Exact yet Efficient Graph Parsing, Bi-directional Locality and the Constructivist Hypothesis

Yajie Ye[1] and **Weiwei Sun**[12]
[1]Wangxuan Institute of Computer Technology
[1]The MOE Key Laboratory of Computational Linguistics
[2]Center for Chinese Linguistics
Peking University
{yeyajie,ws}@pku.edu.cn

Abstract

A key problem in processing graph-based meaning representations is *graph parsing*, i.e. computing all possible derivations of a given graph according to a (competence) grammar. We demonstrate, for the first time, that exact graph parsing can be efficient for large graphs and with large Hyperedge Replacement Grammars (HRGs). The advance is achieved by exploiting locality as terminal edge-adjacency in HRG rules. In particular, we highlight the importance of 1) a terminal edge-first parsing strategy, 2) a categorization of a subclass of HRG, i.e. what we call Weakly Regular Graph Grammar, and 3) distributing argument-structures to both lexical and phrasal rules.

1 Introduction

Language production, though as important as language understanding, has received very limited theoretical and empirical research attention. A fundamental problem in modeling language production is *parsing meaning representations*, i.e. computing all possible analyses of a given meaning representation (MR) according to a (competence) grammar. In theory, the worst-case complexities of existing algorithms are exponential or high-degree polynomial w.r.t. grammar size and input length. In practice, there are few systems that can parse large but frequent MRs with a realistic, wide-coverage grammar in a reasonable time.

The major contribution of this paper is an exact yet efficient method to parse MRs in the framework of graph-based semantic representations (Koller et al., 2019) and Hyperedge Replacement Grammar (Drewes et al., 1997). The ability to enumerate all possible analyses of a graph facilitates surface realization, grammar induction, recursive graph embedding, etc. The advance in efficiency is from exploiting locality of HRG rules from the rarely discussed perspective of language production, a reversed direction to language understanding. We discuss locality in a sense of terminal edge-adjacency and develop a locality-centric complexity analysis of the *de facto* algorithm introduced by Chiang et al. (2013). Our analysis motivates (1) a terminal edge-first parsing strategy, (2) a categorization of a subclass of HRG, i.e. what we call Weakly Regular Graph Grammar, and (3) a computational support in the constructivist hypothesis in theoretical linguistics. Altogether, our analysis leads to a substantial improvement in practical graph parsing. An MR with the number of conceptual nodes ranging from 5 to 50 corresponding to a Wall Street Journal sentence can receive a full-forest analysis in 0.089 second on average with a large-scale comprehensive grammar; Even semantic graphs with c.a. 80 conceptual nodes can be processed in less than 0.5 second.

2 A Graph-Structured Syntax-Semantics Interface

Linguistically-informed graph parsing needs a precise model of the syntax-semantics interface. To this end, we need to precisely describe elementary structures corresponding to linguistic units at (morphological,) lexical and phrasal levels, and precisely describe the MERGE operation of two linguistic units. Under the umbrella of graph-based MRs, we employ hypergraphs and HRGs (Drewes et al., 1997) to achieve the two goals.

Throughout this paper, we define an edge-labeled, ordered **hypergraph** over finite alphabet Σ as a tuple $G = (V, E, \ell)$, where V is a finite set of nodes, $E \subseteq V^+$ is a finite set of hyperedges, and $\ell : E \mapsto \Sigma$ is a labeling function. A hyperedge can connect to more than two nodes or a single node. Labels can be associated to edges but not nodes. The set of nodes connected by edge e are denoted by $V(e)$ and the set of edges connected to

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 4100–4110
July 5 - 10, 2020.

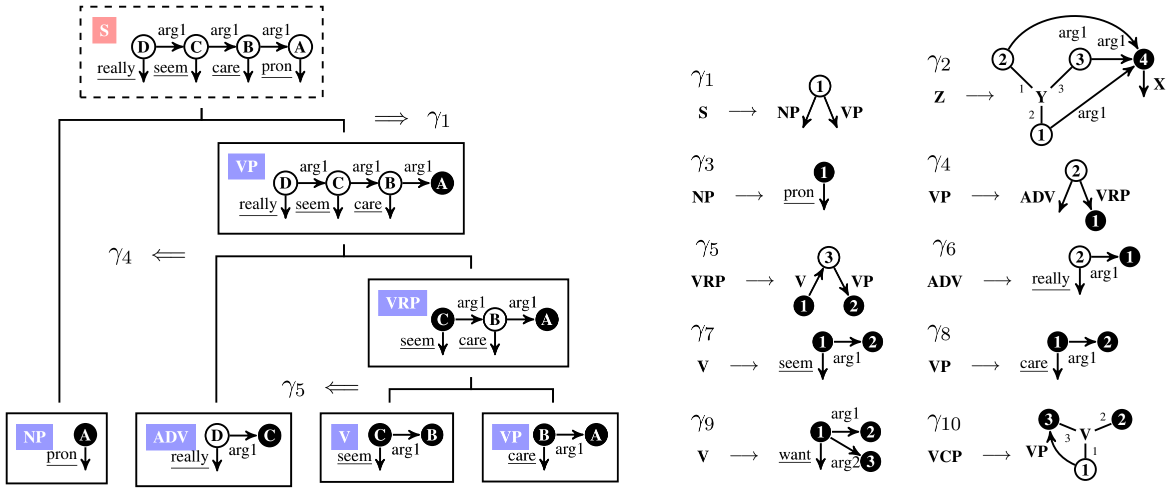

Figure 1: An HRG-based syntactico-semantic derivation for *He really seems to care*. The right part are examples of HRG rules. Throughout this paper, we use filled black nodes to indicate external nodes, arrows to indicate single-node edges and directed arcs to indicate edges connected to two nodes. The edge labeled as **Y** in rule γ_2 connects more than two nodes whose orders are indicated by tiny numbers around lines. Nodes in an HRG rule and subgraphs of an input graph are mentioned with numbers and characters respectively. Since nodes receive no informative labels, we use single-node edges with underlined terminal labels to represent concepts, e.g. "pron." Others terminal labels, e.g. "arg1," express semantic roles.

node v are denoted by $E(v)$. We use graph and hypergraph interchangeably, and similarly for edge and hyperedge.

Fig. 1 presents an example that contains a raising construction. The graph associated to the sentence (indicated by S) is derived along with a syntactic tree, in which the leaves and internal nodes are associated with graphs (indicated by x) as lexical and phrasal interpretations.

The key operation in semantic composition is to glue two graphs, say G_1 and G_2. It is obvious that not every node in G_1 is *visible* to G_2 and vice versa. To emphasize on this point, we augment the representation of a hypergraph (V, E, ℓ) with a list of ordered **external nodes** $V_x \in V^+$ and get **a hypergraph fragment** $H = (V, E, \ell, V_x)$. The number of external nodes is denoted by $rank(H)$.

Graph gluing can be manipulated by an HRG $\mathcal{G} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$, where $\mathcal{N}$ and $\mathcal{T}$ are two finite disjoint alphabets of nonterminal and terminal symbols respectively, $S \in \mathcal{N}$ is the start symbol, and $\mathcal{P}$ is the finite collection of rewriting rules in the form of $A \to R$. The left hand side (LHS) A belongs to $\mathcal{N}$, and the right hand side (RHS) R is a hypergraph fragment over $\mathcal{N} \cup \mathcal{T}$. See γ_1 to γ_{10} in Fig. 1 for example.

A carefully designed HRG can be linguistically elegant, in that its rules are consistent with state-of-the-art linguistic analysis. For instance, raising and control constructions receive principled analysis with rules in Fig.1. HRG can be comparable to other popular grammar formalisms, such as Combinatory Categorial Grammar (CCG; Steedman, 1996, 2000). See Fig. 2 for an illustration.

$(\mathsf{S}\backslash\mathsf{NP_y})/\mathsf{NP_x}$
$\lambda x.\lambda y.\underline{\text{like}}(y, x)$

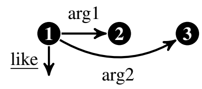

Figure 2: A comparison of CCG and HRG. The external nodes ❶, ❷ and ❸ corresponds to S, $\mathsf{NP_y}$ and $\mathsf{NP_x}$ in the syntactic category respectively.

3 Graph Parsing with a General HRG

In the framework of graph-based MRs, a key problem is *graph parsing*: computing all possible analyses of a given semantic graph according to a grammar. Fig. 3 demonstrates the target structure of graph parsing — derivation forest. A derivation forest allows us to efficiently enumerate every derivation. Coupled with a *local* score function that evaluates the *goodness* of a rule application, a graph parser can further tell the goodness of a particular derivation tree or the full forest as a whole.

Though essential, graph parsing is only partially understood. In this section, we summarize the state-of-the-art algorithm for graph parsing with HRGs (Chiang et al., 2013), and then evaluate its efficiency with a wide-coverage grammar.

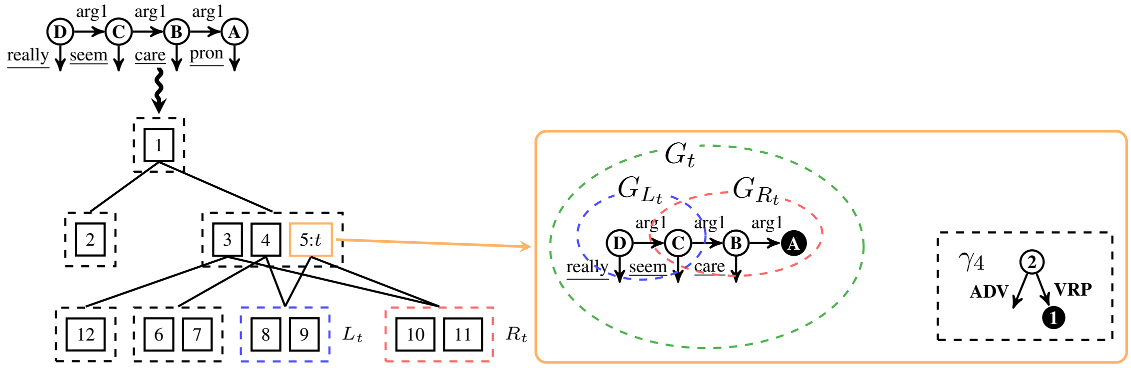

Figure 3: Graph parsing with an HRG. The context-freeness of HRG allows us to represent a derivation as a tree, and sets of derivations as a derivation forest, which is the output structure of graph parsing. In the derivation forest, a dashed rectangle (node) corresponds to a subgraph, which may be immediately built with different HRG rules. Each rule application is separately represented as a box. Necessary and sufficient information includes the BRs of G_t, G_{L_t} as well as G_{R_t} and the rule itself.

3.1 A Dynamic Programming Algorithm

Chiang et al.'s algorithm is a dynamic programming algorithm, in which a collection of in-process subgraphs are iteratively recognized as solutions to subproblems. Two key techniques are introduced concerning (1) how to *pack* a subgraph and (2) how to expand recognized subgraphs.

A subgraph is compactly encoded by **boundary representation** (BR) defined as follows. Assume I is a subgraph of a graph H. A **boundary node** of I is an external node of H or it is incident to an edge that is not in I. A **boundary edge** of I is an edge in I which connects to a boundary node. Let m be an arbitrarily chosen marker node in H. The BR of I is the tuple $b(I) = \langle bn(I), be(I), m \in I\rangle$, where $bn(I)$ is the set of I's boundary nodes, $be(I)$ is the set of I's boundary edges, and $(m \in I)$ is a boolean value indicating whether m is in I. Take P_1 in Fig. 5 for example. The dotted box shows a subgraph that has been recognized. $bn($Y$) =$ {A C F}, and D and G are irrelevant to further recognition.

Now consider combining two subgraphs recognized as nonterminal X and Y according to γ_2 in Fig. 5. As to incrementally match elements of a rule, e.g. γ_2, in an edge-by-edge way, Chiang et al. proposes to leverage a **tree decomposition**[1] T_R of the RHS of an HRG rule $A \rightarrow R$ $(R = \langle V, E, \ell, V_x\rangle)$. A tree decomposition T_R is **nice**, if every node of T_R must be one of: (1) a leaf node associated to empty graph; (2) a unary node which introduces exactly one edge; (3) a binary node which introduces no edges. Throughout, for convenience, let η denote a node from T_R and $R_{\trianglerighteq\eta}$ denote the subgraph of R whose edges are induced by nodes in the subtree rooted by η. If η is binary, its children are denoted by η_1 and η_2. If η is unary, the edge introduced by it and its only child are denoted by e and η_1 respectively.

Oriented by the fundamental architecture of chart parsing/generation (Kay, 1996), T_R are used to define active/passive items and inference rules that process such items. A **passive item** is of

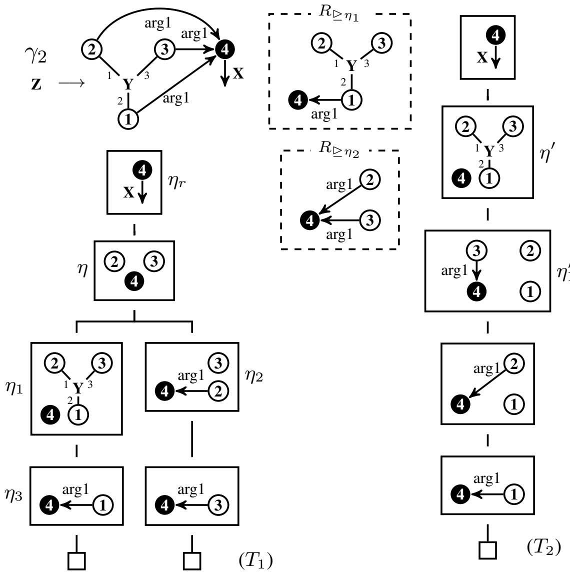

Figure 4: T_1 and T_2 are two nice tree decompositions of RHS of γ_2. Both are of width 3.

[1]A tree decomposition T of a graph fragment $H = \langle V, E, \ell, V_x\rangle$ is a tree that every node η in T is associated with a tuple $\langle V_\eta, E_\eta\rangle$. T must satisfy the following properties: (1) for each $v \in V$, there is a node η such that $v \in V_\eta$; (2) for each $e \in E$, there is exactly one node η such that $e \in E_\eta$ and $V(e) \subseteq V_\eta$; (3) for each $v \in V$, all nodes in T that cover v are connected; (4) for the root of T η_r, $V_x \subseteq V_{\eta_r}$.

(R0) $\dfrac{[A \to R, \eta_r, J, \psi]}{[A, J, \psi(X_R)]}$ **(R1)** $\dfrac{}{[A \to R, \eta, \emptyset, \emptyset]}$ **(R3)** $\dfrac{[A \to R, \eta_1, I, \varphi_1][A \to R, \eta_2, J, \varphi_2]}{[A \to R, \eta, I \cup J, \varphi_1 \cup \varphi_2]}$

(R2.T) $\dfrac{[A \to R, \eta_1, I, \varphi_1]}{[A \to R, \eta, I \cup e^*, \varphi_1 \cup \{e \mapsto e^*\}]}$ **(R2.NT)** $\dfrac{[A \to R, \eta_1, I, \varphi_1][\ell(e), J, X]}{[A \to R, \eta, I \cup J, \varphi_1 \cup \{e \mapsto X\}]}$

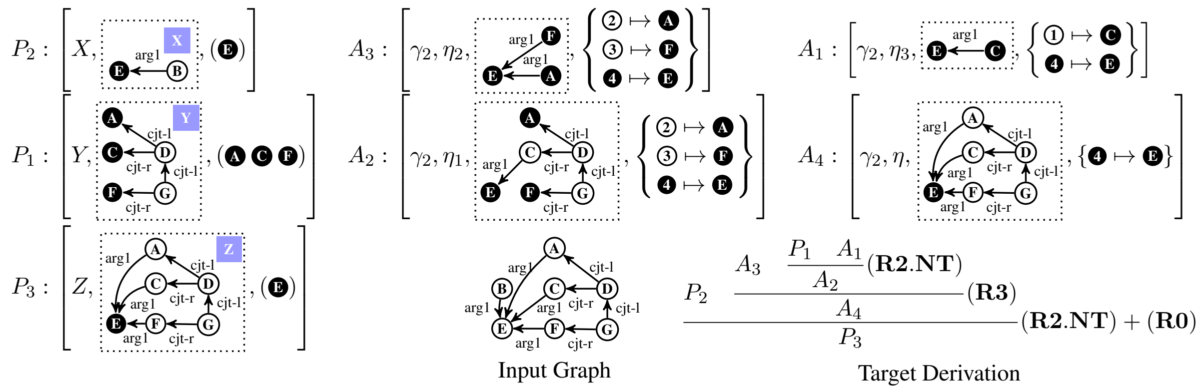

Figure 5: A sketch of the inference rules and how Chiang et al.'s algorithm works as chart parsing.

the form $[A, J, B_x]$ where J is a subgraph of G which can be derived initially from some rule with A as LHS ($A \Rightarrow^* J$) and B_x is an explicit ordering of $bn(J)$. An **active item** is of the form $[A \to R, \eta, I, \varphi]$ where η is in T_R, I is a subgraph of G which derives from $R_{\trianglerighteq\eta}$ and φ is the bijection from $bn(R_{\trianglerighteq\eta})$ to $bn(I)$. A small number of inference rules (as shown in Fig. 5) are sufficient to control merging the chart items. **R0** is applied on the root node of T_R. **R1**, **R2.T**, **R2.NT** and **R3** are applied on leaf nodes, unary nodes that introduce a terminal edge, unary nodes that introduce a nonterminal edge and binary nodes respectively. e^* is an edge of G such that $\ell(e) = \ell(e^*)$. $\{e \mapsto e^*\}$ or $\{e \mapsto X\}$ reprensets the mapping that sends each node of e to the corresponding node of e^* or X. $\psi(X_R)$ denotes a list generated by applying ψ on each node of X_R in order. Refer to the original paper for a complete description of the algorithm. See the bottom part of Fig. 5 for a partial recognition along with T_1 in Fig. 4.

3.2 Treewidth-centric Complexity Analysis

It is an advantage of using tree decomposition that the treewidth of a grammar leads to a bound on the number of boundary nodes which we must keep track of during parsing. When applying an inference rule at η, all mentioned boundary nodes are called **active nodes** and denoted as $A(\eta)$. $A(\eta) = bn(R_{\trianglerighteq\eta_1}) \cup bn(R_{\trianglerighteq\eta_2})$ if η is binary, and $A(\eta) = bn(R_{\trianglerighteq\eta_1}) \cup V(e)$ otherwise. Let k be the treewidth of a grammar and d be the maximum degree of any node in the input graph. The number of rule instantiations at η is actually in $\mathcal{O}(n^{|A_\eta|}3^{d|A_\eta|})$. The first part $n^{|A_\eta|}$ is the number of ways of mappings between active nodes in a rule and nodes in an input graph. The second part $3^{d|A_\eta|}$ is an upper bound of realizations for boundary edges. Chiang et al. proves that $A(\eta) \subseteq V_\eta$, implying that $k+1$ is an upper bound of $|A(\eta)|$. Therefore, the time complexity is in $\mathcal{O}((3^d n)^{k+1})$. The space complexity is in $\mathcal{O}((2^d n)^{k+1})$ by a parallel analysis.

3.3 Measuring Practical Performance

Successful integration of two chart items according to an inference rule requires that the items are disjoint and can make up a new bijection. When two chart items pass the check, the following *successful integration* is viewed as a successful rule instantiation, and in this case, the operation cost is taken into account. When two chart items fail to pass the check, there will be no successful rule instantiation, and in this case, the operation cost for this failed integration is overlooked by the treewidth-centric complexity analysis. The cost to figure out an integation is impossible is actually comparable to that of a successful integation.

Measuring practical performance with respect to both successful and failed integration operations is a necessary complement to the theoretical analysis, especially when the number of failed integrations is prominent. In the following experiments, we will report the exact numbers for successful (indicated as **#Succ**) and total (=successful+failed; indicated as **#Total**) integrations.

3.4 Evaluation with a Realistic Grammar

To profile the parsing algorithm, we conduct experiments on the Elementary Dependency Structure (EDS; Oepen and Lønning, 2006) graphs provided by DeepBank v1.1 (Flickinger et al., 2012). The data is separated into training, development and test sets according to standard setup for string parsing. We get a wide-coverage linguistically-meaningful grammar[2] by applying the grammar extraction algorithm described in Chen et al. (2018). The grammar is lexicalized (LxG), in that argument-structures are lexically encoded, like almost all popular deep grammars used in NLP. Tab. 1 shows the statistics of the rules.

LxG	#Rule		Treewidth	#Node	#Terminal
Lexical	46,101	avg.	1.07	2.15	2.47
		max.	4	10	18
Phrasal	8,594	avg.	1.62	2.94	0.79
		max.	6	7	10

Table 1: Basic properties of our lexicalized grammar. **#Node** and **#Terminal** indicate the numbers of nodes and terminal-edges in a single rule.

Referring to Bolinas[3], we re-implement the algorithm in C++ and test its efficiency on 4500 EDS graphs that are randomly selected from the training set with the size in the range of 5 to 50. By size of a graph, we mean the number of its nodes. If the number of total subgraphs allocated during parsing is larger than 2.6×10^7, the parser will throw an out-of-memory error (OOM). In all the following tables, all statistics are the average values over instances which successfully receive derivation forests. The platform for all experiments is x86 64 GNU/Linux with one Intel(R) Core(TM) i7-5930K CPU at 3.50GHz.

Tab. 2 summarizes the results. For small graphs, the algorithm achieves a promising speed. For larger graphs, most of parsing time is wasted on the failed integrations. Fig. 6 represents the numbers of successful and total integrations. We can clearly see that the difference between the two types of integrations increase very quickly when an input graph is enlarged. In §4.5 we will discuss how to reduce failed integrations.

[2]We only consider rules the RHS of which are connected. A few graphs that are not connected and thus removed. A very small portion of DeepBank graphs result in disconnected rules. These graphs contain arguable annotations related to (1) distributive readings of coordination, (2) quantifier of bare NPs, and/or (3) small clauses. We leave appropriate analysis of these phenomena for future investigation.

[3]www.isi.edu/licensed-sw/bolinas/

#Node	Time(s)	#Succ/#Total	OOM	#Graph
All	21.64	0.21%	305	4500
<10	0.02	12.55%	0	500
10∼20	0.45	1.42%	0	1000
20∼30	9.36	0.34%	4	1000
30∼50	47.68	0.19%	301	2000

Table 2: Performance of our implementation of Chiang et al. (2013). First column is the size of input graphs. Last column is the number of graphs in given range.

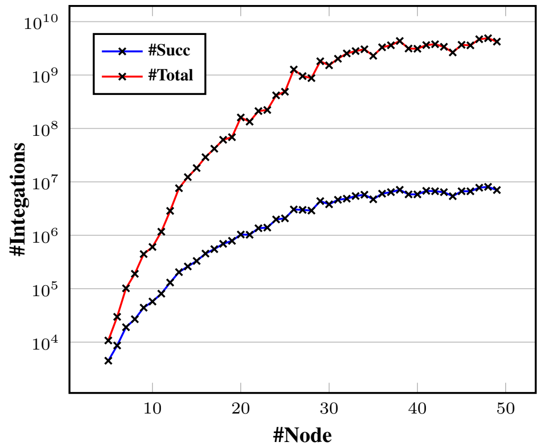

Figure 6: The numbers of successful/total integrations relative to the size of graph. All data points are the average value of multiple graphs of the same size.

4 Speeding Up by Exploiting Locality

4.1 Locality as Edge-Adjacency

Some notion of locality is conceptually necessary for studying complex structures. Adjacency is a key perspective to express locality in some linguistic theories, such as CCG (Steedman, 2000, p. 54):

(1) **The Principle of (String-)Adjacency**
Combinatory rules may only apply to finitely many phonologically realized and string-adjacent entities.

Almost all string parsing algorithms benefit from this string-adjacency. Now let us picture string-adjacency using *a graph language*. Fig. 7 gives a visualization of the linear chain structure of a word sequence. The terminal edge labeled as next in γ_{11} explicitly displays a local relation: ② and ③ being able to be recognized almost simultaneously. String-adjacency turns to be **terminal edge-adjacency** from a graph-theoretic view.

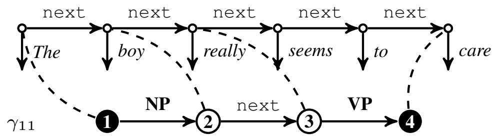

Figure 7: A graph-based view of string-adjacency.

What does terminal edge-adjacency actually mean? From a semiotic perspective of a language system, being either natural or artificial, a key property is *form-meaning* connection. A particular *form* triggers a particular *meaning*. What can be observed can be directly recognized, and then makes other things recognizable. Considering language production, the input is an MR, and in the graph-based framework, it is terminal edges that are directly observable. In this way a terminal edge makes nodes connected to it co-recognizable.

The existing algorithms, including Chiang et al. (2013) and Groschwitz et al. (2015), do not consider terminal edge-adjacency. We will show that capturing locality in this sense is beneficial, just like what successful string parsing algorithms do.

4.2 Locality-centric Complexity Analysis

Some active nodes are not independent with each other if we take terminal edge-adjacency into consideration. We call a graph consisting of only terminal edges a *terminal graph*. For a graph fragment H, we use $term(H)$ to denote the subgraph of H that is induced from all and only terminal edges. We informally illustrate the idea of dependency between nodes in a rule, and then present a precise analysis. Fig 8 is a prototype of a binary node in T_R. ❻ ④ ❾ ⑤ are active nodes of η. But if one of these nodes is identified in an input graph, the possible positions of the other three nodes are highly restricted.

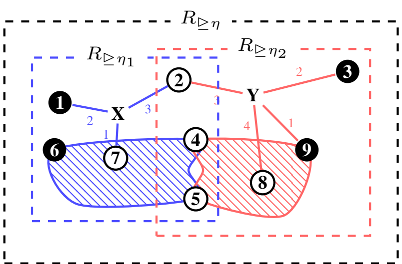

Figure 8: A prototypical case for recognizing a binary node η in T_R. The area in blue represents $term(R_{\trianglerighteq\eta_1})$ and the red one represents $term(R_{\trianglerighteq\eta_2})$. Boundary nodes of each area are placed on the border. The nodes in black are the boundary nodes of $R_{\trianglerighteq\eta}$.

Proposition 1. Consider a graph G and connected terminal graph R_t. If there is a node v_1 in R_t that is tied to a node v_1^* in G, then finding all isomorphisms of R_t in G can be completed in $\mathcal{O}(d^{m_t})$ time, where m_t is the number of edges in R_t and d is the maximum degree of any node in G.

Proof. We perform a depth-first search over R_t starting at v_1 and arranging all edges of R_t as a sequence according to the order in which they are visited. Let the edge sequence be $e_1, e_2, ..., e_{m_t}$. We match edges in this sequence one by one. When we handle $e_j (1 \leq j \leq m_t)$, there must be a node $v \in V(e_j)$ such that $v = v_1$ or $v \in V(e_k)(1 \leq k < j)$. In other words, v is already tied to a node $v^* \in G$. As a result, the number of possible mappings of e_j is at most d, because the degree of v^* is at most d. Therefore, the number of isomorphisms of R_t is in $\mathcal{O}(d^{m_t})$. As a result, all isomorphisms can be found in $\mathcal{O}(d^{m_t})$ time. □

When l active nodes locate in a connected component of $term(R_{\trianglerighteq\eta})$, these nodes are somewhat **dependent**. By Proposition 1, the number of valid node mappings of these l nodes is bounded by $\mathcal{O}(nd^{m_t})$ rather than $\mathcal{O}(n^l)$.

Definition 1. For any node η in T_R, $\delta(\eta)$ denotes the size of a maximal subset of $A(\eta)$ such that all nodes in this subset is independent with each other. We use $S(\eta)$ to denote one of such maximal subsets. Similar to treewidth, we define $\delta(T_R) = \max_{\eta \text{ in } T_R} \delta(\eta)$ and $\delta(R)$ as the minimum δ of any tree decomposition of R.

In Fig. 8, we have $\delta(\eta) = 4$ and {② ❸ ❶ ❻} is a maximal subset of $A(\eta)$ as required.

Proposition 2. For any graph fragment R, $\delta(R) \leq k + 1$ where k is the treewidth of R.

Proof. This proposition is trivial. For any η, we have $\delta(\eta) \leq |A_\eta| \leq |V_\eta| \leq k+1$ (Proposition 3 in Chiang et al.). By the definition of $\delta(R)$, we have $\delta(R) \leq \delta(T_R) = \max_{\eta \text{ in } T_R} \delta(\eta) \leq k + 1$. □

Proposition 3. The number of ways of instantiations of any inference rule is in $\mathcal{O}(n^{\delta^*} d^{m_g} 3^{dn_g})$, where n_g/m_g is the maximum count of nodes/terminal-edges of any RHS in $\mathcal{G}$ and δ^* is the maximum δ of any RHS in $\mathcal{G}$.

Proof. When applying an inference rule on η, we first select the mappings for nodes in $S(\eta)$ independently. According to the definition of $S(\eta)$, for an active node $v \notin S(\eta)$, there must be a node

$u \in S(\eta)$ such that u and v belong to the same connected component c of $term(R_{\trianglerighteq\eta})$. Because u is already bounded, the number of isomorphisms of c is in $\mathcal{O}(d^{m_c})$ where m_c is the number of edges in c. By enumerating nodes like v, we can get a set of connected components consisting of only terminal edges. The number of isomorphisms of all these components is in $\mathcal{O}(\prod_c d^{m_c}) \leq \mathcal{O}(d^{m_g})$. Therefore, the number of possible mappings for all active nodes is in $\mathcal{O}(n^{|S(\eta)|} d^{m_g}) \leq \mathcal{O}(n^{\delta^*} d^{m_g})$. The analysis for boundary edges is similar to Chiang et al.'s. The only difference is that the tree decomposition which minimizes δ may not minimize the treewidth k. Since $k \leq n_g - 1$, the number of ways of boundary edges is in $\mathcal{O}(3^{dn_g})$. □

We can conclude from Proposition 2 and 3 that our locality-centric analysis is tighter than the treewidth-centric one, and the upper bound of time complexity may decrease for some restricted HRGs. In Fig. 4, the treewidth of T_2 is 3, but $\delta(T_2) = 1$. So the number of rule instantiations that can be applied along with T_2 is in $\mathcal{O}(n)$ instead of $\mathcal{O}(n^4)$. In §4.3, we will introduce Weakly Regular Graph Grammar (WRGG), a new subclass of HRG, the δ of which is more intuitively understandable.

4.3 Weakly Regular Graph Grammar

We discuss prototypes of HRG rules, investigating their key properties in a linguistic context. We then formally define WRGG that reflects the linguistic emphasis and also show that WRGG is actually a very expressive subclass of HRG.

Firstly, the HRG rule under discussion allows at most two non-terminals at RHS. Computationally speaking, we can transform a multi-branching rule into multiple binary rules without loss of expressiveness, as we are able to get a CFG in Chomsky Normal Form for any CFG. Linguistically speaking, multi-branching rules have been removed from generative linguistic theories, since at least Minimalist Program (Chomsky, 1995). Fig. 9 presents four prototypes with the binary constriction. γ_3, γ_6, γ_7, γ_8 and γ_9 in Fig. 1 are of T0, γ_1, γ_4, γ_5 and γ_{10} are of T1, and γ_2 is of T3. Secondly, for a lexicalized grammar, most rules are of T0 or T1, since constructions barely take *semantic materials*. If a rule introduces *heavy* constructional meaning, it may affect one of its intermediate constituents (T2) or bridge the meanings between both of its intermediate constituents (T3),

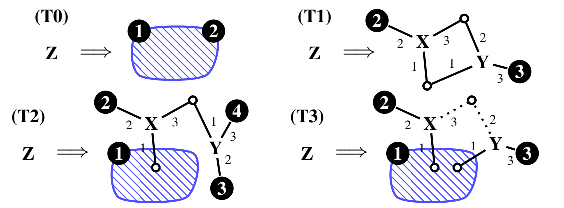

Figure 9: Prototypes of binary HRG rules: **(T0)** lexical rules, **(T1)** rules without terminal edges, **(T2)** only one nonterminal edge of the rules contains non-free nodes, and **(T3)** both nonterminal edges of the rules contain non-free nodes. The area in blue represents all terminal edges. Dashed edges indicate optionality.

and hardly affect its intermediate constituent separately. Even though a rule has multiple terminal components, we can replace it with several rules of T0-T3. Thirdly, a node that is only connected to a nonterminal edge is a kind of placeholder, in that it does not affect current semantic composition but will be used in future. Otherwise it has been removed in a previous step. Finally, we do not consider disconnected RHS because it yields disconnected graphs.

Definition 2. A node v in an edge-labeled graph G is **free**, if $E(v)$ contains only nonterminal edge(s). The number of those nodes is denoted by $f(G)$.

In Fig. 8, ❶ ② ❸ are free nodes of $R_{\trianglerighteq\eta}$.

Definition 3. A weakly regular rule $A \to R$ satisfies the following conditions: (1) R is connected; (2) $term(R)$ is an empty graph or a connected graph; (3) if a free node of R is incident to only one edge, it is also an external node.

Definition 4. An HRG is weakly regular, if all of its rules are weakly regular.

Proposition 4. If $A \to R$ is binary and weakly regular, then $\delta(R) = f(R)$ or $f(R) + 1$.

The proofs of this proposition can be found in the appendix. The tree shown in Fig. 10 is a valid nice tree decomposition of R and the δ of the tree is $f(R)$ or $f(R) + 1$. We argue that for parsing with a binary WRGG, the number of free nodes is more meaningful and we can use the tree decomposition shown in Fig. 10 rather than a tree decomposition with minimum treewidth.

Courcelle (1991) introduces Regular Graph Grammar (RGG). It is provable that RGG is a subclass of WRGG. There are no free nodes in RGG and graph parsing with an RGG can be finished in linear time by applying Chiang et al.'s algorithm.

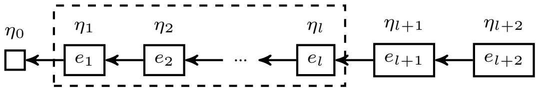

Figure 10: A terminal edge-first tree decomposition of a binary and weakly regular rule. For every node $\eta_i (1 \le i \le l+2)$, $V_{\eta_i} = bn(R_{\trianglerighteq \eta_i}) \cup V(e_i)$ and $E_{\eta_i} = \{e_i\}$. $e_1, \dots, e_l$ are terminal edges ordered by visiting time of a depth-first traversal. e_{l+1}, e_{l+2} are nonterminal edges arranged in order such that $R_{\trianglerighteq \eta_{l+1}}$ is also a connected graph.

This result is comparable to another algorithm proposed by Gilroy et al. (2017). However, the strong restrictions of `RGG` make it too weak to model linguistic structures. `WRGG` is much more linguistically adequate.

4.4 Distributed Argument-Structure

We value the trigger role played by terminal edges in an `HRG` rule. Now let us revisit the derivation governed by a lexicalized grammar. It is obvious that lexical rules try to use up all terminal edges at the initial stage of syntactico-semantic composition. If we can distribute terminal edges to all rules, both lexical and phrasal, we are able to get a reduced number of free nodes on average and in exactly this way improve graph parsing remarkably. The idea to distribute argument-structures exhibits a constructivist perspective, which is a competing hypothesis to lexicalism that dominates our field for dozens of years, since at least Bresnan and Kaplan (1982). The constructivist approaches to argument structures have been recently discussed by different theoretical linguistic theories, including but not limited to Distributed Morphology (Halle and Marantz, 1993, 1994) and Sign-Based Construction Grammar (Boas and Sag, 2012). The emphasis on the advantage of *Distributed Argument-Structure* under the consideration of language production is a computational support for many constructivist approaches.

Fig. 11 demonstrates a derivation with a construction grammar. Compare γ_{12} to γ_4 and γ_{13} to γ_5, we can clearly see that δ is significantly reduced. A comparison of lexical rules also confirms the importance of distributed argument-structure.

4.5 Fast Accessing of Chart Items

We will complete our discussion on locality by considering the *edge-zero* case, i.e. unifying nodes. In Fig. 8, when we try to integrate $R_{\trianglerighteq \eta_1}$ and $R_{\trianglerighteq \eta_2}$, we must make sure that the three nodes

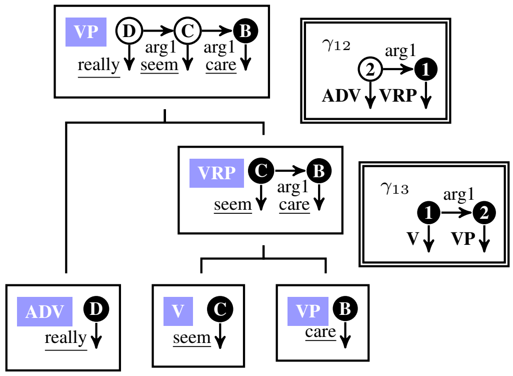

Figure 11: Semantic composition with a CxG.

on the boundary, viz. ②, ④ and ⑤, are identical in terms of mappings relative to η_1 and η_2 respectively. Otherwise, a failure occurs. In both cases, trying to unify them causes a bottleneck for graph parsing, as conceptually suggested in §3.3 and empirically confirmed by Tab. 2.

Considering the above problem in the framework of chart parsing, we would like to construct a data structure to efficiently access all chart items. In particular, when partial information is provided, this data structure can quickly find all compatible chart items. In this paper, we use a map, with the keys being partial information for quiry and the values being sets of chart items. The implementation used in §3.4 follows the method proposed by Chiang et al. (2013), only mentioning $\ell(e)$ or η for indexing, which is not efficient in practice. We propose to build a more comprehensive map. See Tab. 3 for an example of our map.

Indexing key(s)	**Item**
⟨Y, 3, {⟨1, Ⓐ⟩, ⟨2, Ⓒ⟩, ⟨3, Ⓕ⟩}⟩ ⟨Y, 3, {⟨1, Ⓐ⟩, ⟨3, Ⓕ⟩}⟩ ⟨Y, 3, {⟨2, Ⓒ⟩, ⟨3, Ⓕ⟩}⟩ ⟨Y, 3, {⟨1, Ⓐ⟩}⟩ ⟨Y, 3, {⟨2, Ⓒ⟩}⟩ ⟨Y, 3, {⟨1, Ⓐ⟩, ⟨2, Ⓒ⟩}⟩ ⟨Y, 3, {⟨3, Ⓕ⟩}⟩	P_1
⟨X, 1, {⟨1, Ⓔ⟩}⟩	P_2, A_4
⟨η, {⟨②, Ⓐ⟩, ⟨③, Ⓕ⟩, ⟨❹, Ⓔ⟩}⟩	A_2, A_3

Table 3: Examples for indexing chart items in Fig. 5. P_1 has multiple keys.

During recognizing η, the set of nodes which connect branching subgraph(s) $C(\eta)$ is $bn(R_{\trianglerighteq \eta_1}) \cap bn(R_{\trianglerighteq \eta_2})$ for binary case, and $bn(R_{\trianglerighteq \eta_1}) \cap V(e)$ for unary case. Let $e = (v_1, ..., v_{|V(e)|})$ denote a hyperedge and $index(e, v_i) = i$ denote an indexing function. For a list of nodes B, $B[i]$ denotes its i-th node. A passive item $[A, J, B_x]$ has multiple indexing keys. For a non-empty set of positive integers $mask \subseteq \{1, 2, ..., |bn(J)|\}$, $\langle A, |bn(J)|, \{\langle i, B_x[i]\rangle | i \in mask\}\rangle$ is a plausible key. For active item $[*, \eta_1, I, \varphi]$, let η be the parent of η_1. If η is binary, the item should be indexed by $\langle \eta_1, \{\langle v, \varphi(v)\rangle | v \in C(\eta)\}\rangle$. Otherwise η is unary and introduces some edge e. The item should be indexed by $\langle \ell(e), |V(e)|, \{\langle index(e, v), \varphi(v)\rangle | v \in C(\eta)\}\rangle$. During parsing, two items will be integrated only when they have the same key.

Note that the number of possible $mask$'s for a passive item grows exponentially w.r.t. the number of the corresponding external nodes. However a significant number of $mask$'s are not used by any tree decomposition of any rule. And such $mask$'s can be found by processing a grammar before parsing. For all HRGs used for experiments, the maximum number of *useful* $mask$'s for a passive item is 15.

4.6 Empirical Evaluation

A construction grammar (CxG) is automatically induced in a similar way to the experiments in §3.4. Note that our grammar extraction procedure makes sure that this grammar is weakly regular. As shown in Tab. 4, the average number of free nodes in CxG is much smaller. We conduct new experiments using the improvements mentioned in previous sections. We re-run the improved parser on 4195 EDS graphs, which can successfully receive derivation forests from the original parser. Tab. 5 and Fig. 12 show the effectiveness of our improvements. The terminal-first tree decomposition (as illustrated in Fig. 10) is able to significantly reduce the number of integrations. Our indexing method can effectively reduce the number of failed integrations. For the CxG, using the terminal-edge first strategy is more effective than the indexing strategy. Note that the cost to build a map for indexing chart items is not ignorable.

		#Rule		Treewidth	δ	#Free
CxG	Lexical	34,348	avg.	0.36	—	—
			max.	4	—	—
	Phrasal	7,978	avg.	1.68	**1.59**	**0.59**
			max.	7	7	6
LxG	Phrasal	8,594	avg.	1.62	2.51	2.27
			max.	6	7	7

Table 4: Statistics of the CxG. **#Free** means the number of free nodes in HRG rules.

		Time(s)	#Succ	#Total	#Succ/#Total
LxG	original	21.64	303.6	146535	0.21%
	+terminal-first	21.02	174.9	145320	0.12%
	+index	1.93	303.6	7165	4.24%
	+both	1.51	174.9	6923	2.53%
CxG	original	0.41	61.4	1082	5.67%
	+terminal-first	0.12	9.0	406	2.23%
	+index	0.32	61.4	190	32.34%
	+both	0.07	9.0	31	29.34 %
	large (+both)	0.45	50.1	485	10.34 %
	305 (+both)	0.32	35.5	379	9.40 %

Table 5: Performance of our implementation with improvements. **The unit of integrations is 10^4 in the table.** *terminal-first* means the terminal-first tree decomposition; *index* means the method proposed in §4.5; *both* means to use both *terminal-first* and *index*. *large* means to test the algorithm on 189 graphs with the size in the range of 70 to 90. *305* means to test the algorithm on the 305 graphs which receives an OOM error in previous experiment (§3.4).

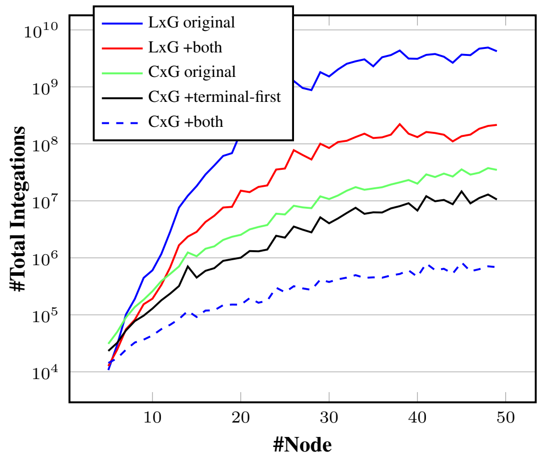

Figure 12: The number of total integrations relative to size of input graphs. All data points in the plot are the average value on test samples of a given size.

5 Conclusion

We introduce several locality-centric refinements to advance graph parsing and empirically evaluate their effectiveness. We show that exact graph parsing can be efficient even for large graphs and with large graph grammars.

Acknowledgments

This work is supported in part by the National Hi-Tech R&D Program of China (No. 2018YFC0831900). Weiwei Sun is the corresponding author.

References

Hans Christian Boas and Ivan A Sag. 2012. *Sign-based construction grammar*. CSLI Publications/Center for the Study of Language and Information.

Joan Bresnan and Ronald M. Kaplan. 1982. Introduction: Grammars as mental representations of language. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages xvii–lii. MIT Press, Cambridge, MA.

Yufei Chen, Weiwei Sun, and Xiaojun Wan. 2018. Accurate shrg-based semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 408–418. Association for Computational Linguistics.

David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. 2013. Parsing graphs with Hyperedge Replacement Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 924–932, Sofia, Bulgaria. Association for Computational Linguistics.

Noam Chomsky. 1995. *The Minimalist Program*. MIT Press, Cambridge, MA.

Bruno Courcelle. 1991. The monadic second-order logic of graphs v: On closing the gap between definability and recognizability. *Theoretical Computer Science*, 80(2):153–202.

F. Drewes, H.-J. Kreowski, and A. Habel. 1997. Hyperedge Replacement Graph Grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 95–162. World Scientific Publishing Co., Inc., River Edge, NJ, USA.

Daniel Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank: A dynamically annotated treebank of the wall street journal. In *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories*, pages 85–96.

Sorcha Gilroy, Adam Lopez, and Sebastian Maneth. 2017. Parsing graphs with regular graph grammars. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*, pages 199–208.

Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2015. Graph parsing with s-graph grammars. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1481–1490.

M Halle and Alec Marantz. 1993. *Distributed morphology and the pieces of inflection*, pages 111–176. The MIT Press.

Morris Halle and Alec Marantz. 1994. Some key features of distributed morphology. *MIT working papers in linguistics*, 21(275):88.

Martin Kay. 1996. Chart generation. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 200–204.

Alexander Koller, Stephan Oepen, and Weiwei Sun. 2019. Graph-based meaning representations: Design and processing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 6–11, Florence, Italy. Association for Computational Linguistics.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based mrs banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC-2006)*, Genoa, Italy. European Language Resources Association (ELRA). ACL Anthology Identifier: L06-1214.

Mark Steedman. 1996. *Surface Structure and Interpretation*. Linguistic Inquiry Monographs. Mit Press.

Mark Steedman. 2000. *The syntactic process*. MIT Press, Cambridge, MA, USA.

A Proof for Proposition 4

We provide the proof for R with two nonterminal edges: e_X and e_Y.

Firstly, we prove $\delta(R) \geq f(R)$**.** For any nice tree decomposition T_R of R, let η_m be the node with minimum height such that $R_{\trianglerighteq\eta_m}$ contains both e_X and e_Y.

[1] η_m is binary. Let η_1, η_2 be the two children of η_m. Without loss of generality, we assume $R_{\trianglerighteq\eta_1}$ contains e_X and $R_{\trianglerighteq\eta_2}$ contains e_Y.

[2] η_m is unary. Let η_1 be the only child of η_m. In this case, η_m introduces either e_X or e_Y. Without loss of generality, we assume η_m introduces e_X.

Let v be a free node of R.

Case 1 v is incident to only one of e_X and e_Y. By property (3) of weakly regularity, v is an external node of R. Therefore, $v \in bn(R) \subset bn(R_{\trianglerighteq\eta_1}) \subset A(\eta_m)$.

Case 2 v is incident to both e_X and e_Y. When η_m is binary ([1]), we have $v \in bn(R_{\trianglerighteq\eta_1}) \cap bn(R_{\trianglerighteq\eta_2}) \subset A(\eta_m)$. When η_m is unary ([2]), we have $v \in V(e_X) \subset A(\eta_m)$.

By the above discussion, we conclude that all free nodes of R are active nodes of η_m and it is obvious that free nodes are independent. As as result, we have $\delta(T_R) \geq \delta(\eta_m) \geq f(R)$. The arbitrariness of T_R ensures that $\delta(R) \geq f(R)$.

Secondly, we prove that $\delta(R) \geq f(R) + 1$ **for prototype T3.** If the rule is type **T3**, then there exist two nodes v, u such that v is incident with e_X, u is incident with e_Y and u, v are in $term(R)$.

Case 1 $u = v$. We have $u \in bn(R_{\trianglerighteq\eta_1}) \cap bn(R_{\trianglerighteq\eta_2}) \subset A(\eta_m)$.

Case 2 $u \neq v$ and η_m is unary ([2]). We have $v \in V(e_X) \subset A(\eta_m)$.

Case 3 $u \neq v$ and η_m is binary ([1]). According to the property (2) of weakly regularity, $term(R)$ is connected. So there exists a path $e_1, e_2, ..., e_s (s \geq 1)$ in $term(R)$ such that $u \in V(e_1)$ and $v \in V(e_s)$. Let i be the minimum index such that e_i is not in $R_{\trianglerighteq\eta_1}$. If $i = 1$, then u has an edge e_1 which is not in $R_{\trianglerighteq\eta_1}$. Therefore, $u \in bn(R_{\trianglerighteq\eta_1}) \subseteq A(\eta_m)$. If $s \geq i > 1$, then all nodes inside $V(e_{i-1}) \cap V(e_i)$ are boundary nodes of $R_{\trianglerighteq\eta_1}$. Therefore these nodes all belong to $A(\eta_m)$. If i does not exist, then v has an edge e_s which is not in $R_{\trianglerighteq\eta_2}$. Therefore, $v \in bn(R_{\trianglerighteq\eta_2}) \subseteq A(\eta_m)$.

By the above discussion, there is at least one active node which is not a free node. It is trivial that the node is independent with any free nodes. Therefore, $\delta(T_R) \geq \delta(\eta_m) \geq f(R) + 1$. The arbitrariness of T_R ensures that $\delta(R) \geq f(R) + 1$.

Thirdly, we prove that the equality can be achieved. It is trivial to prove that the tree T shown in Fig. 10 is a valid nice tree decomposition by going through the properties of tree decomposition. Since $R_{\trianglerighteq\eta_i}$ $(1 \leq i \leq l)$ is a connected terminal graph, we have $\delta(\eta_i) = 1$. By going through the four possible prototypes of current rule shown in Fig. 9, we conclude that $\delta(\eta_j) \leq f(R) + 1$, for $l + 1 \leq j \leq l + 2$. Therefore, $\delta(R) \leq \delta(T) = \max_{\eta \text{in} T} \delta(\eta) \leq f(R) + 1$.

In summary, we have $f(R) \leq \delta(R) \leq f(R) + 1$.