrived within the appropriate gap to the current time unit (text index).
3. The list $\mathcal{L}_\beta$ of delayed vertices $u \in L$ for at least α time units before they are considered.

The $\mathcal{L}_v$ lists are updated by deleting u nodes that arrived more than β time units ago and inserting u nodes that just arrived α time units ago and do not appear already in the data structure. Therefore, when an appearance of node v is detected, all the patterns $u\{\alpha, \beta\}v$ for $u \in \mathcal{L}_v$ are reported according to the time stamps in τ_u, as the output includes all appearances of the gapped patterns. In addition, the edges for which v is their responsible-neighbour are scanned, and those for which the assigned-neighbour u is marked as arrived, are reported.

The removal of $u \in L$ from $\mathcal{L}_v$ must be delayed by at least $m_v - 1$ time units, where m_v is the length of the substring represented by v. If u is removed from $\mathcal{L}_v$ after a delay of $m_v - 1$, then we may be forced to remove a large number of such vertices at a given time. Therefore, the removal of u is delayed by $M - 1$ time units, where M is the length of the longest subpattern that corresponds to a vertex in R.

Time and Space Complexity: [4] show that using the above data structures, the $DMOG$ problem with uniformly bounded gap borders can be solved in $O(|D|)$ preprocessing time, $O(\delta(G_D){\cdot}lsc{+}op)$ time per text character, where *op* is the number of patterns that are reported due to the character arriving, and $O(|D|{+}lsc{\cdot}(\beta{-}\alpha{+}M){+}\alpha)$ space.

Non-Uniformly Bounded Gaps. In the case of non-uniformly bounded gaps, each edge $e=(u,v)$ has its own boundaries $\{\alpha_e,\beta_e\}$, yielding a multi-graph. Let α^* and β^* be the minimum left and maximum right gap borders in the non-uniformly bounded dictionary. A framework similar to the previous subsection is used, yet, instead of the list $\mathcal{L}_v$, a fully dynamic data structure S_v supporting 4-sided 2-dimensional orthogonal range reporting queries, is used for saving the occurrences of responsible neighbour of v.

For each responsible-neighbour $u \in L$ of v, that arrived in the active window in time t, where $e=(u,v)$, the point $(t{+}\alpha_e{+}1, t{+}\beta_e{+}1)$ is inserted into S_v, yielding the information saved is the intervals in which an occurrence of v implies an occurrence of a gapped pattern. When a vertex $v \in R$ arrives at time t, a range query $[0,t]\times[t,\infty]$ over S_v returns the points that have (x,y) coordinates in the given range, thus a pattern appearance.

To implement S_v, a Mortensen's data structure [18] is used. It supports the set of $|S_v|$ points from $\mathbb{R}^2$ with $O(|S_v|\log^{7/8+\epsilon}|S_v|)$ words of space, insertion and deletion time of $O(\log^{7/8+\epsilon}|S_v|)$ and $O(\frac{\log|S_v|}{\log\log|S_v|}+op)$ time for range reporting queries on S_v, where *op* is the size of the output.

Time and Space Complexity: [4] show that using the above, the $DMOG$ problem with non-uniformly bounded gap borders on a graph G with m edges (gapped patterns) and n vertices can be solved with $O(|D|)$ preprocessing time, $\tilde{O}(\delta(G)+op)$ time per query vertex, where *op* is the number of edges reported due to the vertex arriving, and $\tilde{O}(m+\delta(G)(\beta^*-\alpha^*)+\alpha^*)$ space.

2.2 DMOG for Dense Graphs

In the case of dense graphs where $\delta(G_D)=\Omega(\sqrt{d})$, the solutions described above require $O(lsc\cdot\sqrt{d})$ time. For such cases a different method for orienting the graph is suggested by [4], referred to as a *threshold* orientation, where a vertex in G_D is defined as *heavy* if it has more than $\sqrt{d/lsc}$ neighbours, and *light* otherwise. Hence, the number of heavy vertices is less than $\sqrt{lsc\cdot d}$. An edge where at least one of its endpoints is light is oriented to leave the light vertex. For such edges the algorithms from the previous subsection are applied in $\tilde{O}(lsc+\sqrt{lsc\cdot d}+op)$ time complexity.

Reporting edges between two heavy vertices is done differently. Although the number of vertices from L that arrive at the same time can be as large as lsc and the number of neighbours of each such vertex can be very large, the number of heavy vertices in R is still less than $\sqrt{lsc\cdot d}$. So [4] use a batched scan on all vertices of R to keep the time cost low. The vertices from L are ordered in a tree T according to suffix relations between the subpatterns associated with the vertices, where a vertex u is an ancestor of a vertex u' if and only if the subpattern associated with u is a suffix of the subpattern associated with u'.

Uniformly Bounded Gaps. Let $R = \{v_1, v_2, \ldots\}$, where $|R| = O(\sqrt{lsc \cdot d})$, since we only deal with heavy vertices.

For this case, the $\mathcal{L}_v, \mathcal{L}_\beta, \tau_u$ data structures are used as well as the framework of the solution to $DMOG$ for bounded gaps in sparse graphs, as described in Subsection 2.1. In addition, in order to add vertices that are suffix of each other to $\mathcal{L}_v$ in a single operation, the following data structures are also used:

1. For each vertices $u \in L$ and $v_i \in R$ such that $e = (u, v_i) \in E$, **a pointer** $next(e)$ is set to an edge $e' = (u', v_i)$ where u' is the lowest proper ancestor of u in T such that there is an edge from u' to v_i. If no such vertex u' exists then $next(e) = null$. All these pointers can be added in linear time, and their space usage is linear.
2. For each vertex $u \in L$, **an array** $A_u[]$ of size $|R|$ is built, where $A_u[i]$ is a pointer to a list of edges connecting all ancestors of u in T(which may be u) to v_i. If $e = (u, v_i) \in E$, then $A_u[i]$ points to $e = (u, v_i)$, and the list of all ancestors of u in T that have edges touching v_i is obtained through the $next(\cdot)$ pointers. Similarly, if there is no edge (u, v_i) then the entry of $A_u[i]$ points to the edge (u', v_i) where u' is the lowest proper ancestor of u in T such that there is an edge from u' to v_i. If no such edge exists then $A_u[i] = null$.

The $A_u[]$s arrays are constructed by filling $A_u[i]$ while consulting $A_{u'}[]$, where u' is a proper ancestor of u and $A_{u'}[]$ was already filled. In order to reduce space usage of the A_u arrays, the A_u arrays are constructed during preprocessing time only for specially chosen $O(\sqrt{\frac{d}{lsc}})$ vertices so that the time cost for constructing the rest of the A_u arrays online is $O(\sqrt{lsc \cdot d})$.

Time and Space Complexity: [4] show that the DMOG problem with one gap and uniform gap borders can be solved with $O(|D|)$ preprocessing time, $O(lsc + \sqrt{lsc \cdot d} + op)$ time per text character, and $O(|D| + lsc(\beta - \alpha + M) + \alpha)$ space.

Non-Uniformly Bounded Gaps. Recall that for the non-uniform gaps, the gap boundaries of an edge e are denoted by α_e and β_e. For the case of dense graphs and unbounded gaps [4] used different data structures:

1. For each $e = (u, v_i) \in E$, **an array** $next_e$ of size $\beta_e - \alpha_e + 1$ is maintained, where for $\alpha_e \leq j \leq \beta_e$, $next_e[j]$ points to an edge $e' = (u', v_i)$ such that u' is the lowest ancestor of u in T (possibly u itself) such that there is an edge $e' = (u', v_i)$ where $\alpha_{e'} \leq j \leq \beta_{e'}$ and the pointers $next_e[j]$ do not form a loop. If no such edge exists then $next_e[j] = null$. (For simplicity, the indices of the array are treated as starting from α_e and ending at β_e)
2. For each pair of vertices $u \in L$ and $v_i \in R$, **an array** $W_{u,i}$ of size $\beta^* - \alpha^* + 1$ is maintained. If $e = (u, v_i) \in E$, then $W_{u,i}[j]$ is a pointer to e if its gap boundaries include j, and to $next_e[j]$ otherwise.The remaining entries of $W_{u,i}[j]$ are null.
3. For each vertex $v_i \in R$, **a cyclic *active window* array** AW_i of size $\beta^* - \alpha^* + M + 1$ is maintained, where $AW_i[j]$ is a pointer to a list of lists of edges that all need to be reported if v_i appears in $j - 1$ time units from now.

The total space usage for the $next_e$ pointer arrays is $\rho := \sum_{e \in E}(\beta_e - \alpha_e + 1) \leq d(\beta^* - \alpha^*)$, and they can be constructed in $O(\rho)$ time. Yet, if all the arrays $W_{u,i}$ are computed in the preprocessing, the time and space would be $O(lsc \cdot d(\beta^* - \alpha^*))$. Therefore, [4] reduce this cost by postponing the construction of a carefully chosen part of them to the query time.

Time and Space Complexity: [4] show that the DMOG problem with non-uniform gap borders can be solved with $O(|D| + d(\beta^* - \alpha^*))$ preprocessing time, $\tilde{O}(lsc + \sqrt{lsc \cdot d}(\beta^* - \alpha^* + M) + op)$ time per query text character, and $\tilde{O}(|D| + d(\beta^* - \alpha^*) + \sqrt{lsc \cdot d}(\beta^* - \alpha^* + M) + \alpha^*)$ space.

3 Solving Online Dictionary Recognition with One Gap

Our solution follows the framework of [4] showing that it is possible to make changes in their algorithms in order to solve the DROG problem. Recall that the definition of the DROG problem requires reporting only a single appearance of each gapped pattern in the dictionary, where each gapped pattern is represented as an edge in the bipartite graph G_D. Our basic idea is, therefore, quite simple: in order to avoid repetitious reports of an edge (u, v), after the first time an edge is reported we delete it from the graph, thus assuring that u will not be inserted to the data structures maintaining v's responsible neighbours again. In order to avoid considering vertices whose associated edges are all reported, we add two counters to each vertex v, *lcount* and *rcount*, which are initialized with the number of responsible neighbours of v and with the number of neighbours that v is responsible for, respectively. Each report and deletion of an edge (u, v) implies a decrease in $lcount(u)$ and in $rcount(v)$.

The remaining task we should carefully take care of is to assure that for every edge (u, v), u appears only once in the data structure of v, so that reporting the edge will be unique even if v occurs again. This task is nontrivial since in some cases it is not possible to save a single copy of u in the data structure of v (otherwise, we might miss an occurrence of a dictionary gapped pattern), therefore, the deletion of the edge (u, v) requires a deletion of additional appearances of u from the data structure associated with v. This should be done without causing an unbearable overhead in the time complexity.

In the following subsections we consider the four solutions described in Section 2, giving for each of them a tailored adaptation for the online *Dictionary Recognition with One Gap* problem.

3.1 DROG for Sparse Graphs

Uniformly Bounded Gaps. Following the framework described in Subsection 2.1, We use the $\mathcal{L}_v$ lists to maintain at most a single appearance of the responsible neighbours of v. Hence, when going over the list and reporting edges in case v occurred, each edge is reported once without scanning the τ_u list of u appearance times. Therefore, the solution to the DROG problem for uniformly bounded gap borders is identical to the solution for DMOG with the additional task of deleting an edge after its first report, updating the relevant *rcount* and *lcount* and considering only vertices whose *rcount* and *lcount* values are non zero.

This immediately gives Theorem 2.

Theorem 2. *The online DROG problem with uniformly bounded gap borders on a graph G_D with m edges and n vertices can be solved in $O(m + n)$ preprocessing time, $O(lsc \cdot \delta(G_D) + op^*)$ time per query vertex, where op^* is the number of new distinct dictionary patterns reported due to a character arrival, and $O(m + \beta)$ space.*

Non Uniformly Bounded Gaps. In this case, for every vertex $v \in R$, the data structure S_v supporting 4-sided 2-dimensional orthogonal range reporting queries saves the occurrences of responsible neighbours of v, as in Subsection 2.1. Now, the deletion of a reported edge from G_D is not sufficient in order to assure a unique report of edge occurrence, since the same vertex u can be represented by several points in a certain S_v data structure due to several arrival times. If several points are within the query bounds, the range query will return all these occurrences of the edge (u, v). We need to avoid such redundant reports.

In order to avoid an increase in the time complexity, we modify the algorithm as follows. When a vertex u arrives at time t, each assigned-neighbour v such that $e = (u, v)$, the point $(t + \alpha_e + 1, t + \beta_e + 1)$ is inserted to the data structure S_v, representing a time interval in which an occurrence of v yields an occurrence of the edge e. Our modification is to unite every two points representing overlapping or adjacent intervals into a single point. This procedure is delicately performed, as the intervals may be separated again, when one of the occurrences of u represented by the interval becomes irrelevant – when located beyond the gaps bounds. An example of the union effect is depicted in Figure 1. In addition, all intervals associated with the same edge are linked, in order to enable deleting all of them, when the edge is reported due to one of the intervals.

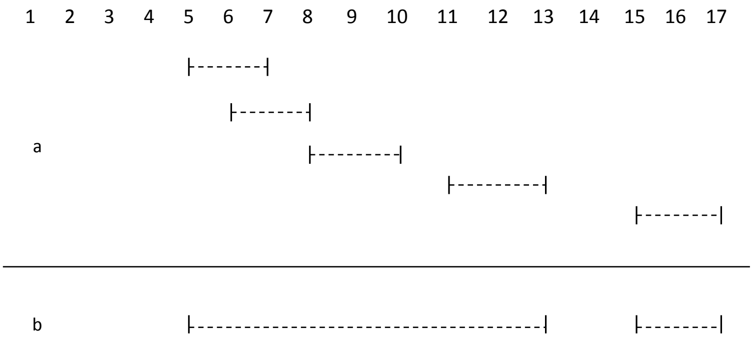


Figure 1. Consider $e = (bbb\{2 - 4\}a)$ and $\tau_u = \{2, 3, 5, 8, 12\}$. (a) depicts the intervals represented by points that are inserted to S_a by the DMOG algorithm. (b) depicts the intervals represented by points that are inserted to S_a by the DROG algorithm.

The modified algorithm is:
For an arrived vertex at query time t, do:

1. If the arrived vertex is $v \in R$, such that $rcount(v) \neq 0$,
   (a) A range query $[0, t] \times [t, \infty]$ is performed over S_v. Edges representing the range output are reported.
   (b) Edges for which v is their responsible-neighbour are scanned, and those for which the assigned-neighbour u is marked as arrived are reported according to a search in their time stamp.

   (c) For every reported edge $e = (u, v)$,
      i. e is deleted from G_D,
      ii. Every point associated with e is deleted from S_v,
      iii. $lcount(u)$ and $rcount(v)$ are decreased by one.
2. If the arrived vertex is $u \in L$, such that $lcount(u) \neq 0$,
   (a) u is inserted to $\mathcal{L}_{\beta^*}$,
   (b) For each assigned-neighbour v such that $e = (u, v) \in E$,
      i. If τ_u is empty or $t + \alpha_e > t' + \beta_e + 1$, where t' is the newest time stamp in τ_u, then $(t + \alpha_e + 1, t + \beta_e + 1)$ is inserted to S_v.
      ii. Else, let (x, y) be the last point associated with e that was inserted to S_v, then delete (x, y) from S_v and insert $(x, t + \beta_e + 1)$ to S_v.
3. For vertices $u \in L$ arriving exactly $\alpha^* + 1$ time units before time t, such that $lcount(u) \neq 0$,
   (a) u is marked as arrived,
   (b) $t - \alpha^* - 1$ is added to τ_u.
4. For vertices $u \in L$ arriving exactly $\beta^* + M + 1$ time units before time t, such that $lcount(u) \neq 0$,
   (a) u is removed from $\mathcal{L}_{\beta^*}$,
   (b) The time stamp $t - \beta_* - M - 1$ is removed from τ_u,
   (c) For each assigned-neighbour v, such that $e = (u, v) \in E$,
      i. Let t' be the oldest time stamp in τ_u,
      ii. If τ_u is empty or $t - \beta^* - M + \beta_e < t' + \alpha_e$, then the point $(t - \beta^* + \alpha_e, t - \beta^* + \beta_e)$ is deleted from S_v,
      iii. Else, let (x', y') be the first point associated with e in S_v, delete (x', y') from S_v and insert $(t' + \alpha_e + 1, y')$ to S_v.

Theorem 3 follows.

Theorem 3. *The online DROG problem with non-uniformly bounded gap borders can be solved in $O(|D|)$ preprocessing time, $\tilde{O}(\delta(G_D) \cdot lsc + op^*)$ time per text character, where op^* is the number of new distinct dictionary patterns reported due to character arrival, and $\tilde{O}(|D| + lsc \cdot \delta(G_D)(\beta^* - \alpha^* + M) + \alpha^*)$ space.*

Proof. Correctness: The strategy of inserting a point $(x, y) = (t + \alpha_e + 1, t + \beta_e + 1)$ to S_v when u appears at time t and $e = (u, v)$, implies that when v appears at time t^*, the range query $[0, t^*][t^*, \infty]$ is performed giving all points (x, y) where $x \leq t^*$ and $t^* \leq y$. Thus, all overlapping intervals associated with edge e including point t^* are reported. Our method of uniting all these intervals to a single interval ensures a single report due to an occurrence of e and the occurrence of v at time t^*. Note that only points representing intervals related to the same edge are united to the same point, and therefore, to the same gap since several possible gaps between some vertices u and v define distinct edges. Hence, it is only necessary to consider the start point of a new interval with the endpoint of the last interval included in S_v.

Uniting overlapping intervals is sufficient for a unique report of an edge e, yet if there exist several intervals representing e in S_v due to later occurrences of u, they are deleted when e is reported, in order to avoid additional redundant report of e in future arrivals of v.

Time complexity: As the framework of [4] is used, the time complexity is in accordance with a decrease due to the fact that our total output size is limited to d. The modifications require only extra $O(1)$ operations in all cases except for the necessity to delete all points representing intervals associated with e from S_v after reporting e. However, as deletion from S_v requires the same time as the insertion to it, we account a deletion of a point by its insertion that was already accounted in the time complexity of the algorithm.

In addition, uniting points representing overlapping intervals and deleting points associated with a reported edges may decrease the time complexity of each insertion, deletion and range query, as their time complexity depends on $|S_v|$ in [18], which we use. Moreover, the range query requires $O(\frac{\log|S_v|}{\log\log|S_v|} + op)$ time, where op is the size of the output of the query, so by reducing the total output size to be at most d in all queries, we reduce the time complexity.

The space complexity is identical to that of the DMOG solution. □

3.2 DROG for Dense Graphs

Uniformly Bounded Gaps. We follow the framework of Subsection 2.2 of constructing the A_u arrays, and inserting A_u to $\mathcal{L}_{v_i}$ when u arrives, where $e = (u, v_i) \in E$. After reporting an edge, it is deleted from G_D, yet, as in the previous subsection, this deletion is not sufficient for preventing repetitious report of the edge e, since additional occurrences of v_i or of u'', where the subpattern associated with u is a suffix of the subpattern associated with u'', can yield additional reports of e. Nevertheless, by analyzing carefully the problematic scenarios this difficulty can be overcome. The modified algorithm follows.

When a vertex arrives at query time t:

1. If the arrived vertex is $v_i \in R$, such that $rcount(v_i) \neq 0$,
   - (a) For every $A_u[i] \in \mathcal{L}_{v_i}$ where $A_u[i]$ contains a pointer to edge e,
     - i. if e appeared in the appropriate time frame according the τ_u list (since the tail of $\mathcal{L}_{v_i}$ should be skipped), then **while** $e \neq null$
       - A. Edge e is reported once,
       - B. $e = next(e)$
     - ii. The edges for which v_i is their responsible-neighbour are scanned, and those for which the assigned-neighbour u is marked as arrived are reported.
   - (b) For every reported edge $e = (u, v_i)$
     - i. e is deleted from E_D,
     - ii. $A_u[i] = null$,
     - iii. $A_u[i]$ is deleted from $\mathcal{L}_{v_i}$,
     - iv. $lcount(u)$ and $rcount(v_i)$ are decreased by one.
2. If the arrived vertex is $u \in L$, such that $lcount(u) \neq 0$, then u is inserted into $\mathcal{L}_\beta$.
3. For vertices $u \in L$ arriving exactly $\alpha + 1$ time units before time t, such that $lcount(u) \neq 0$,
   - (a) For each v_i, where $A_u[i] \neq null$,
     - i. $A_u[i]$ is added to the beginning of $\mathcal{L}_{v_i}$,
     - ii. if it was already in the reporting list $\mathcal{L}_{v_i}$ then the older copy is removed from $\mathcal{L}_{v_i}$.
   - (b) u is marked as arrived,
   - (c) $t - \alpha - 1$ is added to τ_u.
4. For vertices $u \in L$ arriving exactly $\beta + M$ time units before time t, such that $lcount(u) \neq null$,

(a) u is removed from $\mathcal{L}_\beta$,
(b) The time stamp $t - \beta - M$ is removed from τ_u,
(c) For each v_i, where $A_u[i] \neq null$, $A_u[i]$ is deleted from $\mathcal{L}_{v_i}$.

This gives Theorem 4.

Theorem 4. *The online DROG problem with uniform gap borders can be solved in $O(|D|)$ preprocessing time, $O(lsc+\sqrt{lsc \cdot d}+op^*)$ time per text character, where op^* is the number of new distinct dictionary patterns reported due to the character arriving, and $O(|D| + lsc(\beta - \alpha + M) + \alpha)$ space.*

Proof. Correctness: According to the algorithm each $A_u[i]$ is uniquely added to $\mathcal{L}_{v_i}$, thus, in case edge $e = (u, v_i)$ is detected, it is reported once from the current $\mathcal{L}_{v_i}$. The deletion of $A_u[i]$ from $\mathcal{L}_{v_i}$ after reporting $e = (u, v_i)$ prevents another report of e upon additional occurrences of v_i. By adding the requirement of overriding $A_u[i]$ by *null*, it is guaranteed that no additional report of e, either by another occurrence of u or by an occurrence of u'', where the subpattern associated by u is a suffix of the subpattern associated by u''. In the latter case, if the subpattern associated by u'' occurred after the subpattern associated by u in the text, $\mathcal{L}_{v_i}$ contains $A_{u''}[i]$, thus, $e = (u, v_i)$ is included in the list derived from $A_{u''}[i]$. When v_i occurs within a proper gap from the occurrence of u'', all the edges in the list derived from $A_{u''}[i]$ are reported. Nevertheless, our modified algorithm requires that before reporting edge e, the array $A_u[i]$ associated with e is checked to be non null, if $A_u[i] = null$ the report of the edges derived from the occurrence of u'' is terminated, as the null implies that all edges associated with u as well as its suffixes and v_i were already reported if they existed. In case u'' occurred before u, the edge $e = (u, v_i)$ is reported by the list derived from $A_{u''}[i]$, yet when e is reported the algorithm deletes $A_u[i]$ from $\mathcal{L}_{v_i}$, so the occurrence of u is not checked again with regard to v_i. Hence, a single occurrence of (u, v_i) is reported.

Time Complexity: The main modifications of the original DMOG algorithm for uniformly bounded gaps are the check whether $A_u[i] \neq null$ before reporting the edge $e = (u, v_i)$, and the procedure of deleting an edge after its report. We show that both these additions to the algorithm do not increase its time complexity. First, note that every check of $A_u[i]$ can be attributed to a reported edge. Either $A_u[i]$ is found to be non *null*, due to locating u in the text and reporting (u, v_i), or $A_u[i]$ is found to be *null* due to a report of a $e'' = (u'', v_i)$, where u is a suffix of u''. Even in the latter case, the check operation can be accounted for by the report of the (u'', v_i), as a termination of a loop reporting the edges derived from the fact that some $A_u[i]$ already occurred in the report process, thus the loop reaches an already reported edge implying that all the following edges in the list were already reported. Second, a reported edge indeed induces the deletion operations required to preserve the uniqueness of the edges reported. However, the number of such operations is constant and can be accounted for the reported occurrence of an edge. Also note, that updating $A_u[i] = null$ after reporting the associated edge does not change the efficiency of the A_u arrays construction, as it implies that all edges associated with u and its suffixes and v_i were already reported, so the list starting in a predecessor of u comes to an end, as no further suffixes should be considered with node v_i. The rest of the time complexity analysis is the same as that of [4] except for the *op* factor which is replaced by a total of at most d reports due to a single report of every edge.

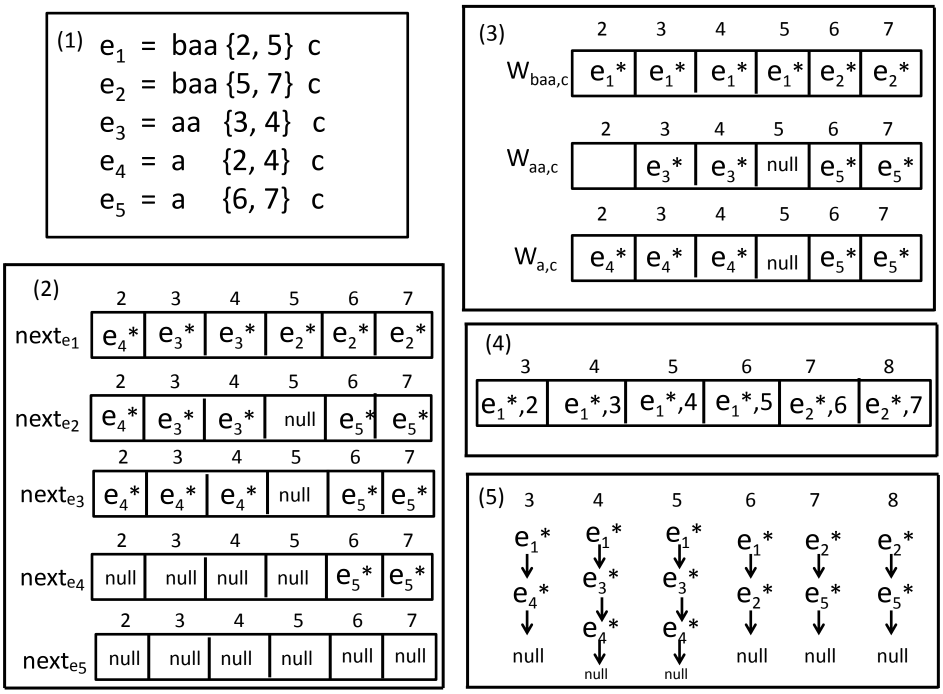
The space complexity is unchanged. □

Non-Uniformly Bounded Gaps. Following the same basic idea, after reporting an edge $e = (u, v_i)$ it is deleted from G_D and $W_{u,i}[j]$ should be assigned *null* for $\alpha_e \leq j \leq \beta_e$, so that additional occurrence of u are not inserted to the active window of v_i, $AW_{v_i}[]$. In addition, other occurrences of u already in $AW_{v_i}[]$ may derive a repetitious report of e due to an additional occurrence of v_i. In the case of uniformly bounded gaps, we handled the problem by deleting all appearances of $e = (u, v_i)$ from the data structure $\mathcal{L}_{v_i}$ of located neighbours of v_i. However, dealing with non-uniformly bounded gaps is more complicated, since when edge (u, v_i) is reported, some edges $e' = (u', v_i)$, where u' represents a subpattern that is a suffix of the subpattern represented by u, are reported too while other such e's are not reported due to different gap boundaries. Consequently, we cannot automatically delete all appearances of u from $W_{u,i}[j]$ and from the data structure $AW_{v_i}[]$ of located neighbours of v_i, since it may cause misses of occurrence of some suffixes of u, as a node occurrence implies all the suffixes of the subpattern associated by that node occurred as well, so a deletion of the occurrence of u causes the suffixes of u to have no indication of it.

For example, consider the dictionary appearing in Figure 2 and suppose e_1 was reported due to a gap of size 2 between the subpatterns. Hence e_1, e_4 are reported, as e_4 is included in $next_{e_1}[2]$ list. If e_1 and e_4 are deleted from every $AW_c[j]$ or $W_{baa,c}[j]$, it causes and implicit deletion of e_2 and e_3 from these locations, since both are present in $AW_c[4-6]$ implicitly through the $next_{e_1}[3-5]$ lists. In case c occurs again with gap of $4-6$ locations, edges e_3 and e_2 should be reported, as it may be their only occurrence in the text. We, therefore, only assign $W_{baa,c}[3]$ to *null*, and $W_{baa,c}[j \neq 3]$ is updated to the longest suffix of *baa*, u'', where $e'' = (u'', c) \in E$, e'' appears in the list emanating from $next_{e_1}[j]$ and e'' was has not been reported yet. $AW_c[j]$ is updated accordingly. Our modified algorithm follows.

When a vertex arrives at query time t, the data structures are updated as follows.

```
1. For each $v_i \in R$, such that $rcount(v_i) \neq 0$,
   (a) the active window array $AW_i$ is shifted by one position by incrementing its
       starting position in a cyclic manner,
   (b) $AW_i[\beta^* - \alpha^* + M + 1]$ is cleared (It may have been reported in the previous
       query when it was $AW_i[1]$).
2. If the arrived vertex is $v_i \in R$, such that $rcount(v_i) \neq 0$, then for every $W_{u,i}[j] \in AW_i[1]$,
   (a) $[e, j] = [(u, v_i), j](= W_{u,i}[j], j)$,
   (b) Report & Update($[e, j]$)
       i. Edge $e$ is reported,
       ii. $W_{u,i}[j] = null$,
       iii. $e' = next_e[j]$
       iv. if $e' \neq null$, then Report & Update($[e', j]$),
       v. For $f = \alpha_e$ to $\beta_e$, such that $f \neq j$, if $e' = next_e[f]$ and $e'$ was reported,
          then
          A. $next_e[f] = next_{e'}[f]$,
          B. If $W_{u,v_i}[f]$ contains a pointer to $e$ then $W_{u,v_i}[f] = next_e[f]$.
3. If the arrived vertex is $u \in L$, such that $lcount(u) \neq 0$, then
   For each $v_i \in R$ and each $j$, such that $W_{u,i}[j] \neq null$,
   (a) $(W_{u,i}[j], j)$ is inserted to the list at $AW_i[j + m_{v_i}]$, where $m_{v_i}$ is the length of
       the subpattern associated with $v_i$, (since in $j + m_{v_i}$ time units from now, if
       $v_i \in R$ arrives, the edges pointed to by $W_{u,i}[j]$ should be reported).
```



(1)
e_1 = baa {2, 5} c
e_2 = baa {5, 7} c
e_3 = aa {3, 4} c
e_4 = a {2, 4} c
e_5 = a {6, 7} c

(2)

	2	3	4	5	6	7
$next_{e_1}$	e_4*	e_3*	e_3*	e_2*	e_2*	e_2*
$next_{e_2}$	e_4*	e_3*	e_3*	null	e_5*	e_5*
$next_{e_3}$	e_4*	e_4*	e_4*	null	e_5*	e_5*
$next_{e_4}$	null	null	null	null	e_5*	e_5*
$next_{e_5}$	null	null	null	null	null	null

(3)

	2	3	4	5	6	7
$W_{baa,c}$	e_1*	e_1*	e_1*	e_1*	e_2*	e_2*
$W_{aa,c}$		e_3*	e_3*	null	e_5*	e_5*
$W_{a,c}$	e_4*	e_4*	e_4*	null	e_5*	e_5*

(4)

3	4	5	6	7	8
e_1*,2	e_1*,3	e_1*,4	e_1*,5	e_2*,6	e_2*,7

(5)
3: e_1* ↓ e_4* ↓ null
4: e_1* ↓ e_3* ↓ e_4* ↓ null
5: e_1* ↓ e_3* ↓ e_4* ↓ null
6: e_1* ↓ e_2* ↓ null
7: e_2* ↓ e_5* ↓ null
8: e_2* ↓ e_5* ↓ null

Figure 2. Consider the dictionary in (1). Its *next* arrays appear on (2), its W_u, v_i arrays appear on (3), AW_c after *baa* was found appears on (4), and the implicit lists that are saved in AW_c appear in (5). The * denotes a pointer to an edge.

Theorem 5 follows.

Theorem 5. *The online DROG problem with non-uniform gap borders can be solved in $O(|D| + d(\beta^* - \alpha^*))$ preprocessing time, $\tilde{O}(lsc + \sqrt{lsc \cdot d}(\beta^* - \alpha^* + M) + op^*)$ time per query text character, where op^* is the number of new distinct dictionary patterns reported due to the character arriving, and $\tilde{O}(|D| + d(\beta^* - \alpha^*) + \sqrt{lsc \cdot d}(\beta^* - \alpha^* + M) + \alpha^*)$ space.*

Proof. Correctness: To avoid repetitious reports of an edge e, after e is reported once due to a gap of size j', e is deleted both from G_D and from $W_{u^*,v_i}[j']$, where $(u^*, v_i) = e$ or $(u, v_i) = e$ and u represents a subpattern that is a suffix of the one represented by u^*. As described before, the deletion from $W_{u^*,v_i}[j]$, $j \neq j'$, yields an update to the array entry of the first edge to the $next_e[j]$ list that has not been reported yet, thus ensuring that a reported edge (u, v_i) is reported again. Nevertheless, unreported edges are not overlooked, even if it is of the form (u', v_i), where u' represents a suffix of the subpattern represented by u. Such edges are indicated by u in the $W_{u,v_i}[]$ array on indices where the gaps of the edges overlap.

Time Complexity: Due to the recursive nature of the updating procedure, every update of $W_{u,v_i}[j]$ requires $O(1)$ time, as the arrays of all u's are already updated, where u' represents a suffix of the subpattern represented by u. Recall from Subsection 2.2 that there are at most $\sqrt{lsc \cdot d}$ nodes currently in L, so that each report of an edge requires $O(\sqrt{lsc \cdot d}(\beta^* - \alpha^*))$ time. Such a time requirement already exists in the original *DMOG* solution. The rest of the time complexity analysis is the same as that of [4], except for the *op* factor which is replaced by a total of at most d reports due to the single report of every edge.

The space complexity is identical to that of the $DMOG$ solution. □

4 Conclusion and Open Problems

In this paper we give the first formalization of the Dictionary Recognition with One Gap (DROG) problem and give practical solutions for this problem in the online setting required for NIDS applications. Some open problems are:

- Can some of the factors in these solutions be reduced?
- Can these solutions be adapted to take care of a dictionary with subpatterns having more than one gap?

Since the DROG problem is a crucial bottleneck procedure in NIDS applications these open problems should be addressed in the future.

References

1. A. V. Aho and M. J. Corasick: *Efficient string matching: An aid to bibliographic search.* Commun. ACM, 18(6) 1975, pp. 333–340.
2. A. Amir, M. Farach, R. M. Idury, J. A. L. Poutré, and A. A. Schäffer: *Improved dynamic dictionary matching.* Inf. Comput., 119(2) 1995, pp. 258–282.
3. A. Amir, D. Keselman, G. M. Landau, M. Lewenstein, N. Lewenstein, and M. Rodeh: *Text indexing and dictionary matching with one error.* J. Algorithms, 37(2) 2000, pp. 309–325.
4. A. Amir, T. Kopelowitz, A. Levy, S. Pettie, E. Porat, and B. R. Shalom: *Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap*, in 27th International Symposium on Algorithms and Computation, ISAAC 2016, December 12-14, 2016, Sydney, Australia, 2016, pp. 12:1–12:12.
5. A. Amir, A. Levy, E. Porat, and B. R. Shalom: *Dictionary matching with one gap*, in CPM, 2014, pp. 11–20.
6. A. Amir, A. Levy, E. Porat, and B. R. Shalom: *Dictionary matching with a few gaps.* Theor. Comput. Sci., 589 2015, pp. 34–46.
7. P. Bille, I. L. Gørtz, H. W. Vildhøj, and D. K. Wind: *String matching with variable length gaps.* Theor. Comput. Sci., 443 2012, pp. 25–34.
8. P. Bille and M. Thorup: *Regular expression matching with multi-strings and intervals*, in Proc. of SODA, 2010, pp. 1297–1308.
9. G. S. Brodal and L. Gasieniec: *Approximate dictionary queries*, in Proc. of CPM, 1996, pp. 65–74.
10. N. Chiba and T. Nishizeki: *Arboricity and subgraph listing algorithms.* SIAM J. Comput., 14(1) 1985, pp. 210–223.
11. R. Cole, L. Gottlieb, and M. Lewenstein: *Dictionary matching and indexing with errors and don't cares*, in Proc. of STOC, 2004, pp. 91–100.
12. K. Fredriksson and S. Grabowski: *Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance.* Inf. Retr., 11(4) 2008, pp. 335–357.
13. T. Haapasalo, P. Silvasti, S. Sippu, and E. Soisalon-Soininen: *Online dictionary matching with variable-length gaps*, in Proc. of SEA, 2011, pp. 76–87.
14. K. Hofmann, P. Bucher, L. Falquet, and A. Bairoch: *The PROSITE database, its status in 1999.* Nucleic Acids Research, 27(1) 1999, pp. 215–219.
15. W.-K. Hon, T.-W. Lam, R. Shah, S. V. Thankachan, H.-F. Ting, and Y. Yang: *Dictionary matching with uneven gaps*, in CPM, 2015, pp. 247–260.
16. G. Kucherov and M. Rusinowitch: *Matching a set of strings with variable length don't cares.* Theor. Comput. Sci., 178(1-2) 1997, pp. 129–154.
17. M. Morgante, A. Policriti, N. Vitacolonna, and A. Zuccolo: *Structured motifs search.* Journal of Computational Biology, 12(8) 2005, pp. 1065–1082.

18. C. W. Mortensen: *Fully dynamic orthogonal range reporting on RAM.* SIAM J. Comput., 35(6) 2006, pp. 1494–1525.
19. E. W. Myers: *A four russians algorithm for regular expression pattern matching.* J. ACM, 39(2) 1992, pp. 430–448.
20. G. Myers and G. Mehldau: *A system for pattern matching applications on biosequences.* CABIOS, 9(3) 1993, pp. 299–314.
21. G. Navarro and M. Raffinot: *Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching.* Journal of Computational Biology, 10(6) 2003, pp. 903–923.
22. Verint: *Personal communication*, 2013.
23. M. Zhang, Y. Zhang, and L. Hu: *A faster algorithm for matching a set of patterns with variable length don't cares.* Inf. Process. Lett., 110(6) 2010, pp. 216–220.