Mind the Gap: Essentially Optimal Algorithms for Online Dictionary Matching with One Gap*

Amihood Amir[1], Tsvi Kopelowitz[2], Avivit Levy[3], Seth Pettie[4], Ely Porat[5], and B. Riva Shalom[6]

1 **Bar-Ilan University, Ramat Gan, Israel; and Johns Hopkins University, Baltimore, USA**
amir@cs.biu.ac.il
2 **University of Michigan, Ann Arbor, USA**
kopelot@gmail.com
3 **Shenkar College, Ramat Gan, Israel**
avivitlevy@shenkar.ac.il
4 **University of Michigan, Ann Arbor, USA**
pettie@umich.edu
5 **Bar-Ilan University, Ramat Gan, Israel**
porately@cs.biu.ac.il
6 **Shenkar College, Ramat Gan, Israel**
rivash@shenkar.ac.il

Abstract

We examine the complexity of the online Dictionary Matching with One Gap Problem (DMOG) 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 of the patterns from D that are suffixes of the text that has arrived so far, before the next character arrives. In more general versions the gap symbols are associated with *bounds* determining the possible lengths of matching strings. Online DMOG captures the difficulty in a bottleneck procedure for cyber-security, as many digital signatures of viruses manifest themselves as patterns with a single gap.

In this paper, we demonstrate that the difficulty in obtaining efficient solutions for the DMOG problem, even in the offline setting, can be traced back to the infamous 3SUM conjecture. We show a conditional lower bound of $\Omega(\delta(G_D)+op)$ time per text character, where G_D is a bipartite graph that captures the structure of D, $\delta(G_D)$ is the *degeneracy* of this graph, and op is the output size. Moreover, we show a conditional lower bound in terms of the magnitude of gaps for the bounded case, thereby showing that some known offline upper bounds are essentially optimal.

We also provide matching upper-bounds (up to sub-polynomial factors), in terms of the degeneracy, for the online DMOG problem. In particular, we introduce algorithms whose time cost depends linearly on $\delta(G_D)$. Our algorithms make use of graph orientations, together with some additional techniques. 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 are able to obtain even more efficient solutions.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Pattern matching, Dictionary matching, 3SUM, Triangle reporting

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.12

* Partially supported by ISF grant 571/14, BSF grant 2014028, and NSF grants CCF-1217338, CNS-1318294, and CCF-1514383.

1 Introduction

Understanding the computational limitations of algorithmic problems often leads to algorithms that are efficient for inputs that are seen in practice. This paper, which stemmed from an industrial-acdemic connection [31], is a prime example of such a case. We focus on an aspect of Cyber-security which is a critical modern challenge. Network intrusion detection systems (NIDS) perform protocol analysis, content searching and content 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 that one gap, however, those considered in NIDS systems typically have at most *one* gap, and are a serious bottleneck in such applications [31]. Analyzing the set of gapped patterns considered by the SNORT software rules shows that 77% of the patterns have at most one gap, and more than 44% of the patterns containing gaps have only one gap. 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 [28, 19] and has been revisited many times in the intervening years (e.g. [27, 10, 25, 9, 16, 29, 32]), in this paper we study what is apparently a mild generalization of the problem that has nonetheless resisted many researcher's attempts at finding a definitive efficient solution.

The set of d patterns to be detected, called a *dictionary*, could be quite large. While dictionary matching is well studied (see, e.g. [2, 4, 12, 5, 15]), NIDS applications motivate the *dictionary matching with one gap* problem, defined formally as follows.

▶ **Definition 1.** The *Dictionary Matching with One Gap Problem (DMOG)*, 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: All locations in T where a pattern $P_i \in D$, $1 \leq i \leq d$, ends.

In the offline DMOG problem T and D are presented all at once. We study the more practical *online* DMOG 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 the subset of patterns with a match ending at this character should be reported before the next character arrives. Three cost measures are of interest: a preprocessing time, a time per character, and a time per match reported. Online DMOG is a serious bottleneck for NIDS, though it has received much attention from both the industry and the academic community.

1.1 Previous Work

Finding efficient solutions for DMOG has proven to be a difficult algorithmic challenge as, unfortunately, little progress has been obtained on this problem 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, 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 our upper bounds are essentially optimal (assuming some popular conjectures, as described in Section 2).

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.

	Preprocessing Time	Total Query Time	Algorithm Type	Remark
[24]	none	$\tilde{O}(\|T\| + \|D\|)$	online	reports only first occurrence
[32]	$O(\|D\|)$	$\tilde{O}(\|T\| + d)$	online	reports only first occurrence
[18]	$O(\|D\|)$	$O(\|T\| \cdot lsc + socc)$	online	reports one occurrence per pattern and location
[7]	$\tilde{O}(\|D\|)$	$\tilde{O}(\|T\|(\beta - \alpha) + op)$	offline	DMOG
[20]	$O(\|D\|)$	$\tilde{O}(\|T\|(\beta^* - \alpha^*) + op)$	offline	DMOG
This paper	$O(\|D\|)$	$\tilde{O}(\|T\| \cdot \delta(G_D) \cdot lsc + op)$	online	DMOG
This paper	$O(\|D\|)$ $O(\|D\|)$	$\Omega(\|T\| \cdot \delta(G_D)^{1-o(1)} + op)$ $\Omega(\|T\| \cdot (\beta - \alpha)^{1-o(1)} + op)$	online or offline	DMOG

1.2 New Results

The DMOG problem has several natural parameters, e.g., $|D|, d$, and the magnitude of the gap. We establish almost sharp upper and lower bounds for the cases of unbounded gaps ($\alpha = 0, \beta = \infty$), uniformly bounded gaps where all patterns have the same bounds, α and β, on their gap, and the most general non-uniform gaps version, where each pattern $P_i \in D$ has its own gap bounds, α_i and β_i. We show that the complexity of DMOG actually depends on a "hidden" parameter that is a function of the *structure* of the gapped patterns. The dictionary D can be represented as a graph G_D, which is a multi-graph in the non-uniformly bounded gaps case, where vertices correspond to first or second subpatterns and edges correspond to patterns[1]. We use the notion of graph *degeneracy* $\delta(G_D)$ which is defined as follow. 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.

Vertex-triangle queries. A key component in understanding both the upper and lower bounds for DMOG is the *vertex-triangles* problem, where the goal is to preprocess a graph so that given a query vertex u we may list all triangles that contain u. The vertex-triangles problem, besides being a natural graph problem, is of particular interest here since, as will be demonstrated in Section 2, it is reducible to DMOG. Our reduction demonstrates that the complexity of the DMOG problem already emerges when all patterns are of the form of two characters separated by an unbounded gap. This simplified online DMOG problem is

[1] While it may be more natural to consider a directed or bipartite graph, the notion of degeneracy ignores directions, and so let G_D here be an undirected graph for sake of explaining the notion.

equivalent to the following *Induced Subgraph* (ISG) problem. Preprocess a directed graph $G = (V, E)$ such that given a query sequence of vertices online (these vertices need not be all of V), after vertex v_i arrives, all edges $(v_j, v_i) \in E$ with $j < i$ are reported. Notice that answering consecutive queries is done independently. Thus, characters and gapped patterns in DMOG correspond to vertices and edges in ISG, respectively. We show that vertex-triangles queries are reducible to ISG.

This reduction serves two purposes. First, in Section 2 we prove a *conditional lower bound* (CLB) for DMOG based on the 3SUM conjecture by combining a reduction from triangle enumeration to the vertex-triangles problem with our new reduction from the vertex-triangles problem to DMOG. Our lower bound states that any online DMOG algorithm with low preprocessing and reporting costs must spend $\Omega(\delta(G_D)^{1-o(1)})$ per character, assuming the 3SUM conjecture. Interestingly, the path for proving this CLB deviates from the common conceptual paradigms for proving lower bounds conditioned on the 3SUM conjecture, and is of independent interest. In particular, the common paradigm considers set-disjointness or set-intersection type problems, which correspond to edge triangle queries, while here we consider vertex-triangle queries. Moreover, our CLB holds for the *offline* case as well, and can be rephrased in terms of other parameters. For example, in the DMOG problem with uniform gaps $\{\alpha, \beta\}$, we prove that the per character cost of scanning T must be $\Omega((\beta - \alpha)^{1-o(1)})$. This gives some indication that some recent algorithms for the offline version of DMOG problem are essentially optimal ([7, 20]).

Second, in Section 3 we provide optimal solutions (under the 3SUM conjecture), up to subpolynomial factors, for ISG and, therefore, also for vertex-triangles queries, with $O(|E|)$ preprocessing time and $O(\delta(G) + op)$ time per each vertex, where op is the size of the output due to the vertex arrival. The connection between ISG and DMOG led us to extend the techniques used to solve ISG, combine them with additional ideas and techniques, thereby introduce several new online DMOG algorithms whose dependence on $\delta(G)$ is linear. Thus, graph degeneracy seems to capture the intrinsic complexity of the problem. On the other hand, the statement of our general algorithmic results is actually a bit more complicated as it depends on other parameters of the input, namely lsc, the length of the *longest suffix chain* in the dictionary, i.e., the longest sequence of dictionary subpatterns such that each is a proper suffix of the next. While the parameter lsc could theoretically be as large as d, in practice it is very small [31]. Nevertheless, we also present algorithms that in the most dense cases reduce the dependence on lsc.

Lower bounds leading to practical upper bounds. After trying to tackle the DMOG problem from the upper bound perspective, we suspected that a lower bound could be proven, and indeed were successful in showing a connection to the 3SUM conjecture. The CLB proof provides insight for the inherent difficulty in solving DMOG, but is also unfortunate news for those attempting to find efficient upper bounds. Fortunately, a careful examination of the reduction from 3SUM to DMOG reveals that the CLB from the 3SUM conjecture can be phrased in terms of $\delta(G_D)$, which turns out to be a small constant in the input instances considered by NIDS (according to an analysis of the graph created using SNORT rules) [31]. This lead to designing algorithms whose runtime can be expressed in terms of $\delta(G_D)$, and can therefore be helpful in practical settings. The following table summarizes our upper-bounds for DMOG.

The design of our algorithms stem from a solution for ISG using $O(m)$ preprocessing time and $O(\delta(G) + op)$ query time. This solution for ISG is extended to solutions for the various DMOG versions. However, since subpatterns can be suffixes of each other, up to lsc vertices

Table 2 A summary of upper bounds for DMOG described in this paper. Unbounded, uniform and non-uniform refer to the type of gap bounds under consideration. M is the maximal length of a subpattern in the dictionary D.

Gaps Type	Preprocessing Time	Query Time per Text Character	Space
unbounded	$O(\|D\|)$	$O(\delta(G_D) \cdot lsc + op)$	$O(\|D\|)$
uniform	$O(\|D\|)$	$O(\delta(G_D) \cdot lsc + op)$	$O(\|D\| + lsc(\beta - \alpha + M) + \alpha)$
non-uniform	$O(\|D\|)$	$\tilde{O}(\delta(G_D) \cdot lsc + op)$	$\tilde{O}(\|D\| + lsc(\beta^* - \alpha^* + M) + \alpha^*)$
uniform	$O(\|D\|)$	$O(lsc + \sqrt{lsc \cdot d} + op)$	$O(\|D\| + lsc(\beta - \alpha + M) + \alpha)$
non-uniform	$O(\|D\| + d(\beta^* - \alpha^*))$	$\tilde{O}(\sqrt{lsc \cdot d}(\beta^* - \alpha^* + M) + op)$	$\tilde{O}(\|D\| + d(\beta^* - \alpha^*) + \sqrt{lsc \cdot d}(\beta^* - \alpha^* + M) + \alpha^*)$

can arrive simultaneously in G_D, the time of our algorithms have a multiplicative factor of lsc. We emphasize that we are not the first to introduce the lsc factor even in solutions for simplified relaxations of the DMOG problem [18]. Also, since subpatterns may be long, we must accommodate a delay in the time a vertex corresponding to a second subpattern is treated as if it has arrived, thus inducing a minor additive space usage. Finally, in [6] we obtain more efficient bounds that depend linearly on $\sqrt{lsc \cdot d}$ when $\delta(D_G) \geq \sqrt{\frac{d}{lsc}}$, by first considering special types of graph orientations, called *threshold* orientations, and then carefully applying data-structure techniques. Notice that while in the uniformly bounded case we have $\delta(G_D) = O(\sqrt{d})$, in the non-uniform case $\delta(G_D)$ could be much higher and so these new algorithms become a vast improvement.

Paper Contributions. The main contributions of this paper are:

- Obtaining algorithms for DMOG that are asymptotically fast for practical inputs.
- Proving matching conditional lower bounds (up to sub-polynomial factors) from the 3SUM conjecture, which in particular deviate from the common paradigm of such proofs.
- Formalizing the ISG problem. This problem serves in this paper for supplying a deeper understanding of the DMOG problem, but is also of independent interest.

Paper Organization. Section 2 describes our conditional lower bounds. In Section 3 we introduce a solution for ISG, which is then extended to simplified versions of the uniformly and non-uniformly bounded DMOG problems in Sections 3.1 and 3.2. In Section 4.1, the ISG algorithms are extended to solutions for the various DMOG versions. More details and results appear in [6].

2 3SUM: Conditional Lower Bounds

In this section we prove that conditioned on the 3SUM conjecture we can prove lower bounds for the vertex-triangles problem, the ISG problem, and the (offline) unbounded DMOG problem. Since the other two versions of DMOG (uniformly and non-uniformly bounded) can solve the unbounded DMOG version, the lower bounds hold for these problems as well.

Background. Polynomial (unconditional) lower bounds for data structure problems are considered beyond the reach of current techniques. Thus, it has recently become popular to prove CLBs based on the *conjectured* hardness of some problem. One of the most popular conjectures for CLBs is that the 3SUM problem (given n integers determine if any three sum to zero) cannot be solved in truly subquadratic time, where truly subquadratic time is $O(n^{2-\Omega(1)})$ time. This conjecture holds even if the algorithm is allowed to use randomization (see e.g. [30, 1, 23, 17]). In this section we show that the infamous 3SUM problem can be reduced to DMOG, which sheds some light on the difficulty of the DMOG problem. Interestingly, our reduction does not follow the common paradigm for proving CLBs based on the 3SUM conjecture, providing a new approach for reductions from 3SUM. This approach is of independent interest, and is described next.

Triangles. Pătraşcu [30] showed that 3SUM can be reduced to enumerating triangles in a tripartite graph. Kopelowitz, Pettie, and Porat [23] provided more efficient reductions, thereby showing that many known triangle enumeration algorithms ([21, 13, 11, 22]) are essentially and conditionally optimal, up to subpolynomial factors. Hence, the offline version of triangle enumeration is well understood. The following two indexing versions of the triangle enumeration problem are a natural extension of the offline problem. In the *edge-triangles* problem the goal is to preprocess a graph so that given a query edge e all triangles containing e are listed. The vertex-triangles problem is defined above. Clearly, both these versions solve the triangle enumeration problem, which immediately gives lower bounds conditioned on the 3SUM conjecture.

The edge-triangles problem on a tripartite graph corresponds to preprocessing a family F of sets over a universe U in order to support set intersection queries in which given two sets $S, S' \in F$ the goal is to enumerate the elements in $S \cap S'$ (see [23]). Indeed, the task of preprocessing F to support set-intersection enumeration queries, and hence edge-triangles, is well studied [14, 22]. Furthermore, the set intersection problem has been used extensively as a tool for proving that many algorithmic problems are as hard as solving 3SUM [30, 1, 23]. However, the vertex-triangles problem has yet to be considered directly[2].

The Lower Bounds. We use the vertex-triangles problem in order to show that the ISG problem is hard, and thus the simplest DMOG version of (offline) unbounded setting is 3SUM-hard. Our proof begins from the conditional lower bounds for triangle enumeration introduced in [23]. The most significant conditional lower bounds that we prove are stated by the following theorems. Due to space limitations the proofs of these theorems, together with some more conditional lower bounds, are given in Appendix A. To understand the statements of the following theorems, when the total query time of an algorithm can be formulated as $O(t_q + op \cdot t_r)$ time, we say that t_q is the query time and t_r is the reporting time.

▶ **Theorem 2.** *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. For any algorithm that solves the ISG problem on a graph G with m edges, if the amortized expected preprocessing time is $O(m \cdot \delta(G)^{1-\Omega(1)})$ and the amortized expected reporting time is sub-polynomial, then the amortized expected query time must be $\Omega((\delta(G))^{1-o(1)})$.*

▶ **Theorem 3.** *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. For any algorithm that solves the DMOG problem on a dictionary D with d patterns, if the amortized expected preprocessing time is $O(|D| \cdot \delta(G_D)^{1-\Omega(1)})$ and the amortized expected reporting time is sub-polynomial, then the amortized expected query time must be $\Omega((\delta(G_D))^{1-o(1)})$.*

[2] The closely related problem of deciding whether a given vertex is contained by any triangle (a decision version) has been addressed [8].

3 The Induced Subgraph Problem

An Upper Bound via Graph Orientations. In graph orientations the goal is to *orient* the graph edges while providing some guarantee on the out-degrees of the vertices. Formally, 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 notion of graph *degeneracy* is closely related to graph orientations [3]. Chiba and Nishizeki [13] linear time greedy algorithm assigns a $\delta(G)$-orientation of G. We preprocess G using this algorithm, thereby obtaining a c-orientation with $c = \delta(G)$, and use it for solving ISG problem as follows. First, we view an orientation 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). We exploit this distinction by using the notation of an edge $e = (u, v)$ as oriented from u to v, while e could be directed either from u to v or from v to u. We say that u is *responsible* for e, and that e is *assigned* to u. Furthermore, u is a *responsible-neighbor* of v and v is an *assigned-neighbor* of u.

The Bipartite Graph. We begin by converting $G = (V, E)$ to a bipartite graph by creating two copies of V called L (the left vertices) and R (the right vertices). For every edge $(u, v) \in E$ we add an edge in 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. All edges are originally directed from L to R (before the orientation). Furthermore, each vertex in V that arrives during query time is replaced by its two copies, first the copy from R and then the copy from L. This ordering guarantees that a self loop in G is not mistakenly reported the first time its single vertex arrives. Notice that the degeneracy of G is unchanged, up to constant factors, due to this reduction. From here onwards we assume that G is already in this bipartite representation.

The unbounded case discussion and the omitted proofs appear in [6].

3.1 Uniformly Bounded Edge Occurrences

In this case, the ISG problem is restricted with two positive integer parameters α and β so an edge (v_j, v_i) can only be reported if $\alpha < i - j \leq \beta + 1$ (recall that i and j are arrival times of v_i and v_j, respectively). The interval between β time units ago and α time units ago is called the *active window*. It is maintained via a list $\mathcal{L}_\beta$ of the last β vertices. In addition, each vertex $v \in R$ maintains a *reporting list* $\mathcal{L}_v$, which is a linked list containing the responsible-neighbors of v which have appeared during the active window, without repetition. Furthermore, each vertex $u \in L$ has an ordered list of time stamps τ_u of the times u arrived in the current active window.

At query time i, $\mathcal{L}_\beta$ is updated by removing $v_{i-\beta-1}$ and inserting v_i, which is the vertex arriving at time i. If $v_{i-\beta-1} \in L$ then the time stamp of $i - \beta - 1$ is removed from $\tau_{v_{i-\beta-1}}$. In case $\tau_{v_{i-\beta-1}}$ becomes empty then we remove $v_{i-\beta-1}$ from all of the reporting lists of its assigned-neighbors.

When a vertex v_i arrives, the data structures of the vertices are updated accordingly, as follows. If $v_i \in R$,

1. The elements of the reporting list $\mathcal{L}_{v_i}$ are scanned and their edges (u, v_i) are reported according to τ_u.
2. 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.

If $v_{i-\alpha-1} \in L$,

1. $v_{i-\alpha-1}$ is marked as arrived.
2. If $\tau_{v_{i-\alpha-1}}$ was empty before the current arrival, then $v_{i-\alpha-1}$ is added to the reporting lists of its assigned neighbours.
3. $i-\alpha-1$ is added to $\tau_{v_{i-\alpha-1}}$.

▶ **Theorem 4.** *The Induced Subgraph problem with uniformly bounded edge occurrences on a graph G with m edges and n vertices can be solved with $O(m+n)$ preprocessing time, $O(\delta(G)+op)$ time per query vertex, where op is the number of edges reported at vertex arrival, and $O(m+\beta)$ space.*

3.2 Non-Uniformly Bounded Edge Occurrences

In non-uniformly bounded edge occurrences each edge $e=(v_j,v_i)$ has its own boundaries $[\alpha_e,\beta_e]$ and can only be reported if $\alpha_e < i-j < \beta_e+1$. Notice that in this case the input is a multi-graph. The active window for this ISG version is the time window between $\beta^* = \max_{e\in E}\{\beta_e\}$ and $\alpha^* = \min_{e\in E}\{\alpha_e\}$ time units ago.

Similar to Section 3.1, a dynamic list $\mathcal{L}_{\beta^*}$ of the last β^* vertices that have appeared is maintained. However, this approach of a general active window introduces a new challenge. If τ_u includes all the appearances of u within the active window, as was done in Section 3.1, when a vertex $v_i \in R$ arrives, the information in $\mathcal{L}_{v_i}$ cannot be automatically reported, as some of the appearances of nodes $u \in \mathcal{L}_{v_i}$ are not within the gaps of edge (u,v_i), thus only part of their τ_u list needs to be reported. A naive filtering considers for each $u \in \mathcal{L}_{v_i}$ a scan of τ_u and reports only time stamps j where $i-\beta_e < j < i-\alpha_e$, which sums up to $\beta^*-\alpha^*$ time per query vertex. To avoid an overhead in query time, our filtering mechanism checks all appearances of all responsible-neighbours of v_i in a batched query, where each responsible-neighbour appearance is filtered according to the edge's gaps. This is achieved by maintaining for each vertex $v \in R$ a fully dynamic data structure S_v for supporting 4-sided 2-dimensional orthogonal range reporting queries instead of $\mathcal{L}_v$. Given an $[x_0,y_0]\times[x_1,y_1]$-range, it returns the points of S_v that have (x,y) coordinates in the given range. For each responsible-neighbor $v_i \in L$ of v that arrived in the active window, where $e=(v_i,v)$, the point $(i+\alpha_e+1, i+\beta_e+1)$ is inserted into S_v, yielding the occurrences in S_v are from the "point of view" of v.

To implement S_v, we use Mortensen's data structure [26] that 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.

When a vertex v_i arrives at query time i, in addition to adding it to $\mathcal{L}_{\beta^*}$, the following happens. If $v_i \in R$,

1. A range query of $[0,i]\times[i,\infty]$ is performed over S_{v_i}. The edges representing the range output are reported.
2. 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 according to a search in their time stamp.

If $v_{i-\alpha^*-1} \in L$,

1. $v_{i-\alpha^*-1}$ is marked as arrived.
2. For each assigned-neighbour v, such that $e=(v_{i-\alpha^*-1},v)$, $(i-\alpha^*+\alpha_e, i-\alpha^*+\beta_e)$ is inserted to S_v.
3. $i-\alpha^*-1$ is added to $\tau_{v_{i-\alpha^*-1}}$.

▶ **Theorem 5.** *The Induced Subgraph problem with non-uniformly bounded edge occurrences on a graph G with m edges and n vertices can be solved with $O(m+n)$ 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.*

4 Solving DMOG

4.1 DMOG via Graph Orientations

When extending ISG to online DMOG, the longer subpatterns introduce new challenges that need to be addressed. It is helpful to still consider the bipartite graph presentation of the DMOG instance, where vertices correspond to subpatterns and edges correspond to patterns. The algorithms from Section 3 are used as basic building blocks in our algorithms for DMOG by treating a subpattern arriving as the vertex arriving in the appropriate graph, while addressing the difficulties that arise from subpatterns being arbitrarily long strings.

First, a mechanism for determining when a subpattern arrives is needed. One way of doing this is by using the the Aho-Corasick (AC) Automaton [2], using a standard binary encoding technique so that each character costs $O(\log|\Sigma|)$ worst-case time. For simplicity we assume that $|\Sigma|$ is constant. However, while in the ISG problem each character corresponds to the arrival of at most one subpattern, in the DMOG with unbounded gaps each arriving character may correspond to several subpatterns which all arrive at once, since a subpattern could be a proper suffix of another subpattern. We, therefore, phrase the complexities of our algorithms in terms of lsc, which is the maximum number of vertices in the bipartite graph that arrive due to a character arrival. This induces a multiplicative overhead of at most lsc in the query time per text character relative to the time used by the ISG algorithms.

Finally, there is an issue arising from subpatterns no longer being of length one, which for simplicity we first discuss this in the unbounded case. When $u \in L$ arrives and it has an assigned vertex $v \in R$ where m_v is the length of the subpattern associated with v, then we do not want to report the edge (u, v) until at least $m_v - 1$ time units have passed, since the appearance of the subpattern of v should not overlap with the appearance of the subpattern of u. Similarly, in the bounded case, we must delay the removal of u from $\mathcal{L}_v$ by at least $m_v - 1$ time units. Notice that if we remove u 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. We, therefore, delay the removal of u by $M - 1$ time units, where M is the length of the longest subpattern that corresponds to a vertex in R. This solves the issue of synchronization, however, some of the reporting lists now have elements that should not be reported. Nevertheless, we can spend time in a reporting list that corresponds to the size of the output using standard list and pointer techniques.

Combining these ideas with the algorithms in Section 3 gives Theorems 6, 7 and 8.

▶ **Theorem 6.** *The DMOG problem with one gap and unbounded gap borders can be solved with $O(|D|)$ 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 $O(|D|)$ space.*

▶ **Theorem 7.** *The DMOG problem with uniformly bounded gap borders can be solved such that dictionary patterns are reported online 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.*

▶ **Theorem 8.** *The DMOG problem with non-uniformly bounded gap borders can be solved such that dictionary patterns are reported online in: $O(|D|)$ preprocessing time, $\tilde{O}(\delta(G_D)\cdot$*

$lsc + op$) time per text character, where op is the number of patterns that are reported due to the character arriving, and $\tilde{O}(|D| + lsc \cdot \delta(G_D)(\beta^* - \alpha^* + M) + \alpha^*)$ space.

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 434–443, 2014.

2 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.

3 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

4 Amihood Amir, Martin Farach, Ramana M. Idury, Johannes A. La Poutré, and Alejandro A. Schäffer. Improved dynamic dictionary matching. *Inf. Comput.*, 119(2):258–282, 1995.

5 Amihood Amir, Dmitry Keselman, Gad M. Landau, Moshe Lewenstein, Noa Lewenstein, and Michael Rodeh. Text indexing and dictionary matching with one error. *J. Algorithms*, 37(2):309–325, 2000. doi:10.1006/jagm.2000.1104.

6 Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the gap. *CoRR*, abs/1503.07563, 2015.

7 Amihood Amir, Avivit Levy, Ely Porat, and B. Riva Shalom. Dictionary matching with one gap. In *CPM*, pages 11–20, 2014.

8 Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8(1):69–94, 2012.

9 Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and David Kofoed Wind. String matching with variable length gaps. *Theor. Comput. Sci.*, 443:25–34, 2012.

10 Philip Bille and Mikkel Thorup. Regular expression matching with multi-strings and intervals. In *Proc. of SODA*, pages 1297–1308, 2010.

11 A. Bjørklund, R. Pagh, V. Vassilevska Williams, and U. Zwick. Listing triangles. In *Proceedings 41st Int'l Colloquium on Automata, Languages, and Programming (ICALP (I))*, pages 223–234, 2014.

12 Gerth Stølting Brodal and Leszek Gasieniec. Approximate dictionary queries. In *Proc. of CPM*, pages 65–74, 1996.

13 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985. doi:10.1137/0214017.

14 Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010. doi:10.1016/j.tcs.2010.06.002.

15 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *Proc. of STOC*, pages 91–100, 2004.

16 Kimmo Fredriksson and Szymon Grabowski. Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance. *Inf. Retr.*, 11(4):335–357, 2008.

17 A. Grønlund and S. Pettie. Threesomes, degenerates, and love triangles. *CoRR*, abs/1404.0799, 2014.

18 Tuukka Haapasalo, Panu Silvasti, Seppo Sippu, and Eljas Soisalon-Soininen. Online dictionary matching with variable-length gaps. In *Proc. of SEA*, pages 76–87, 2011.

19 Kay Hofmann, Philipp Bucher, Laurent Falquet, and Amos Bairoch. The PROSITE database, its status in 1999. *Nucleic Acids Research*, 27(1):215–219, 1999.

20 Wing-Kai Hon, Tak-Wah Lam, Rahul Shah, Sharma V. Thankachan, Hing-Fung Ting, and Yilin Yang. Dictionary matching with uneven gaps. In *CPM*, pages 247–260, 2015.

21 A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

22 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Dynamic set intersection. In *WADS*, 2015.

23 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, 2016.

24 Gregory Kucherov and Michaël Rusinowitch. Matching a set of strings with variable length don't cares. *Theor. Comput. Sci.*, 178(1-2):129–154, 1997.

25 Michele Morgante, Alberto Policriti, Nicola Vitacolonna, and Andrea Zuccolo. Structured motifs search. *Journal of Computational Biology*, 12(8):1065–1082, 2005.

26 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM J. Comput.*, 35(6):1494–1525, 2006.

27 Eugene W. Myers. A four russians algorithm for regular expression pattern matching. *J. ACM*, 39(2):430–448, 1992.

28 G. Myers and G. Mehldau. A system for pattern matching applications on biosequences. *CABIOS*, 9(3):299–314, 1993.

29 Gonzalo Navarro and Mathieu Raffinot. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. *Journal of Computational Biology*, 10(6):903–923, 2003. doi:10.1089/106652703322756140.

30 Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *STOC*, pages 603–610. ACM, 2010. doi:10.1145/1806689.1806772.

31 Verint. Personal communication, 2013.

32 Meng Zhang, Yi Zhang, and Liang Hu. A faster algorithm for matching a set of patterns with variable length don't cares. *Inf. Process. Lett.*, 110(6):216–220, 2010. doi:10.1016/j.ipl.2009.12.007.

A Full Details for Section 2

▶ **Theorem 9** ([23][3]). *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. Then for any constant $0 < x < 1/2$, any algorithm for enumerating all triangles in a graph G with m edges, $\Theta(m^{1-x})$ vertices, and $\hat{d} = \delta(G) = \Theta(m^x)$, where $\hat{d}$ is the average degree of a vertex in G, must spend $\Omega(m \cdot \delta(G)^{1-o(1)})$ expected time.*

▶ **Theorem 10.** *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. For any algorithm that solves the vertex-triangles problem on a graph G with m edges, if the amortized expected preprocessing time is $O(m \cdot \delta(G)^{1-\Omega(1)})$ and the amortized expected reporting time is sub-polynomial, then the amortized expected query time must be at least $\Omega((\hat{d} \cdot \delta(G))^{1-o(1)})$, where $\hat{d}$ is the degree of the queried vertex.*

Proof. We reduce the triangle enumeration problem considered in Theorem 9 to the vertex-triangles problem. We preprocess G and then answer vertex-triangles queries on each of the m^{1-x} vertices thereby enumerating all of the triangles in G. If we assume a sub-polynomial reporting time, then by Theorem 9 either the preprocessing takes $\Omega(m \cdot \delta(G)^{1-o(1)})$ time or each query must cost at least $\Omega(\frac{m \cdot \delta(G)^{1-o(1)}}{m^{1-x}}) = \Omega((m^x \delta(G))^{1-o(1)}) = \Omega((\hat{d} \cdot \delta(G))^{1-o(1)})$ time. ◀

We are now ready to prove Theorems 2 and 3.

Proof of Theorem 2 and Theorem 3. We reduce the vertex-triangles problem considered in Theorem 10 to ISG as follows. We preprocess the graph G for ISG queries. Now, to

[3] The actual statement in [23] refers to the arboricity of G instead of the degeneracy of G. However, both terms are the same, up to a factor of 2.

answer a vertex-triangle query on some vertex u, we input all of the neighbors of u into the ISG algorithm. Thus, there is a one-to-one correspondence between the edges reported by the ISG algorithm and the triangles in the output of the vertex-triangles query. Since each vertex-triangle query must cost $\Omega(\hat{d} \cdot \delta(G)^{1-o(1)})$ amortized expected time then the amortized expected time spent for each of the $\hat{d}$ neighbors of u must be at least $\Omega(\delta(G)^{1-o(1)})$ amortized expected time. Since ISG is a special case of DMOG, and given Theorem 2, the proof of Theorem 3 follows directly. ◀

▶ **Theorem 11.** *Assume 3SUM requires $\Omega(n^{2-o(1)})$ expected time. For any algorithm that solves the uniformly bounded DMOG problem on a dictionary D with d patterns, if the amortized expected preprocessing time is $O(|D| \cdot \delta(G_D)^{1-\Omega(1)})$ and the amortized expected reporting time is sub-polynomial, then the amortized expected time spent on each text character must be at least $\Omega((\beta - \alpha)^{1-o(1)})$.*

Proof. The proof is similar to the proofs of Theorems 2 and 3. First, we convert the input graph G of the vertex-triangles problem to a tripartite graph G_T by creating three copies of the vertices V_1, V_2, V_3 and for each edge (u, v) in G we add 6 edges to G_T between all possible copies of u and v. We also add a dummy vertex to G_T with degree 0. Each triangle in G corresponds to a constant number of triangles in G_T. Let α be any positive integer and let $\beta = \alpha + 2\hat{d}$. We use ISG to solve vertex-triangles queries in Theorem 2, but we only ask queries on the neighbors of vertices in V_1 in a specially tailored way as follows. We first list the neighbors of u from V_2, followed by α copies of the dummy vertex, and then list the neighbors from V_3. From the construction of the tripartite graph and the input to the ISG algorithm, two vertices of an edge that is part of the output of the ISG algorithm must be separated in the input list by at least α vertices, and by at most the length of the list which is β. Thus, the time spent on each vertex must be at least $\Omega(\delta(G)^{1-o(1)}) = \Omega((m^x)^{1-o(1)}) = \Omega((\beta - \alpha)^{1-o(1)})$ amortized expected time. Since ISG is a special case of DMOG, the theorem follows directly. ◀