JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 23, Number 3, 2016

Pp. 165–179
DOI: 10.1089/cmb.2015.0168

Algorithms for Regular Tree Grammar Network Search and Their Application to Mining Human–viral Infection Patterns

ILAN SMOLY,[1,*] AMIR CARMEL,[1,*] YONAT SHEMER-AVNI,[2] ESTI YEGER-LOTEM,[3,#] and MICHAL ZIV-UKELSON[1,#]

ABSTRACT

Network querying is a powerful approach to mine molecular interaction networks. Most state-of-the-art network querying tools either confine the search to a prespecified topology in the form of some template subnetwork, or do not specify any topological constraints at all. Another approach is grammar-based queries, which are more flexible and expressive as they allow for expressing the topology of the sought pattern according to some grammar-based logic. Previous grammar-based network querying tools were confined to the identification of paths. In this article, we extend the patterns identified by grammar-based query approaches from paths to trees. For this, we adopt a higher order query descriptor in the form of a regular tree grammar (RTG). We introduce a novel problem and propose an algorithm to search a given graph for the *k* highest scoring subgraphs matching a tree accepted by an RTG. Our algorithm is based on the combination of dynamic programming with color coding, and includes an extension of previous k-best parsing optimization approaches to avoid isomorphic trees in the output. We implement the new algorithm and exemplify its application to mining viral infection patterns within molecular interaction networks. Our code is available online.

Key words: algorithms, computational molecular biology, phylogenetic trees, RNA secondary structure, sequence analysis.

1. INTRODUCTION

MOLECULAR INTERACTION NETWORKS HAVE BECOME a prominent resource for inferring protein functions, detecting cellular processes, predicting disease pathways, and more (Barabási et al., 2011). A powerful approach to mine these networks is via network querying: Given a query subnetwork, network querying tools identify instances within the network that match the query. Previous network querying tools mostly support queries in the form of a template subnetwork that is confined to a specific topology, such as

Departments of [1]Computer Science, [2]Virology, and [3]Clinical Biochemistry and Pharmacology, Ben-Gurion University of the Negev, Beer-Sheva, Israel.

A preliminary version of this work appeared in the proceeding of WABI 2015 conference.

*These authors contributed equally to the paper.

#These authors contributed equally to the paper.

paths (Shlomi et al., 2006), trees (Pinter et al., 2005; Scott et al., 2006) or graphs with bounded tree-width (Dost et al., 2008; Scott et al., 2006). Other approaches, denoted topology free, support queries in the form of a set of colors, and seek in the network vertices with matching colors that can be connected by a spanning tree (Bruckner et al., 2010; Lacroix et al., 2006).

However, the diversity in biology, where proteins may act in a variety of strategies for the same function, calls for a more flexible, yet expressive, type of query descriptor. Here, we focus on a third approach to network querying, denoted grammar-based queries. Such queries are given a grammar as input, and seek in the network subgraphs accepted by the grammar. On one hand, grammar-based queries are flexible as they provide the ability to define various topologies in a single query. On the other hand, they still have the power to impose some constraints on the topology of the sought patterns, by expressing them as a set of logic rules. Previous works on grammar-based network querying defined queries via string grammars, including regular expressions (Fan et al., 2011; Koschmieder and Leser, 2012) and context-free grammars (Sevon and Eronen, 2008), and identified paths in graphs. In this work, we extend the scope of the grammars to regular tree grammars (RTGs), and the scope of the identified subgraphs from paths to trees.

Note that even the most basic problem variant, that of finding a path in a network based on a regular grammar query descriptor, is generally intractable (Mendelzon and Wood, 1995). Thus, previous grammar-based search approaches either bounded the size of the sought paths (Fan et al., 2011) or applied inadmissible heuristics such as seed-and-extend (Koschmieder and Leser, 2012). In our solution, we also bound the size of the sought subtrees.

We introduce a new problem and propose a novel algorithm to search a given graph for the k highest scoring subgraphs of size bounded by m, matching a tree accepted by a regular tree grammar (RTG). Our proposed approach is based on a two-stage dynamic programming algorithm combining RTG k-best parsing with network search. One challenge encountered here is that RTGs recognize ordered trees, while the subtrees in the graph to be searched are unordered. We handle this by incorporating redundancy-avoidance logic and show how to extend an efficient k-best parsing algorithm (Huang and Chiang, 2005) to support it. Also, in order to support the dynamic-programming-based parsing algorithm employed in this article, we use a normalized form of the grammar, similarly to the way the CYK algorithm parses context-free string grammars in Chomsky normal form (CNF) (Lange and Leiß, 2009). For this purpose, we harness a previously known binarization approach to RTG parsing (Martens and Niehren, 2005). Finally, to handle the exponentially growing search space, and to avoid cycles, we employ color-coding (Alon et al., 1995), which was also used by other network querying tools (Bruckner et al., 2010; Dost et al., 2008; Scott et al., 2006; Shlomi et al., 2006), and show how to probabilistically bound the number of color-coding iterations needed to compute the k-best subtrees for a given error threshold.

We implement the proposed algorithm in a software package, denoted RTGnet. Due to space constraints, additional data, including some of the figures and extended results, are found in the Supplementary Material, available online at www.libertpub.com/cmb

The rest of the article proceeds as follows. In section 2, we give some preliminaries on RTGs. In section 3, we present and define the RTG network parse-and-search problem. In section 4, we describe our algorithm for solving this problem. Finally, in section 5, we exemplify the biological relevance of our approach via applications to mining viral infection patterns within molecular interaction networks.

2. REGULAR TREE GRAMMARS AND CURRY-ENCODING

The patterns we seek are ordered rooted trees, to be recognized by regular tree grammars (Comon et al., 2007).

Definition 1. *A regular tree grammar (RTG) is a tuple $A=\{N, \Sigma, P, S\}$, where N is a finite set of nonterminal symbols, Σ is an alphabet of terminal symbols, S is the initial nonterminal, and P is a finite set of productions of type $X \rightarrow a(R)$, where R is a regular expression over N and $X \in N$, $a \in \Sigma$.*

Each production rule $X \rightarrow a(R)$ defines a subtree in which the root is some vertex labeled a. Denote by u the root vertex, then R expresses the nonterminal symbols that are used to recursively generate the child subtrees of u. If $R=\epsilon$, then u is a leaf. Let $T(\Sigma)$ denote the set of all ordered, rooted trees with vertex labels in Σ. An ordered, rooted, vertex-labeled tree $\tau \in T(\Sigma)$ is accepted by A if there is a chain of consecutive

derivations of production rules from P that starts in S and generates τ. The language $L(A) \subseteq T(\Sigma)$ is the set of all trees accepted by A. Regular tree languages (RTL) is the class of tree languages that are generated by RTGs.

Note that RTGs are closely related to context free grammars (CFGs): (1) Any production rule in an RTG can be written as a production rule in a CFG (with special nonterminal symbols for brackets). (2) For any CFG G, the derivation trees producing all words $w \in L(G)$ can be defined using an RTG, simply by encapsulating each rule with brackets and adding a dummy symbol as the root. Furthermore, following these two principles, the class of languages defined by the *yield* of RTGs (i.e., the sequence of labels of the leafs of the trees accepted by an RTG) is equivalent to the class of languages defined by CFGs.

RTGs are classically categorized as either ranked or unranked. Ranked RTGs generate trees for which there is a global bound on the number of children each vertex may have. Unranked RTGs, which are the more general case supported by our framework, have no such bound.

For our dynamic-programming-based parsing algorithm, we harness a recursive deterministic binarization approach, denoted *currying* (Carme et al., 2004) (see Figure 1). This approach encodes an unranked tree τ into a ranked binary tree, denoted $curry(\tau)$, in linear time, as follows. Given a tree $\tau = a(\tau_1,\ldots,\tau_n) \in T(\Sigma)$, $curry(\tau) = @(curry(\tau'), curry(\tau_n))$ with $\tau' = a(\tau_1,\ldots,\tau_{n-1})$. If $n=0$ then $curry(\tau) = a$. The *extension operator* @ is used to denote the labels of the internal vertices of the binarized trees, while their leaves correspond to the vertices of the original tree. Correspondingly, the given RTG A is also converted in linear time into a curry-encoded RTG (defined below), $curry(A)$, that accepts the curry encodings of the trees in $L(A)$, such that the bijection $curry(L(A)) = L(curry(A))$ is obeyed (Carme et al., 2004; Comon et al., 2007).

Figure 1 examplifies the currying procedure for a given RTG A (Fig. 1A). Production-rule number 3 of RTG A derives a regular expression equivalent to the automaton shown in Figure 1B. The curry-encoded RTG $curry(A)$ (Fig. 1C), is obtained via the decomposition of the automaton such that each of the new binary production-rules in $curry(A)$ corresponds to a specific transition in the automaton. A tree that is accepted by A, denoted $\tau \in L(A)$ is shown in Figure 1D. This tree can be currified into a binary tree $curry(\tau) \in L(curry(A))$ (Fig. 1E), where $curry(\tau)$ corresponds to the derivation-tree obtained by parsing τ according to $curry(A)$.

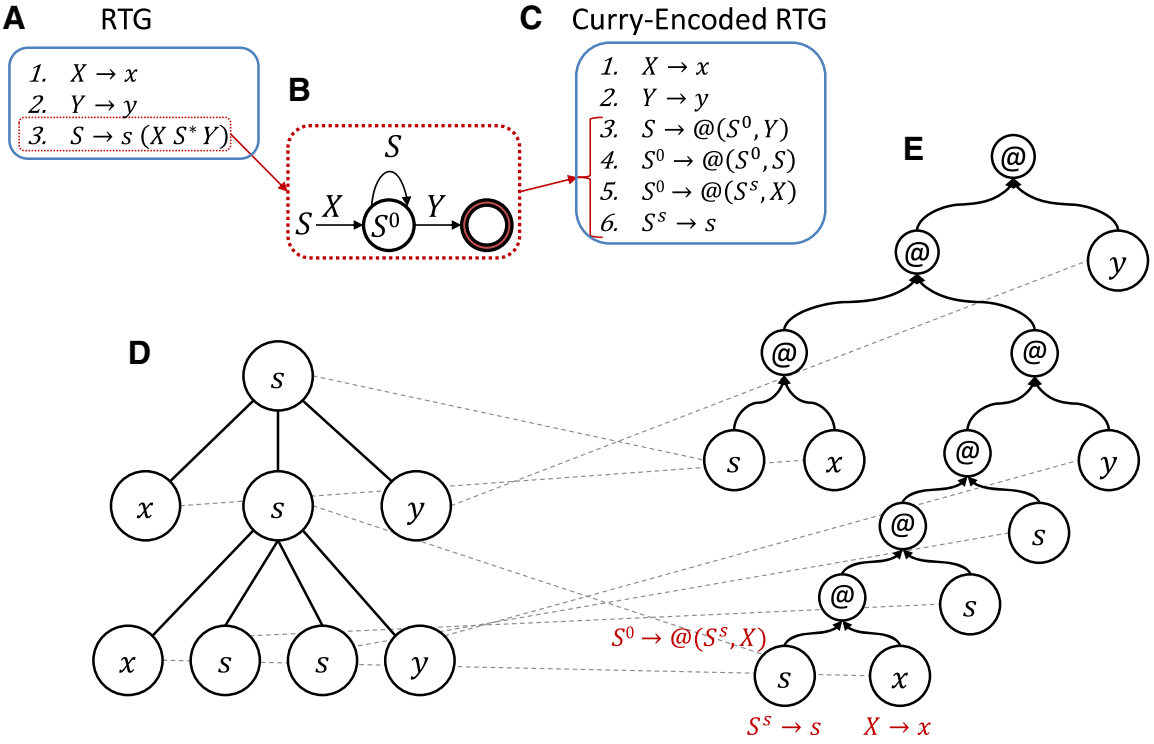

FIG. 1. (**A–C**) Curryfication of a regular tree grammar. (**D–E**) Curryfication of an ordered tree accepted by the grammar.

Definition 2. *A curry-encoded regular tree grammar is a tuple* $A = \{N, \Sigma, P, S\}$ *where* N *is a finite set of nonterminal symbols,* Σ *is an alphabet of terminal symbols,* S *is the initial nonterminal, and* P *is a finite set of productions of type* $X \rightarrow @(Y, Z)$ *or* $X \rightarrow a$, *where* $X, Y, Z \in N$, $a \in \Sigma$, *and* $@ \notin \Sigma$.

Based on this, in the rest of this article we assume that the input grammar A is in curry-encoded form, and denote by g the number of its production rules. The binarization of the candidate trees will be done during the parsing process. Also, for convenience, we write $X \rightarrow Y\ @Z$ when $X \rightarrow @(Y, Z)$.

3. THE RTG NETWORK PARSE-AND-SEARCH PROBLEM

Given an RTG $A = \{N, \Sigma, P, S\}$ of size g, a directed vertex-labeled graph $G = (V, E)$, and an integer parameter m, let $T[G, A, m]$ be a set of labeled trees, such that for any tree $\tau \in T[G, A, m]$: (1) $\tau \in L(A)$, (2) $|\tau| \leq m$, and (3) τ is a subtree in G. Let $s(\tau) \in \Re$ denote a predefined monotonic scoring scheme (definition will follow) on τ, where $\Re$ denotes the real numbers. Let $Max_s^k(T[G, A, m])$ denote the subset of k-best scoring subtrees in $T[G, A, m]$ according to s. Our search is based on the following optimization problem.

Problem 1 (RTG network parse-and-search problem). Given G, A, k, m, s as defined above, the RTG network parse-and-search problem is to compute $Max_s^k(T[G, A, m])$.

The problem of deciding whether a given ordered rooted tree is accepted by a given RTG can be solved in polynomial time, using a bottom-up tree automaton that starts at the leaves of the tree and moves upward, associating along a run a state with each subtree inductively (Comon et al., 2007). However, solving Problem 1 entails extending RTG parsing to handle, as input, a *directed graph* rather than an *ordered rooted tree* and to support a local RTG derivation-tree search rather than just tree parsing.

Furthermore, the extension from ordered tree parsing to network parse-and-search raises several other challenges: First, the number of potential subtrees grows exponentially with the size of the sought subnetworks, yielding a hard search problem. Second, RTGs recognize ordered trees, while the subtrees in the graph to be searched are unordered. This is problematic when aiming to report the k-best results, as several copies of the same top-scoring subtree could potentially be identified and reported several times per RTG query, where two repeated copies could differ by branch orderings, yielding an incorrect output. A third challenge to be handled is that the input network could contain cycles, while the patterns defined by RTG queries are trees. Cycles should be avoided as they could cause wasteful looping in densely connected subnetworks such as protein complexes, masking off the significant noncyclic (tree) patterns that we are searching for.

Our proposed approach is based on iterative randomized color-coding (Alon et al., 1995), which allows us to handle the exponentially growing search space, and to avoid cycles. Color-coding is a probabilistic approach that bounds the error expectancy by executing multiple graph coloring iterations. In each iteration, every vertex in the queried network is assigned a color chosen randomly out of m colors. The colored network is then searched for colorful subgraphs in which each color appears exactly once. Thus, instead of maintaining an enumeration space of size $\binom{n}{m}$, due to all possible selections of subsets of size m from among n vertices, one can maintain a search space of size 2^m enumerating sets of m distinct colors in considerably lower complexity.

The following theorem and proof show how to bound the number of required color-coding iterations in our search problem, which computes the k-best results.

Theorem 1. *For any* $\epsilon > 0$, *after* $e^m \ln\left(\frac{k}{\epsilon}\right)$ *iterations, the output list contains all the k-best subtrees with error probability* $\leq \epsilon$.

Proof. For $1 \leq i \leq k$, let V_i be the set of nodes of the $i'th$ best subtree. For any i, $|V_i| \leq m$, thus V_i is colorful in iteration j, with probability $\geq \frac{m!}{m^m} \geq e^{-m}$ (since $|V_i|!$ is the number of legal colorings, $|V_i|^{|V_i|}$ is the total number of colorings, and $|V_i| \leq m$). Denote by A_i^j the event that V_i is not colorful in iteration j, then $\Pr\left[A_i^j\right] \leq 1 - e^{-m}$. Note that if V_i is colorful in at least one iteration, then V_i is part of the output. Hence, the k subtrees are not in the output list if and only if there exists some tree V_i that is not colorful in any iteration. Denote by A_i the event that V_i is not colorful in any iteration (i.e., $A_i = \cap_{j=1}^t A_i^j$), then: $\Pr[Failure] = \Pr[A_1 \cup A_2 \cup \ldots \cup A_k] \leq \sum_{i=1}^k \Pr[A_i]$. Denote by t the number of color coding iterations.

$\Pr[A_i] = \Pr[A_i^1 \cap A_i^2 \cap \ldots \cap A_i^t] = \prod_{j=1}^{t} \Pr[A_i^j] \leq \prod_{j=1}^{t} (1-e^{-m}) = (1-e^{-m})^t \leq (e^{-(e^{-m})})^t$ Then for $t > e^m \ln\left(\frac{k}{\epsilon}\right)$, $\Pr[Failure] \leq \sum_{i=1}^{k} \Pr[A_i] \leq \sum_{i=1}^{k} (e^{-(e^{-m})})^t = k \cdot e^{-e^{-m} \cdot t} < k \cdot e^{-e^{-m} \cdot e^m \ln\left(\frac{k}{\epsilon}\right)} = \epsilon$. ■

Based on this, the focus of the next section will be the work per each color-coding iteration, which is formalized as follows. Let $C : V \rightarrow \{1,\ldots,m\}$ be a coloring function. We define $T^C[G, A, m]$ to be a subset of $T[G, A, m]$, such that for any tree $\tau \in T^C[G, A, m]$, each vertex of τ is distinctly colored according to C (i.e., τ is colorful). Let $Max_s^k(T^C[G, A, m])$ denote the subset of k-best scoring subtrees in $T^C[G, A, m]$ according to s.

Problem 2 (RTG colored network parse-and-search problem). Given G, A, k, m, s, C as defined above, the RTG colored network parse-and-search problem is to compute $Max_s^k(T^C[G, A, m])$.

4. ALGORITHMS FOR SOLVING PROBLEM 2

In this section we describe a two-stage dynamic programming algorithm for solving Problem 2, based on the formulation of the algorithm as an instance of optimal derivation in the ordered acyclic hypergraph framework (Finkelstein and Roytberg, 1993; Patro and Kingsford, 2013; Ponty and Saule, 2011). This framework offers a convenient formalization of dynamic programming algorithms and the book-keeping of the computations involved. Furthermore, it allows us to smoothly extend previous algorithms for k-best parsing, which were also formulated in the same framework (Huang and Chiang, 2005).

An overall illustration of our algorithm for solving Problem 1, by solving Problem 2 within color-coding iterations, is shown in Figure 2. Here, the input consists of a vertex-labeled graph G (Fig. 2A), representing a biological network, a parameters m that limits the number of vertices in the sought trees (Fig. 2B), an RTG A (Fig. 2C) that is transformed into a curry-encoded RTG, and a parameter k specifying the number of top-scoring results to compute (Fig. 2D).

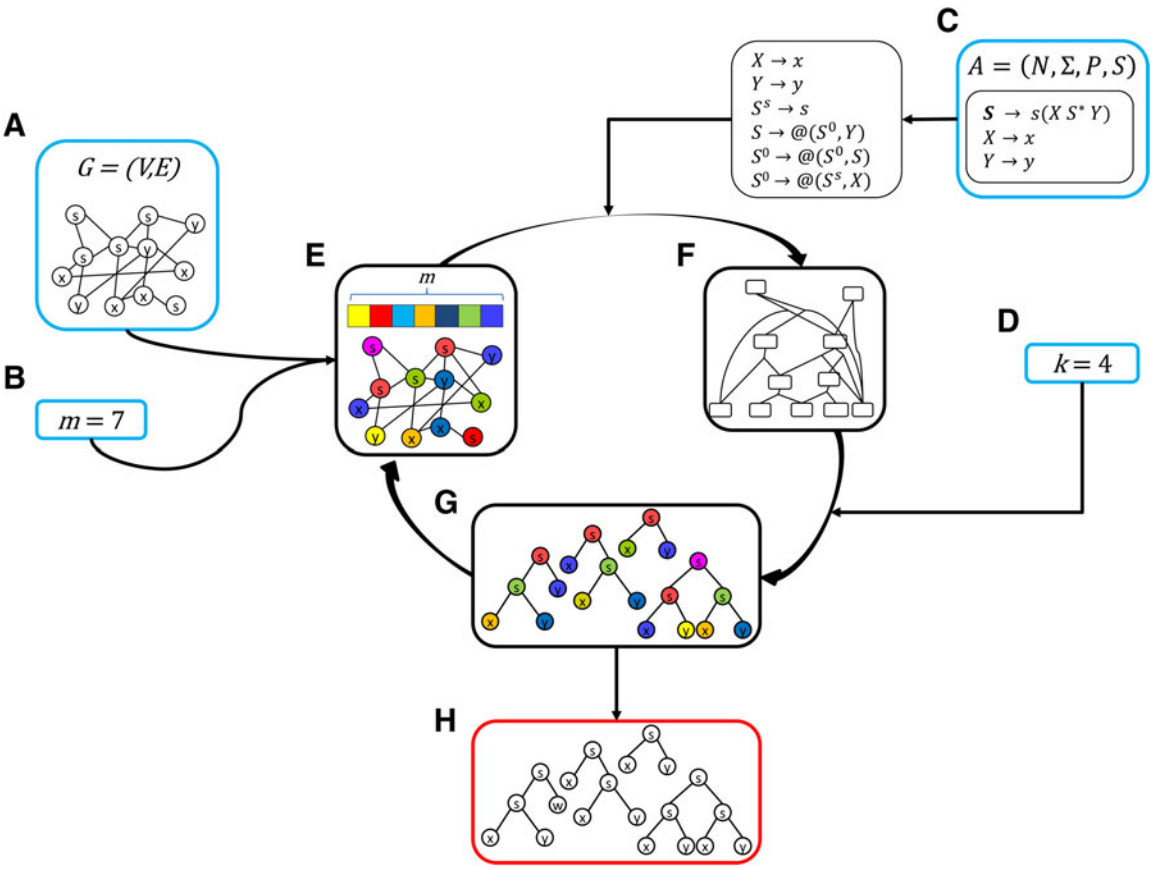

FIG. 2. The workflow of the algorithm for solving Problem 1.

Each color-coding iteration begins with a randomized coloring of the vertices of G (Figure 2E) with m distinct colors. Then, within each iteration, we apply a two-stage algorithm. In the first stage (section 4.1), the curryfied representations of all the candidate subtrees within the graph G are constructed and encoded in an ordered acyclic hypergraph H (Figure 2F). Then, in the second stage (section 4.2), the hypergraph H is processed again to compute the k-best scoring subtrees (Figure 2G). Across color-coding trials, we maintain a global sorted list of size k, which stores the trees corresponding to the k-best derivations found so far (Figure 2H). After each color-coding iteration, this list is updated with the k-best derivations yielded by that iteration. This is done in $O(mk \log k)$ time and $O(mk)$ space, that is, without increasing the overall time and space complexities of the algorithm that we propose for solving Problem 2.

4.1. Stage 1: hypergraph construction and its optimal derivation

For the hypergraph construction, we propose a bottom-up algorithm that iteratively merges the candidate subtrees from smaller subtrees, supporting the following conditions simultaneously: (1) the two subtrees implement a production-rule from the input grammar, (2) the two subtrees can be connected by a single edge, and (3) the two subtrees consist of distinctly colored vertices. Our proposed algorithm is formulated in the ordered hypergraph framework as follows.

A directed ordered hypergraph is a pair $H=(V_H, E_H)$, where V_H is the set of hypernodes, and E_H is the set of ordered, directed hyperedges. Each hyperedge e is a pair $\langle head(e), tail(e)\rangle$, where $head(e)$ is a hypernode called the *head* of the hyperedge and $tail(e)$ is an ordered sequence of hypernodes, called the *tail* of the hyperedge. Denote by $t_i(e)$ the *i'th* element of the tail of e. Note that the hyperedges in our hypergraph are all binary. Thus, henceforth we write a hyperedge with head h and tail t_1, t_2 as $(h \leftarrow \langle t_1, t_2\rangle)$.

Given an RTG $A=\{N, \Sigma, P, S\}$, a directed graph $G=(V, E)$, a bound m on the size of the sought trees, and a coloring function C of vertices in the input graph. We define the hypergraph in our framework as follows:

Definition 3. *A* hypernode $x \in V_H$ *is of the form* (X, u, q)*, where* $X \in N$*,* $u \in V$*,* $q \subseteq \{1,\ldots,m\}$ *and* $C(u) \in q$*.*

A hypernode $x=(X, u, q)$ represents a class of subtrees T_x from G, such that for each $\tau \in T_x$, (1) τ is rooted in u, (2) τ is composed of vertices with distinct colors from q, and (3) τ matches (in terms of both topology and vertex labels) a tree obtained by consecutive derivations of production-rules from P, that start with the nonterminal X.

Definition 4. *A* hyperedge $e \in E_H$ *is of the form* $(x \leftarrow \langle y, z\rangle)$*, such that* $y=(Y, u, q_y)$*,* $z=(Z, v, q_z)$*,* $x=(X, u, q_y \cup q_z)$*, and the following conditions hold:*

1. $X \rightarrow Y\ @Z \in P$
2. $(u, v) \in E$
3. $q_y \cap q_z = \emptyset$

Figure 3 exemplifies the construction of two hyperedges (black arrows) incoming to a specific hypernode (x), for a colored vertex-labeled graph G. For example, the left hyperedge $(x \leftarrow \langle y_1, z_1\rangle)$ satisfies all three conditions: (1) $(v_1, v_5) \in E$, (2) $X \rightarrow Y'@Z' \in P$, and (3) the tail hypernodes contain disjoint sets of colors. The hyperedge $(x \leftarrow \langle y_1, z_2\rangle)$ is not constructed, since both y_1 and z_2 contain the color red, in contradiction to condition 3.

We call the set of hyperedges, with x as their *head*, the *backward star* of x, and denote it by $BS(x)=\{e \in E_H | x = head(e)\}$. Let $L_H \subseteq V_H$ denote the set of leaf hypernodes of H. Each hyperedge $e \in E_H$ and each leaf hypernode $\ell \in L_H$ is associated with weights $c(e)$, $c(\ell)$. The function $c : E_H \cup L_H \rightarrow \Re$ could be grammar-based, reflecting a derivation-rule specific scoring, or network-based, reflecting the confidence level of the corresponding edges or vertices in the network, or alternatively some combination of the two strategies.

For $e \in E_H$, let $s_c(e)$ denote some scoring scheme according to c, used for computing its score based on the score of the optimal derivation of its tail hypernodes. For example, in our applications, s_c is an addition function, $s_c(e)=c(e) + D^*(t_1(e)) + D^*(t_2(e))$. The score of the optimal derivation of a hypernode $x \in V_H$ is denoted $D^*(x)$, where

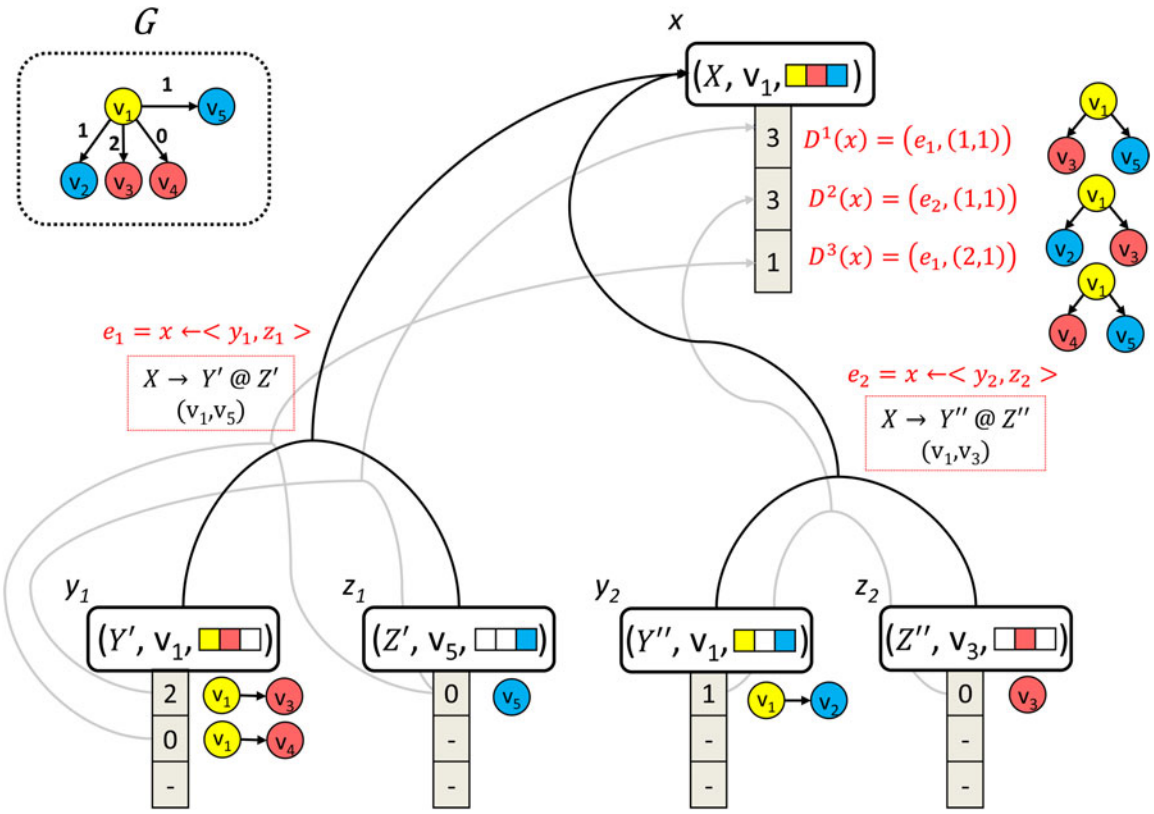

FIG. 3. Example of a step in the construction of the hypergraph H from the input graph G (see upper-left corner of the figure). The figure illustrates a partial hypergraph of H rooted in hypernode x. Hyperedges and derivations are marked black and gray, respectively. Note that this is only a partial view of the hypernodes and hyperedges participating in the hypergraph.

$$D^*(x) = \begin{cases} c(x) & v \text{ is a leaf} \\ \max_{e \in BS(x)} s_c(e) & \text{otherwise} \end{cases}$$

Based on the above definitions, Stage 1 of our algorithm applies an agglomerative construction of the hypergraph, while simultaneously computing $D^*(x)$ for all $x \in V_H$.

From Definition 4, every hyperedge corresponds to some edge in E, a production rule in P and a bipartition of some subset of C. Since the number of bipartitions of all subsets of $C = \{1, 2, \ldots, m\}$ is 3^m (obtained from $\sum_{i=0}^{m} \binom{m}{i} 2^i = 3^m$ via Newton's binomial), we get that $|E_H| \leq g|E|3^m$. In order to construct E_H efficiently we use binary search trees $T_{u,X}$ for every $u \in V$, $X \in N$, which store every hypernode of the form (X, u, q), for any $q \subseteq C$, in the following manner: For every node in level i of $T_{u,X}$, the left edge corresponds to nodes (X, u, q') with $i+1 \in q'$ and the right edge corresponds to nodes (X, u, q'') in which $i+1 \notin q''$. Since the height of every tree is m, we are able to construct every hyperedge in $O(m)$ time, and thus to achieve the following theorem (some of these principles were also discussed in Betzler et al., 2008; Bruckner et al., 2010; Scott et al., 2006).

Theorem 2. *The time complexity of Stage 1 of the proposed algorithm for solving Problem 2 is* $O(mg|E|3^m)$*, and the space complexity is* $O(|E_H|) = O(g|E|3^m)$.

4.2. Stage 2: computing k-best scoring trees

The hypernodes V_H of the hypergraph H, constructed in Stage 1, encode all trees that are accepted by the grammar A, together with the subtrees used to construct them. To extract the k-best scoring trees from H, we extend the k-best optimization algorithm, *FindKBest*, developed by Huang and Chiang (2005).

For exemplifications of our next definitions, we refer the reader again to Figure 3. Let $D(x)$ denote the list of the k-best derivations of a hypernode x (gray arrows) and $D^i(x)$ denote the *i'th* best derivation of x. Each derivation in $D(x)$ is a pair, $(e, (i, j))$ where $e = x \leftarrow \langle y, z \rangle$ and (i, j) is a pair of indices specifying two subderivations, $D^i(y)$ and $D^j(z)$, respectively. Each derivation corresponds to a specific tree in T_x. For example, the third derivation of hypernode x in the figure is marked $D^3(x) = (x \leftarrow \langle y_1, z_1 \rangle, (2, 1))$.

To compute $D(x)$ for all hypernodes, the algorithm *FindKBest* (see pseudocode Algorithm 1) is executed for each hypernode $x \in V_H$ in topological order, starting from the leaves and moving up the hypergraph. The algorithm maintains a priority queue *cand*[x] of potential candidates to be added to $D(x)$, sorted in decreasing score order. Initially, *cand*[x] is populated with the k top-scoring derivations among all hyperedges in $BS(x)$. This is achieved by calling the procedure *GetCandidates* (line 2 of the pseudocode). If $|D(x)| < k$, then *cand*[x] is further populated iteratively. In each such iteration, the top derivation $(e, (i, j))$ in *cand*[x] is removed from the priority queue and added to $D(x)$. Then, its neighboring derivations, $(e, (i, j+1))$ and $(e, (i+1, j))$ are inserted into *cand*[x]. This procedure continues until $|D(x)| = k$ or until all candidate derivations are exhausted.

The correctness of the algorithm *FindKBest* is based on the monotonicity of s_c, according to the following definition: s_c is monotonic with regards to $\preceq$, if for any two derivations $d_1 = (e_1, (i_1, j_1))$, $d_2 = (e_2, (i_2, j_2))$, $d_2 \preceq d_1 \Rightarrow s_c(d_2) \leq s_c(d_1)$, where $d_2 \preceq d_1 \Leftrightarrow e_1 = e_2$ and $i_1 \leq i_2, j_1 \leq j_2$.

Based on the above definition, for any derivation $(e, (i, j))$, the next-best scoring candidate derivation following $(e, (i, j))$ is either $(e, (i, j+1))$ or $(e, (i+1, j))$. This is the essential observation behind the *cube pruning* and *cube growing* approaches (Gesmundo and Henderson, 2010; Huang and Chiang, 2005) that is crucial to correctly, yet efficiently, enumerate the k-best derivations.

One obstacle encountered in our application is that the trees in G are unordered while the trees defined by RTGs are ordered. This means that any given subtree τ from G could have a one-to-many mapping to several top-scoring derivations in H. These derivations correspond to ordered trees that are isomorphic under rooted unordered isomorphism. (In rooted unordered isomorphism, trees τ_1 and τ_2 are isomorphic if τ_1 can be obtained from τ_2 by subtree reordering operations). As some or all of these isomorphic derivations could be reported among the k-best derivations, this undermines the correctness of the algorithm as a solution to Problem 2.

To prevent this from happening, we assign to each vertex $v \in G$ a unique integer id, and define a unique canonical encoding to each candidate derivation, representing its corresponding subtree as a parentheses-annotated sequence over lexicographic orderings of its vertex ids. This property ensures that any two isomorphic trees have the same canonical encoding. The canonical encoding logic is integrated into the algorithm *FindKBest* (line 7 of the pseudocode), ensuring that, for each hypernode x and for any two isomorphic trees τ_1 and τ_2 that are induced by two distinct derivations and are competing for $D(x)$, only a single, highest-scoring representative is kept. It is implemented as follows. For each hypernode x, we maintain an additional binary-search tree $B(x)$ containing pointers to all elements in $D(x)$, sorted by lexicographic order of their canonical codes. When a new candidate derivation is about to be added to $D(x)$, we first search for its canonical encoding in $B(x)$. If it is found, the new derivation is not added to $D(x)$.

To obtain the k-best trees that are the final solution to Problem 2, we add to the hypergraph H a pseudo-root hypernode, denoted v^p, such that all hypernodes in H that are marked with the initial symbol S are in the backward star of v^p. After the computation of Stage 2, the k-best list of v^p will contain the k-best trees solving Problem 2.

The work per iteration of Algorithm *FindKBest* is computed as follows. Constructing the canonical representation of a subtree can be naively implemented in $O(m)$ time, while operations on $B(x)$ take $O(mlogk)$ time. Moreover, since the candidate subtree is not necessarily added to $D(x)$ [due to the fact that an isomorphic tree could already be in $D(x)$], the number of iterations of the *while* loop in the algorithm *FindKBest* (line 5 of the pseudocode) is no longer limited to k, as was the case in the original algorithm. In what follows, we bound the number of these redundant *while* loop iterations. This will be formalized via Lemma 1, which leans on the following proposition:

Proposition 1. *Any two derivations induced by the same hyperedge correspond to nonisomorphic trees.*

The correctness of Proposition 1 (formulated in the proof below) is derived from an inductive assumption, relying on the fact that algorithm *findKBest* is executed on the hypernodes in topological order. Hence, we may assume that when algorithm *findKBest* is executed on some hypernode x, the k-best lists of

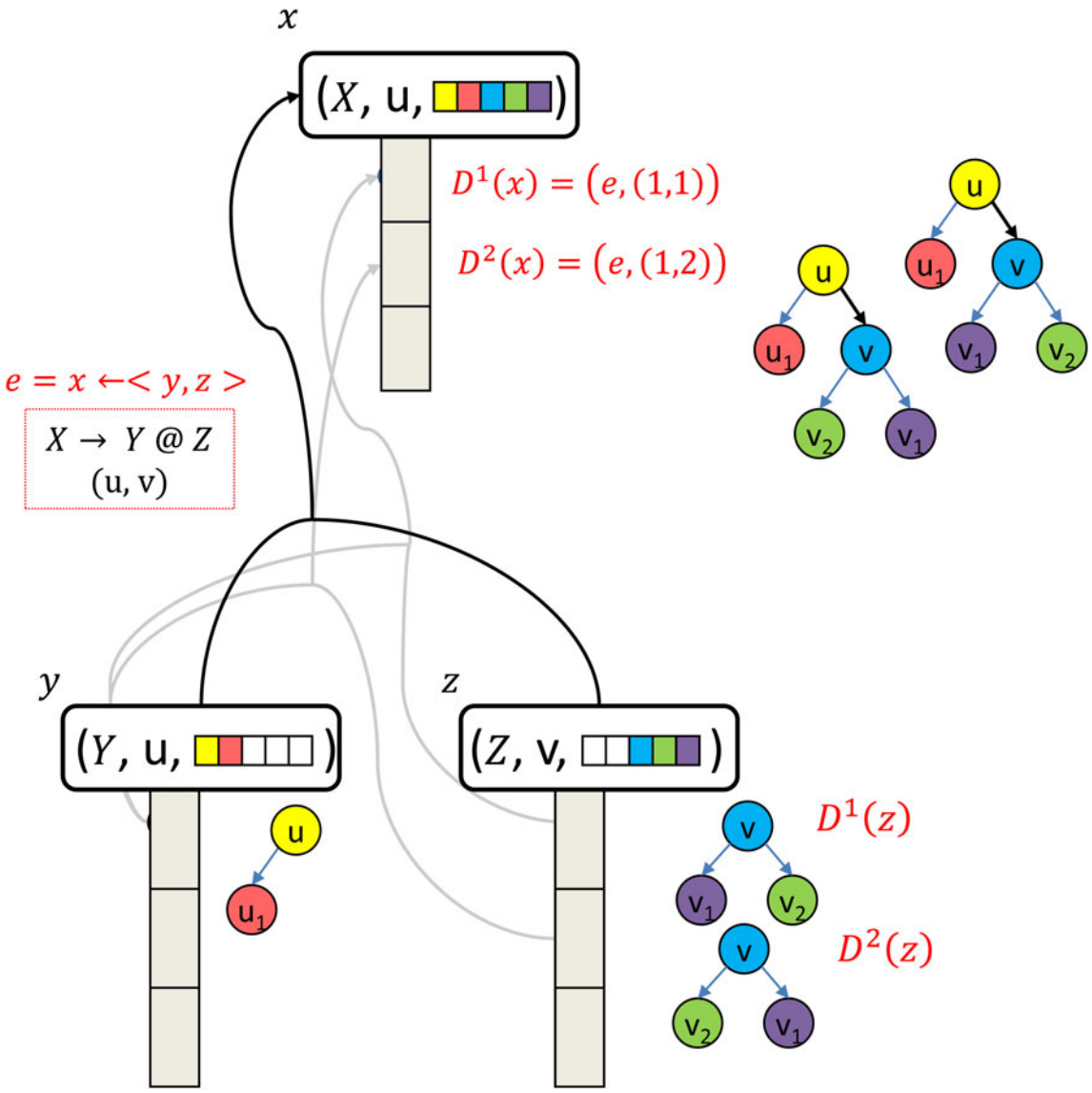

FIG. 4. This figure shows, by contradiction, that two derivations from the same hyperedge can not produce two isomorphic trees, given a hyperedge $e = x \leftarrow \langle y, z \rangle$ and two derivations, $(e, (1, 1))$ and $(e, (1, 2))$, that produce isomorphic trees. Since both trees consist of the same vertices, they must be derived from isomorphic smaller subtrees in the k-best list of at least one of the tail hypernodes: y or z. For example, $D^1(z)$ and $D^2(z)$ are isomorphic, thus contradicting the inductive assumption that smaller hypernodes do not contain isomorphic trees.

all hypernodes that are connected to x have already been computed and do not contain more than one representative from each class of isomorphic trees. Figure 4 exemplifies the rationale of the proof below.

Proof. The proof is by induction on $|q_x|$ — the number of colors in hypernode x. For the base of the induction note that if $|q_x| = 1$ then $x \in L_H$, and thus there is exactly one derivation in $D(x)$. For the induction step let $e = x \leftarrow \langle y, z \rangle$ that corresponds to some edge $(u, v) \in E$, and two derivations $d_1 = (e, (i_1, j_1))$, $d_2 = (e, (i_2, j_2))$ that correspond to some tree T (up to isomorphism). From the induction hypothesis, since $|q_y| < |q_x|$ and $|q_z| < |q_x|$, there are no two derivations in $D(y)$ that represent isomorphic trees, and the same holds for $D(z)$. Denote by T_v the subtree of T rooted in v. Since both $D^{j_1}(z)$ and $D^{j_2}(z)$ represent trees that are isomorphic to T_v, thus $j_1 = j_2$. Consequently, both $D^{i_1}(y)$ and $D^{i_2}(y)$ represent trees that are isomorphic to $T \setminus T_v$ (i.e., the tree T without v and its descendants), and therefore, $i_1 = i_2$. ■

Following Proposition 1, we bound the number of distinct hyperedges in $BS(x)$ that can produce isomorphic trees. This is formulated in the following Lemma:

Lemma 1. *For hypernode $x=(X, u, q_x)$, the number of distinct incoming hyperedges that may produce the same tree (up to isomorphism) is bounded by $m \cdot \alpha_X$, where $\alpha_X=|\{X \to Y\ @Z \mid Y, Z \in N\}|$ denotes the number of different production-rules derived by X.*

The correctness of Lemma 1 is derived from the basic principle, by which each derivation adds a rightmost child-subtree to the root of the derived tree, and there are at most $m-1$ possibilities for this, per each production-rule that derives these trees. Figure 5 exemplifies the rationale of the proof below.

Proof. Assume in contradiction that $\alpha_X \cdot m+1$ distinct hyperedges in $BS(x)$ produce the same tree T (up to isomorphism). From the pigeon-hole principle over Definition 4 there are two distinct hyperedges that correspond to the same production-rule, say $X \to Y\ @\ Z$, and the same edge, say (u, v). Denote these hyperedges as $e_1=(x \leftarrow \langle y_1, z_1\rangle)$ and $e_2=(x \leftarrow \langle y_2, z_2\rangle)$, and denote $y_1=(Y, u, q_{y_1})$, $z_1=(Z, v, q_{z_1})$ and $y_2=(Y, u, q_{y_2})$, $z_2=(Z, v, q_{z_2})$. Since $q_x=q_{y_1} \cup q_{z_1}=q_{y_2} \cup q_{z_2}$ and $q_{y_1} \cap q_{z_1}=q_{y_2} \cap q_{z_2}=\emptyset$, we get that $q_{y_1} \neq q_{y_2}$, $q_{z_1} \neq q_{z_2}$ (otherwise $q_{y_1}=q_{y_2} \leftrightarrow q_{z_1}=q_{z_2}$, which means $y_1=y_2$, $z_1=z_2$). Assume without loss of generality that $q_{z_1} \setminus q_{z_2} \neq \emptyset$, and denote by $d_1=(e_1, (i_1, j_1))$ and $d_2=(e_2, (i_2, j_2))$ the two derivations that produce isomorphic trees. Therefore, there exists a vertex v_1, which is a descendant of v in the subtree represented by $D^{j1}(z_1)$ and not a descendant of v in any subtree represented by derivations in $D(z_2)$, thus we reach a contradiction to the claim that these hyperedges may produce isomorphic trees. ■

Lemma 1 leads to the following theorem, which bounds the time and space complexities of Stage 2 of the algorithm.

Theorem 3. *The time complexity of Stage 2 of the proposed solution to Problem 2 is $O(g|E|3^m + g|V|2^m km(m \log k + \log \alpha))$, where $\alpha = \max_{X \in N} \alpha_X$. The space complexity is $O(m \cdot k \cdot |V_H|) = O(m \cdot k \cdot |N||V|2^m)$.*

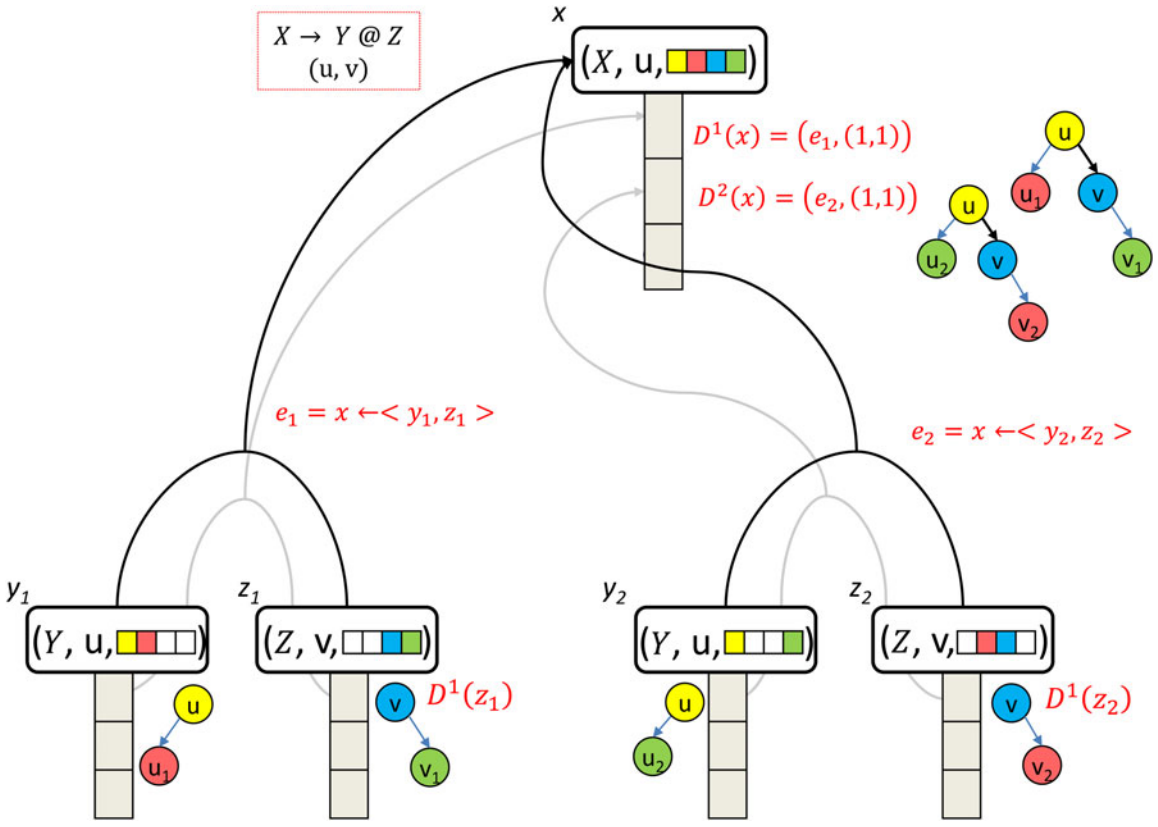

FIG. 5. Distinct hyperedges that are induced by the same production-rule and the same edge can not produce two isomorphic trees. Two distinct hyperedges incoming to the same hypernode $x=(X, u, q_x)$, induced by the same production-rule $X \to Y\ @Z$ and by the same edge (u, v). Since these are distinct hyperedges, they must be induced by distinct bipartition of q_x. Therefore, the trees obtained by these hyperedges can not be isomorphic.

Proof. Following Lemma 1, we analyze the running-time of the algorithm *findKBest* for a given hypernode $x=(X, u, q_x)$. For each tree τ in the k-best list of x, there are at most $m \cdot \alpha_X$ iterations in which we compute τ. Therefore, we have at most $k \cdot m \cdot \alpha_X$ iterations. Operations on $cand[x]$ cost $O(\log(k \cdot m \cdot \alpha_X))$. For each candidate derivation, we construct its canonical representation in $O(m)$ time. We use a binary search tree, $B(x)$, by a lexicographically ordering comparator, to store the canonical codes of all subtrees in the k-best list. Hence operations cost $O(m \cdot \log k)$ time [i.e., $O(\log k)$ for traversing the binary tree of size k, multiplied by an $O(m)$ factor for comparator operations]. Therefore, the running time per hypernode x is computed as follows: $O(|BS(x)|+k \cdot m\alpha_X(\log (k \cdot m \cdot \alpha_X)+m \log k)) \leq O(|BS(x)|+k \cdot m \cdot \alpha_X(m \cdot \log k+\log \alpha))$.

We now compute the overall time-complexity of Stage 2, as the sum of *findKBest* running-times across all hypernodes. For this, note that: (1) $|E_H|=O(g \cdot |E| \cdot 3^m)$ (see Theorem 2); and (2) $\sum_{X \in N} \alpha_X = g$ (every production rule is counted exactly once). Thus, the overall time-complexity is computed as follows:

$O(\sum_{x \in V_H} |BS(x)| + \sum_{x \in V_H} km\alpha_X(m \log k + \log \alpha)) = O(|E_H| + \sum_{v \in V} \sum_{q \subseteq C} \sum_{X \in N} km\alpha_X(m \log k + \log \alpha)) = O(g|E|3^m + g|V|2^m \cdot km(m \log k + \log \alpha))$.

The space complexity is $O(mk \cdot V_H)$. Since each node in the hypergraph stores the canonical codes of the k-best derivations. ■

Algorithm 2, *FindKBest*, of Huang and Chiang (2005) runs in $O(|E_H| + |V_H|k \log k)$ time and computes the k-best derivations in a bottom-up manner for each hypernode in the hypergraph. In the same article, the authors also describe a lazy top-down algorithm, which runs in $O(|E_H| + D_{output}k \log k)$ time, where D_{output} is the size of the output. This is computed by calling the recursive procedure *FindKBestLazy*, starting at the root hypernode and calling itself recursively only as necessary, visiting only hypernodes that will participate in the output. However, in our case, some of the visited hypernodes may belong to redundant isomorphic subtrees. Therefore, the parameter D_{output} does not hold and is replaced by a factor that can be at most $|V_H|$. However, since the lazy approach still offers some practical improvement with graceful degradation toward the complexity of the nonlazy approach, the code we make available as part of this submission implements the lazy variant.

Algorithm 1: *FindKBest*(x, k)

```
Input: A hypernode x and an integer k.
Output: Vector D(x) of k-best scoring derivations rooted in x.
cand[x] ← ∅
GetCandidates(x, αmk)
D(x) ← ∅
B(x) ← ∅
while |cand[x]| > 0 and |D(x)| < k do
    (e, (i, j)) = cand[x].dequeue()
    if (e, (i, j)) ∉ B(x) then
        D(x).insert((e, (i, j)))
        B(x).insert((e, (i, j)))
    cand[x].insert(e, (i+1, j))
    cand[x].insert(e, (i, j+1))
```

5. METHODS

Physical interactions between human proteins and their reliability scores were retrieved via MyProteinNet (Basha et al., 2015) from BIOGRID, MINT, INTACT, and DIP, and scored using a uniform weight scheme. Data of upregulated genes following influenza infection were gathered from Shapira et al. (2009) by using thresholds therein. We considered genes that were upregulated upon infection by wild-type PR8 influenza virus, or by PR8 virus lacking the NS1 gene, or by viral RNA. Each gene was associated with the earliest time point at which it was upregulated. Data of upregulated genes spanned five timepoints that correspond to 4, 6, 8, 12, and 18 hours post infection. The score of each cascade was set to the sum of the absolute log fold-change of genes and interaction scores. Virus–host, protein–protein interactions were downloaded from publicly available interaction databases using PSIQUIC.

6. RESULTS

We demonstrate the power of RTG queries in two applications, both of which mine the human protein interaction network for viral infection patterns. The first application exemplifies the expressive power of grammar-based queries. The second application exemplifies the ability to score patterns as part of the grammar.

Our first query aimed to identify temporal responses of human cells to infection by the flu virus influenza type A. This virus is a single-stranded, negative-sense segmented RNA virus that infects hundreds of millions of people and results in numerous hospitalizations and deaths worldwide (Belshe, 2009). The transcriptional response to infection by the virus or its components was measured across time in human primary bronchial epithelial cells (Shapira et al., 2009). To gain a mechanistic insight into the cellular response to infection, we mapped these data onto a network representing physical interactions between human proteins. We defined a query that searches this network for temporal pathway cascades: Starting with a protein that was up regulated at the earliest time-point following infection, the cascade proceeds iteratively with one or more proteins that were up-regulated at the same or subsequent time point, while requiring that the two proteins interact with each other (Fig. 6A). To prioritize cascades that are biologically relevant, we scored instances based on the sum of transcript levels of the proteins in the cascade and the reliability of their interactions. Using RTGnet we identified the top 1,000 upregulated cascades ($\epsilon = 0.05$) involving at most nine genes, which required 80,248 color-coding iterations.

We demonstrate the mechanistic insight that can be gained from this query by discussing the top-scoring upregulated cascade (Fig. 6A). This top-most cascade delineated important defence steps in the host response to influenza infection, starting from detection of viral particles, inhibition of their production, and ending with cell death. The cascade starts with upregulation of ISG15, an immune-related ubiquitin-like protein that recruits its interacting partners to fight viral infection (Zhao et al., 2005), and two interferon-induced proteins, IFIT1 and IFIT2, which target viral RNA and the synthesis of viral proteins (Abbas et al., 2013). It follows with upregulation of MX1, an interferon-induced GTPase with a known antiviral activity, and DDX85 (also known as RIG-I), a receptor activating a cascade of antiviral responses, whose conjugation with ISG15 is essential for the initiation of this pathway (Arimoto et al., 2008). Upregulated at the consecutive time point was TRIM25, forming the DDX58-TRIM25 complex that mediates the DDX58-ISG15 conjugation (Gack et al., 2007). Upregulated at the fourth time point was RBCK1, which regulates

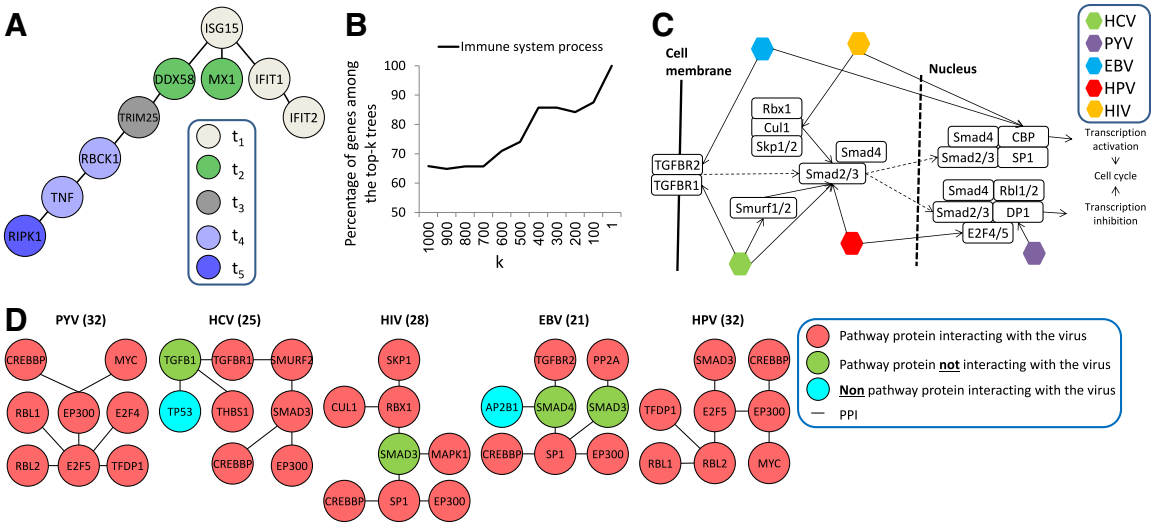

FIG. 6. (**A**) The maximal-scoring instance of the temporal cascade following influenza infection. Protein colors denote the first time point at which they were upregulated. (**B**) The percentage of proteins annotated to immune system process in the k top-scoring subtrees increases as k is decreased from 1,000 to 1. (**C**) A schematic view of the TGF-β signaling pathway showing that distinct viruses interact with different proteins in the pathway. (**D**) Maximal-scoring subtree per virus (and score) in the TGF-β pathway, demonstrating the power of an RTG to define various patterns in a single grammar.

the activity of the DDX58-TRIM25 complex (Inn et al., 2011), and TNF, which together with RBCK1 initiates TNF-mediated gene induction (Haas et al., 2009). This is followed by upregulation of RIPK1, that initiates inflammatory and cell-death cascades upon TNF-receptor signaling (Ea et al., 2006). This cascade is especially interesting since several of its proteins interact with the virus: ISG15 and MX1 interact with influenza's NS1 and NP proteins, correspondingly, to inhibit viral replication (Guan et al., 2011; Verhelst et al., 2012), and TRIM25 was found be to be targeted by influenza's NS1 protein to evade recognition of viral RNA by RIG-I (Gack et al., 2009). Lastly, the silencing of ISG15, IFIT2, TRIM25, or RBCK1 was found to promote influenza lifecycle and replication (Shapira et al., 2009). To assess our prioritization scheme we computed the fraction of proteins annotated to the immune system (GO:0002376) (Ashburner et al., 2006) across different values of k (Fig. 6B). This fraction increased as we limited k and was enriched in the top-ranking subtree relative to other subtrees ($p = 0.0125$, Fisher's exact test).

Our second query aimed to identify human pathways that a specific virus targets in multiple, related points, as these pathways are likely to be of high importance to its survival and replication. Therefore, we designed a query that accepted subtrees in which several proteins in the pathway interact with each other and with the virus, either directly or via a mediator protein. The accepted subtrees were scored as part of the grammar according to the viral interactions of pathway proteins (see Supplementary Material for details): Proteins that directly interact with viral proteins scored high (+4), while proteins with indirect interactions scored according to the number of mediator proteins (+1 for each mediator protein). Note that the topology of the subtrees was not predefined. To implement this query we augmented the human protein interaction network described above with data of protein interactions between human and viral proteins for six viruses: Epstein-Barr virus (EBV), hepatitis C virus (HCV), human papillomavirus (HPV), polyoma virus (PYV), human immunodeficiency virus (HIV1), and influenza A virus subtype H1N1 (H1N1). These viruses were selected because they are evolutionarily remote and their interactions with human proteins were relatively mapped. Associations of human proteins to cellular pathways were extracted from KEGG (Kanehisa et al., 2014). As proof-of-concept we focus on the transforming growth factor-β (TGF-β) signaling pathway that regulates cell growth and differentiation, which is crucial for replication of viruses that establish persistent (chronic or latent) infections. Indeed, the only virus without a high-scoring instance was H1N1, a virus that does not establish such persistent infections (Figure 6C). Top-scoring instances were identified for all other viruses and had distinct topologies, demonstrating the ability of RTGs to define flexible patterns with a single query (Figure 6D).

The instances we found could have been identified by other types of network querying approaches, though they would require hard-tailoring of the application to ensure that all specifications are followed. RTGnet on the other hand offers a general framework, in which various such queries can be expressed and scored.

7. DISCUSSION

We introduce a novel framework, RTGnet, that extends the class of grammar-based queries that can be efficiently searched within a network to include all languages defined by RTGs, and produces the list of k-best results. We demonstrate the capabilities of RTGnet in two applications that highlight the generality of the tool and the topological flexibility of the identified instances. The RTG search modeling that we suggest is particularly useful in networks that are rich in vertex-labels, enabling the design of more expressive grammars. Another advantage of our framework is the ability of k-best optimization to identify a space of near-optimal solutions, making them suitable for learning stochastic production scores, given a reliable source of training data. Future extensions could include the derivation and usage of more informed scoring schemes, more complex grammars to express the queries, and the extension of tree-based instances to general subnetworks.

ACKNOWLEDGMENTS

This research was supported by the Israel Science Foundation 478/10 and 179/14 to M.Z.-U. and 860/13 to E.Y.-L.

AUTHOR DISCLOSURE STATEMENT

The authors declare that no competing financial interests exist.

REFERENCES

Abbas, Y.M., Pichlmair, A., Górna, M.W., et al. 2013. Structural basis for viral 5 ′-ppp-rna recognition by human ifit proteins. *Nature*. 494, 60–64.

Alon, N., Yuster, R., and Zwick, U. 1995. Color-coding. *JACM* 42, 844–856.

Arimoto, K.-I., Konishi, H., and Shimotohno, K. 2008. Ubch8 regulates ubiquitin and isg15 conjugation to rig-i. *Mol. Immunol.* 45, 1078–1084.

Ashburner, M., Ball, C., Blake, J., et al. 2006. Gene ontology: Tool for the unification of biology. The Gene Ontology Consortium database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.* 34.

Barabási, A.-L., Gulbahce, N., and Loscalzo, J. 2011. Network medicine: A networkbased approach to human disease. *Nat. Rev. Genet.* 12, 56–68.

Basha, O., Flom, D., Barshir, R., et al. 2015. Myproteinnet: Build up-to-date protein interaction networks for organisms, tissues and user-defined contexts. *Nucleic Acids Res.* 43, W258–W263.

Belshe, R.B. 2009. Implications of the emergence of a novel h1 inuenza virus. *N. Engl. J. Med.* 360, 2667–2668.

Betzler, N., Fellows, M.R., Komusiewicz, C., and Niedermeier, R. 2008. Parameterized algorithms and hardness results for some graph motif problems, 31–43. In *Combinatorial Pattern Matching*. Springer, New York.

Bruckner, S., Hüffner, F., Karp, R.M., et al. 2010. Topologyfree querying of protein interaction networks. *J. Comput. Biol.* 17, 237–252.

Carme, J., Niehren, J., and Tommasi, M. 2004. Querying unranked trees with stepwise tree automata, 105–118. In *Rewriting Techniques and Applications*. Springer, New York.

Comon, H., Dauchet, M., Gilleron, R., et al. 2007. Tree automata techniques and applications. Release October 12, 2007. www.grappa.univ-lille3.fr/tata

Dost, B., Shlomi, T., Gupta, N., et al. 2008. Qnet: A tool for querying protein interaction networks. *J. Comput. Biol.* 15, 913–925.

Ea, C.-K., Deng, L., Xia, Z.-P., et al. 2006. Activation of ikk by tnfα requires site-specific ubiquitination of rip1 and polyubiquitin binding by nemo. *Mol. Cell* 22, 245–257.

Fan, W., Li, J., Ma, S., et al. 2011. Adding regular expressions to graph reachability and pattern queries, 39–50. In *2011 IEEE 27th International Conference on Data Engineering (ICDE)*. IEEE, New York.

Finkelstein, A., and Roytberg, M. 1993. Computation of biopolymers: A general approach to different problems. *BioSystems* 30, 1–19.

Gack, M.U., Albrecht, R.A., Urano, T., et al. 2009. Inuenza a virus ns1 targets the ubiquitin ligase trim25 to evade recognition by the host viral rna sensor rig-i. *Cell Host Microbe* 5, 439–449.

Gack, M.U., Shin, Y.C., Joo, C.-H., et al. 2007. Trim25 ring-finger e3 ubiquitin ligase is essential for rig-i-mediated antiviral activity. *Nature* 446, 916–920.

Gesmundo, A., and Henderson, J. 2010. Faster cube pruning, 267–274. IWSLT, Citeseer.

Guan, R., Ma, L.-C., Leonard, P.G., et al. 2011. Structural basis for the sequence-specific recognition of human isg15 by the ns1 protein of inuenza b virus. *Proc. Natl. Acad. Sci. USA* 108, 13468–13473.

Haas, T.L., Emmerich, C.H., Gerlach, B., et al. 2009. Recruitment of the linear ubiquitin chain assembly complex stabilizes the tnf-r1 signaling complex and is required for tnf-mediated gene induction. *Mol. Cell* 36, 831–844.

Huang, L., and Chiang, D. 2005. Better k-best parsing, 53–64. In *Proceedings of the Ninth International Workshop on Parsing Technology*. Association for Computational Linguistics, Stroudsburg, PA.

Inn, K.-S., Gack, M.U., Tokunaga, F., et al. 2011. Linear ubiquitin assembly complex negatively regulates rig-i-and trim25-mediated type I interferon induction. *Mol. Cell* 41, 354–365.

Kanehisa, M., Goto, S., Sato, Y., et al. 2014. Data, information, knowledge and principle: Back to metabolism in kegg. *Nucleic Acids Res.* 42, D199–D205.

Koschmieder, A., and Leser, U. 2012. Regular path queries on large graphs, 177–194. In *Scientific and Statistical Database Management*. Springer, New York.

Lacroix, V., Fernandes, C.G., and Sagot, M.-F. 2006. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 3, 360–368.

Lange, M., and Leiß, H. 2009. To CNF or not to CNF? An efficient yet presentable version of the cyk algorithm. *Inform. Didact.* 8, 2008–2010.

Martens, W., and Niehren, J. 2005. Minimizing tree automata for unranked trees, 232–246. In *Database Programming Languages*. Springer, New York.

Mendelzon, A.O., and Wood, P.T. 1995. Finding regular simple paths in graph databases. *SIAM J. Comput*. 24, 1235–1258.

Patro, R., and Kingsford, C. 2013. Predicting protein interactions via parsimonious network history inference. *Bioinformatics* 29, i237–i246.

Pinter, R.Y., Rokhlenko, O., Yeger-Lotem, E., and Ziv-Ukelson, M. 2005. Alignment of metabolic pathways. *Bioinformatics* 21, 3401–3408.

Ponty, Y., and Saule, C. 2011. A combinatorial framework for designing (pseudoknotted) RNA algorithms, 250–269. In *Algorithms in Bioinformatics*. Springer, New York.

Scott, J., Ideker, T., Karp, R.M., and Sharan, R. 2006. Efficient algorithms for detecting signaling pathways in protein interaction networks. *J. Comput. Biol.* 13, 133–144.

Sevon, P., and Eronen, L. 2008. Subgraph queries by context-free grammars. *J. Integr. Bioinform.* 5, 100.

Shapira, S.D., Gat-Viks, I., Shum, B.O., et al. 2009. A physical and regulatory map of host-inuenza interactions reveals pathways in h1n1 infection. *Cell* 139, 1255–1267.

Shlomi, T., Segal, D., Ruppin, E., and Sharan, R. 2006. Qpath: A method for querying pathways in a protein-protein interaction network. *BMC Bioinform*. 7, 199.

Verhelst, J., Parthoens, E., Schepens, B., et al. 2012. Interferoninducible protein mx1 inhibits inuenza virus by interfering with functional viral ribonucleoprotein complex assembly. *J. Virol.* 86, 13445–13455.

Zhao, C., Denison, C., Huibregtse, J.M., et al. 2005. Human isg15 conjugation targets both ifn-induced and constitutively expressed proteins function-27 ing in diverse cellular pathways. *Proc. Natl. Acad. Sci. USA* 102, 10200–10205.

Address correspondence to:
Dr. Michal Ziv-Ukelson
Department of Computer Science
Ben-Gurion University of the Negev
P.O.B. 653
Beer Sheva 84105
Israel

E-mail: michaluz@cs.bgu.ac.il