Finding Key Structures in MMORPG Graph with Hierarchical Graph Summarization

JUN-GI JANG, Seoul National University, Republic of Korea
CHAEHEUM PARK*, Deeping Source, Republic of Korea
CHANGWON JANG, NCSOFT Co., Republic of Korea
GEONSOO KIM, NCSOFT Co., Republic of Korea
U KANG†, Seoul National University, Republic of Korea

What are the key structures existing in a large real-world MMORPG (Massively Multiplayer Online Role-Playing Game) graph? How can we compactly summarize an MMORPG graph with hierarchical node labels, considering substructures at different levels of hierarchy? Recent MMORPGs generate complex interactions between entities inducing a heterogeneous graph where each entity has hierarchical labels. Succinctly summarizing a heterogeneous MMORPG graph is crucial to better understand its structure; however it is a challenging task since it needs to handle complex interactions and hierarchical labels efficiently. Although there exist few methods to summarize a large-scale graph, they do not deal with heterogeneous graphs with hierarchical node labels.

We propose GSHL, a novel method that summarizes a heterogeneous graph with hierarchical labels. We formulate the encoding cost of hierarchical labels using MDL (Minimum Description Length). GSHL exploits the formulation to identify and segment subgraphs, and discovers compact and consistent structures in the graph. Experiments on a large real-world MMORPG graph with multi-million edges show that GSHL is a useful and scalable tool for summarizing the graph, finding important structures in the graph, and finding similar users.

CCS Concepts: • **Information systems** → **Massively multiplayer online games**; **Data mining**; • **Mathematics of computing** → **Graph algorithms**.

Additional Key Words and Phrases: Graph summarization, minimum description length, hierarchical label, massively multiplayer online role-playing game

1 INTRODUCTION

What does a large real-world MMORPG (Massively Multiplayer Online Role-Playing Game) graph look like? What are the key structures existing in a large real-world MMORPG (Massively Multiplayer Online Role-Playing Game) graph? How can we compactly summarize an MMORPG graph with hierarchical node labels, considering substructures at different levels of hierarchy? A large number of interactions between heterogeneous entities appear in MMORPGs, and they are often represented as a heterogeneous graph. For example, a user has several

*The work was done while Chaeheum Park was at Seoul National University.
†Corresponding author.

Authors' addresses: Jun-Gi Jang, Seoul National University, Seoul, Republic of Korea, elnino4@snu.ac.kr; Chaeheum Park, Deeping Source, Seoul, Republic of Korea, chaeheum@snu.ac.kr; Changwon Jang, NCSOFT Co., Seongnam, Republic of Korea, jangcw@ncsoft.com; Geonsoo Kim, NCSOFT Co., Seongnam, Republic of Korea, geostatman@ncsoft.com; U Kang, Seoul National University, Seoul, Republic of Korea, ukang@snu.ac.kr.

1556-4681/2022/3-ART $15.00
https://doi.org/10.1145/3522691

characters, and each character with various equipment visits dungeons. Moreover, each entity has hierarchical labels (e.g., character-dealer-destroyer and equipment-weapon). Summarizing a heterogeneous graph with hierarchical node labels is a crucial problem to understand its characteristics.

Fig. 1 shows the motivation of hierarchical graph summarization. If we ignore hierarchical label information, we may find the structure shown in Fig. 1 (a) which shows that an equipment Soul is shared merely by a group of characters in an MMORPG. On the other hand, if we consider the hierarchical label information, we find that the structure is divided into three subgroups, where each group clearly represents the relation between the equipment and the jobs of characters. For example, the job which uses the equipment Soul the most is Zen Archer, and the Dealer characters (shown in Fig. 1 (b,c)) use the equipment more than the Tanker and the Buffer characters (shown in Fig. 1 (d)) do. Such discovery helps us obtain meaningful insights in the in-game interaction, and design a better user experience in the game.

Several approaches have been proposed to summarize graphs using significant structures (vocabularies) commonly found in real-world graphs. VoG [11] is the first approach to summarize a homogeneous graph with vocabulary terms by compressing the graph in terms of Minimum Description Length (MDL) principle. TimeCrunch [18] extends VoG to summarize dynamic graphs. Both methods, however, fail to summarize a heterogeneous graph with hierarchical labels since they do not consider labels of nodes nor the hierarchy of labels.

We propose GSHL (Graph Summarization with Hierarchical Labels), a summarization method for a heterogeneous graph with hierarchical node labels. Based on the MDL principle that good compression leads to good summarization, we precisely define the encoding cost for heterogeneous graphs with hierarchical node labels. GSHL decomposes a heterogeneous graph into subgraphs using a graph decomposition method, and encodes each subgraph as a structure (e.g., star, clique, bipartite core, chain, etc.). Then GSHL further segments each structure by considering hierarchical labels of nodes in the structure, and summarizes the graph using the encoding cost. Thanks to GSHL, we analyze a large real-world MMORPG graph and find its key structures as well as similar users. Our contributions are as follows:

(1) **Problem formulation.** We formulate the problem of summarizing a heterogeneous graph with hierarchical labels. We use a large real-world graph extracted from a popular real-world MMORPG graph. This is the first work in literature that summarizes a large real-world MMORPG graph, to the best of our knowledge.
(2) **Scalable Method.** We propose GSHL, a novel algorithm to summarize a heterogeneous graph with hierarchical node labels. GSHL carefully exploits MDL to encode hierarchical labels, and generates a succinct summary by segmenting subgraphs at different levels of hierarchy. GSHL is near-linear on the number of edges (see Fig. 12).
(3) **Experiment.** Experiments on the real-world MMORPG graph reveals that GSHL is a useful tool for succinctly summarizing the graph (see Table 4). GSHL discovers interesting patterns (see Fig. 1 and 11), and similar users with similar structures (see Fig. 2).

The rest of this paper is organized as follows. We present preliminaries in Section 2. In Section 3, we describe our proposed MDL formulation for hierarchical graph summarization and our proposed graph summarization algorithm GSHL for heterogeneous graphs with hierarchical node labels. We present the details of the MMORPG data that we studied, and experimental results in Section 4. After discussing related works in Section 5, we conclude in Section 6.

2 PRELIMINARIES

In this section, we describe preliminaries of Minimum Description Length (MDL) (Section 2.1) and Vocabulary-based graph summarization (Section 2.2). Table 1 shows the symbols used.

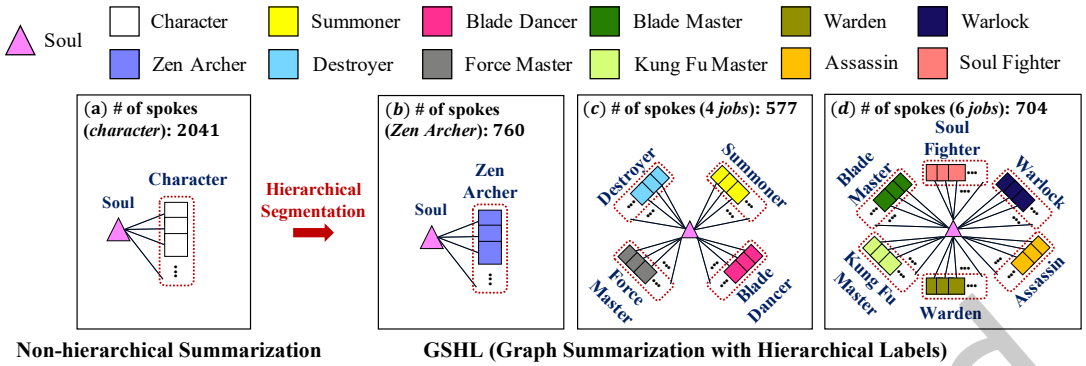
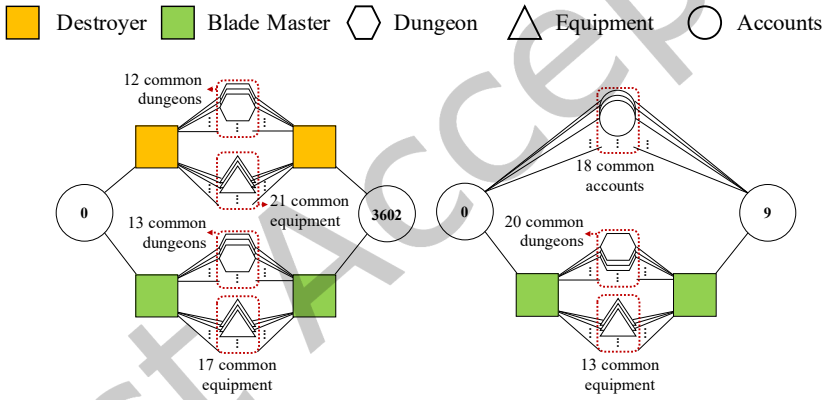


Fig. 1. Our proposed hierarchical graph summarization method GSHL finds more precise and detailed graph structures, compared to non-hierarchical graph summarization. (a) Non-hierarchical graph summarization only finds that an equipment is shared by many characters. It does not recognize the underlying substructures in the characters. (b, c, d) GSHL finds structures in the lower-level labels of the characters and decomposes the character group into three major substructures with different subgroups.



(a) Relation between id 0 and id 3602. (b) Relation between id 0 and id 9.

Fig. 2. The two most similar accounts for account id 0 share similar characters which have similar equipment and visited dungeons. They often share common friends as well.

2.1 MDL for Homogeneous Graph Summarization

Minimum Description Length (MDL) [15] is a model selection method where the best model $\hat{M}$ is the one that gives the best lossless compression:

$$\hat{M} = \arg\min_{M \in \mathcal{M}} (L(M) + L(D|M))$$

M is a model, $\mathcal{M}$ is the set of all possible models, D is the given data, $L(M)$ is the length in bits of M, and $L(D|M)$ is the length in bits of D when encoded using M.

Table 1. Symbol description.

Symbol	Description
$G = (\mathcal{V}, \mathcal{E})$	Graph with node set $\mathcal{V}$ and edge set $\mathcal{E}$
Ω	Set of structure types
st	Star
ch	Chain
fc	Full clique
nc	Near clique
bc	Full bipartite core
nb	Near bipartite core
M	Model
$\mathcal{M}$	Set of all possible models
$\mathbf{A}$	Adjacency matrix of G
s	Structure
$area(s)$	Edges of G included in a structure s
$\mathbf{M}$	Approximate adjacency matrix of $\mathbf{A}$ induced by M
$\mathbf{E}$	Error matrix, $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$
$\oplus$	Exclusive OR
$L(G, M)$	# of bits to encode G using M
$L(M)$	# of bits to encode model M
$L_t(s)$	# of bits to encode a structure s
$L_a(s)$	# of bits to encode node labels in a structure s
$\|s\|$	# of nodes in a structure s
h	# of levels
$\mathbf{E}^+$	Edges that are over-modeled
$\mathbf{E}^-$	Edges that are under-modeled
$L(\mathbf{E}^+)$	Error encoding cost for $\mathbf{E}^+$
$L(\mathbf{E}^-)$	Error encoding cost for $\mathbf{E}^-$
$L(\mathbf{E}^a)$	Encoding cost for labeling error

2.2 Vocabulary-based summarization of Graphs (VoG)

We introduce VoG [11] (Vocabulary-based summarization of Graphs), which summarizes a homogeneous graph with key structures. A set Ω of structure types used in VoG includes the following six key structure types: star, clique, near clique, bipartite core, near bipartite core, and chain. VoG addresses the following problem with the key structures.

Problem 1 (Minimum Graph Description).
Given a homogeneous graph G, its adjacency matrix $\mathbf{A}$, and set Ω of structure types , find model M that minimizes the total encoding length

$$L(G, M) = L(M) + L(\mathbf{E})$$

where $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ $\mathbf{E}$ is the derived error matrix, and $\mathbf{M}$ is the approximate adjacency matrix of $\mathbf{A}$ induced by M.

A model $M \in \mathcal{M}$ is an ordered list of graph structures each having a type from Ω. Each structure $s \in M$ corresponds to a certain portion of the adjacency matrix $\mathbf{A}$ describing its node's interconnectivity; there may be overlapped nodes between different structures, but there is no edge overlap.

VoG summarizes a homogeneous graph with the following full encoding length of a model $M \in \mathcal{M}$:

$$L(M) \ = L_{\mathbb{N}}(|M| + 1) + \log \binom{|M| + |\Omega| - 1}{|\Omega| - 1} + \sum_{s \in M} (-\log(P(x(s)|M)) + L_t(s)) \tag{1}$$

VoG encodes 1) the total number of structures in M using $L_{\mathbb{N}}$, Rissanen's optimal encoding [16] for integers greater than 0, 2) the number of structures for each type using an index over a weak number composition for each structure type $x \in \Omega$ in model M, and 3) the structure type $x(s)$ for each structure $s \in M$ using optimal prefix code, and 4) its connectivity $L_t(s)$. The detailed encodings for connectivity are described in the following section.

2.2.1 Encoding Connectivity. In this section, we introduce how VoG encodes the connectivity of each structure s and derive its cost $L_t(s)$.

Stars. A star has a distinct characteristic of having a single "hub" node surrounded by 2 or more "spoke" nodes. VoG computes $L_t(st)$ of a star st as follows, where $|st|$ is the number of nodes inside the star and $|\mathcal{V}|$ is the number of nodes inside the given graph G:

$$L_t(st) \;=\; L_{\mathbb{N}}(|st| - 1) + \log|\mathcal{V}| + \log\binom{|\mathcal{V}| - 1}{|st| - 1}$$

VoG encodes 1) the number of spokes in the star, 2) the index of the hub node over all $|\mathcal{V}|$ nodes, and 3) which nodes are included in the star's spokes (spoke ids).

Cliques. A full clique is a subset of vertices where every two distinct vertices are adjacent. VoG computes $L_t(fc)$ of a full clique fc as follows, where $|fc|$ is the number of nodes inside the full clique:

$$L_t(fc) \;=\; L_{\mathbb{N}}(|fc|) + \log\binom{|\mathcal{V}|}{|fc|}$$

VoG encodes 1) the number of nodes in the full clique, and 2) the index of a permutation to select $|fc|$ nodes out of $|\mathcal{V}|$ nodes.

Near cliques. A near clique has several missing edges from a full clique. VoG computes $L_t(nc)$ of a near clique nc as follows:

$$L_t(nc) \;= L_{\mathbb{N}}(|nc|) + \log\binom{|\mathcal{V}|}{|nc|} + \log(|area(nc)|) + ||nc||\epsilon_1 + ||nc||'\epsilon_0$$

where $|nc|$ represents number of nodes inside nc, and $|area(nc)|$ is the number of edges in nc. VoG uses $||nc||$ and $||nc||'$, respectively, for the number of observable and non-observable edges in nc. ϵ_1 and ϵ_0 represent the lengths of the optimal prefix code for observable and non-observable edges, respectively.

$$\epsilon_1 = -\log(\frac{||nc||}{||nc|| + ||nc||'}) \quad \epsilon_0 = -\log(\frac{||nc||'}{||nc|| + ||nc||'})$$

VoG encodes 1) the number of nodes in the near clique, 2) the index of a permutation to select $|nc|$ nodes out of $|\mathcal{V}|$ nodes, 3) the number of edges, 4) the observable edges using the optimal prefix code, and 5) the non-observable edges.

Bipartite cores. A bipartite core consists of two sets of nodes A and B where edges exist only between the sets and not within them. A full bipartite core is a fully connected bipartite core. VoG computes $L_t(fb)$ of a full bipartite core fb as follows.

$$L_t(fb) \;=\; L_{\mathbb{N}}(|A|) + L_{\mathbb{N}}(|B|) + \log\binom{|\mathcal{V}|}{|A|} + \log\binom{|\mathcal{V}|}{|B|}$$

where VoG encodes 1) the number of nodes in node sets A and B, and 2) ids of nodes in A and B.

Near bipartite cores. A near bipartite core has several missing edges from a full bipartite core. VoG computes $L_t(nb)$ of a near bipartite core nb as follows, similarly to a near clique:

$$L_t(nb) \;= L_{\mathbb{N}}(|nb|) + \log\binom{|\mathcal{V}|}{|nb|} + \log(|area(nb)|) + ||nb||\epsilon_1 + ||nb||'\epsilon_0$$

Chains. A chain is a sequence of nodes where each node is connected to the previous and the next node of it. VoG computes $L_t(ch)$ of a chain ch as follows, where $|ch|$ is the number of nodes in ch:

$$L_t(ch) \;=\; L_{\mathbb{N}}(|ch| - 1) + \sum_{i=1}^{|ch|} \log\left(|\mathcal{V}| - i + 1\right)$$

VoG encodes 1) the number of nodes in the chain, and 2) their ordered connectivity.

Encoding Connectivity Errors. The encoding length $L(\mathbf{E})$ of connectivity error of a model is the following:

$$L(\mathbf{E}) \;=\; L(\mathbf{E}^+) + L(\mathbf{E}^-)$$

$\mathbf{E}^+$ and $\mathbf{E}^-$ are edges that are over-modeled and under-modeled, respectively; $L(\mathbf{E}^+)$ and $L(\mathbf{E}^-)$ are the error encoding costs for $\mathbf{E}^+$ and $\mathbf{E}^-$, respectively. We encode $\mathbf{E}^+$ and $\mathbf{E}^-$ separately as they are likely to have different distributions. Note that we ignore the errors from near cliques and near bipartite cores as they have been encoded exactly. $L(\mathbf{E}^+)$ and $L(\mathbf{E}^-)$ are defined as follows.

$$\begin{aligned} L(\mathbf{E}^+) \;&=\; \log(|\mathbf{E}^+|) + ||\mathbf{E}^+||\epsilon_1 + ||\mathbf{E}^+||'\epsilon_0 \\ L(\mathbf{E}^-) \;&=\; \log(|\mathbf{E}^-|) + ||\mathbf{E}^-||\epsilon_1 + ||\mathbf{E}^-||'\epsilon_0 \end{aligned} \tag{2}$$

We encode first the number of 1s in $\mathbf{E}^+$ or $\mathbf{E}^-$, and the 1s and 0s using the optimal prefix code.

2.2.2 Summarization. We describe how VoG summarizes a given homogeneous graph using the aforementioned encoding costs. VoG first decomposes a given large graph into several subgraphs using SlashBurn [10], a graph decomposition algorithm whose complexity is $O(T(m + n \log n))$ to generate subgraphs, where m is the number of edges, n is the number of nodes, and T is the number of SlashBurn iterations.

After decomposition, VoG encodes each subgraph as one of the key structure types. For each subgraph, VoG 1) computes the encoding cost for all structure types, 2) compares the encoding costs, and 3) encodes the subgraph as the structure type with the minimum encoding cost. They assume that each subgraph corresponds to one structure among the six ones. VoG determines the role of each node in a given subgraph to encode the subgraph as one of the structure types. For example, VoG determines which node is the hub of a star, and which nodes are included in set A or set B of a bipartite core. VoG uses the following criteria to determine the roles.

Stars. VoG encodes the highest-degree node of a subgraph as the hub, and the remaining nodes as the spokes.

Cliques. All nodes have the same structural role in a clique.

Bipartite cores. VoG determines which nodes are included in set A or set B using binary classification for bipartite cores [11, 18]. VoG adds the highest degree nodes to set A, and its neighbors to set B. Then, VoG performs Fast Belief Propagation (FaBP) [12] to propagate classes and add nodes to set A or B.

Chains. VoG 1) randomly picks a node and finds the farthest node using breadth first search (BFS), 2) sets the discovered node as starting node n_s, 3) finds the farthest node from n_s and sets it as end node n_e, and 4) designates the shortest path from n_s to n_e as the initial chain. Additionally, VoG further enriches the initial chain. VoG constructs an induced subgraph by removing all nodes included in the initial chain except for n_e. Then, Vog extends the initial chain by concatenating the shortest path from n_e to the farthest node in the induced subgraph. VoG repeats all the aforementioned processes for different n_s.

Given a subgraph G', VoG computes the local cost of encoding the subgraph with each structure in Ω. When G' is encoded as a structure s, the local cost of the structure s is the summation of $L_t(s)$ and $L(\mathbf{E}_s)$, where $L_t(s)$ is the connectivity cost of the structure s, and $L(\mathbf{E}_s)$ is the connectivity error cost for the subgraph. $L(\mathbf{E}_s)$ is equal to $L(\mathbf{E}_s^+) + L(\mathbf{E}_s^-)$; $L(\mathbf{E}_s^+)$ and $L(\mathbf{E}_s^-)$ (in Equation (2)) are the error encoding costs for $\mathbf{E}_s^+$ and $\mathbf{E}_s^-$, respectively, which are edges being over-modeled and under-modeled in the given subgraph G', respectively. VoG computes the local encoding costs $L_t(s) + L(\mathbf{E}_s)$ of all the structures for the given subgraph, and then adds the structure s that has the lowest encoding cost to the set C of structures.

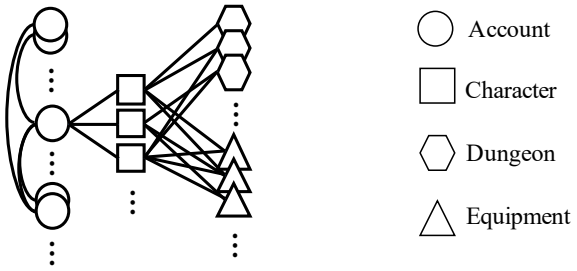


Fig. 3. Graph structure of an MMORPG Blade & Soul. An account has several characters, and each character has equipments and visited dungeons. Accounts are connected to each other via friendship.

VoG constructs the final model which summarizes the given homogeneous graph by carefully selecting structures among the candidate structures in C.

3 PROPOSED METHOD

In this section, we first describe the problem definition in Section 3.1. Then, we give an overview of our method in Section 3.2, our proposed MDL formulation for hierarchical graph summarization in Section 3.3, and our proposed graph summarization algorithm GSHL in Section 3.4.

3.1 Problem Formulation

Our goal is to summarize a heterogeneous graph which has hierarchical label information for each node. Specifically, we target an MMORPG graph which has labels such as account, character, dungeon, and equipment, as shown in Fig. 3. Furthermore, each label has a hierarchy; e.g., the character label contains four sub-labels (e.g., Tanker and Buffer) based on play-styles. Since nodes with hierarchical labels are not handled in previous graph summarization problem (Problem 1), we formulate the following new problem.

Problem 2 (Minimum Hierarchical Graph Description).
Given a heterogeneous graph G with hierarchical labels on its nodes, its adjacency matrix $\mathbf{A}$, and set of structure types Ω, find model M that minimizes the total encoding length

$$L(G, M) = L(M) + L(\mathbf{E}) + L(\mathbf{E}^a) \tag{3}$$

where $\mathbf{E} = \mathbf{M} \oplus \mathbf{A}$ is the derived error matrix, $\mathbf{M}$ is the approximate adjacency matrix of $\mathbf{A}$ induced by M, and $\mathbf{E}^a$ is the labeling error.

The main difference from Problem 1 is that we need to consider hierarchical labels in $L(M)$ and add the labeling error term $L(\mathbf{E}^a)$. A target model $M \in \mathcal{M}$ in Problem 2 is an ordered list of graph structures each having a type from Ω [11]. Each structure $s \in M$ corresponds to a portion of the adjacency matrix; there may be overlapped nodes between different structures, but there is no edge overlap. Each node has hierarchical labels (e.g., Fig. 4 and Table 2).

Following the MDL principle, our method for transmitting the adjacency matrix $\mathbf{A}$ is the following. We 1) transmit model M, 2) induce the approximate adjacency matrix $\mathbf{M}$ of $\mathbf{A}$ by filling out the connectivity of each structure in M, 3) transmit the error $\mathbf{E}$ which is derived from taking the exclusive OR between $\mathbf{M}$ and $\mathbf{A}$ as MDL requires lossless encoding, and 4) transmit labeling error $\mathbf{E}^a$ which is the label information for nodes not included in model M.

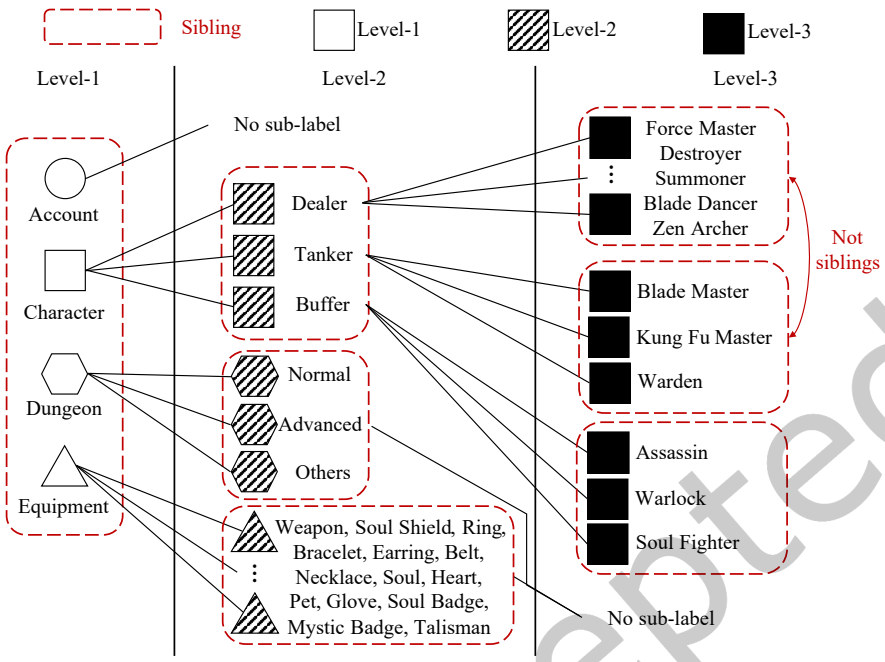


Fig. 4. Hierarchy of labels in a heterogeneous MMORPG graph. There are three levels of hierarchy, and a label may or may not have sub-labels. Two labels are siblings when they share the same parent.

3.2 Overview

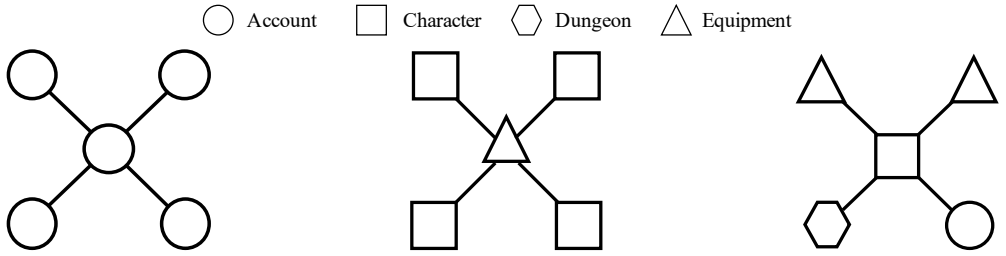
To concisely summarize an MMORPG graph with hierarchical node labels, we need to address the following challenges. The first challenge is how to encode hierarchical labels of nodes. When encoding a subgraph as a structure, we need to formulate encoding costs of labels and consider different levels of hierarchy, in addition to the connectivity. The second challenge is how to construct structures with hierarchical labels for a succinct summary, which includes the structures that have low encoding costs in terms of both connectivity and hierarchical labels.

To address the challenges, we propose the following two main ideas. The first idea is to incorporate label consistency at each level to formulate label encoding costs of each structure (Section 3.3). A label-consistent structure has many nodes with the same label; identifying and encoding such structure give us a low label encoding cost. The second idea is to segment structures at each level of the hierarchy by exploiting the trade-off of the encoding costs in connectivity and hierarchical labels (Section 3.4). Segmenting labels of an inconsistent structure leads to higher connectivity cost but lower label cost. We carefully determine whether to segment or not by comparing encoding costs before and after segmentation.

In the following sections, we present how to design an optimization objective for the summary, and how to encode hierarchical labels. Then, we describe how to generate a summary of the graph based on the formulation.

3.3 Encoding Hierarchical Labels

Our objective is to define encoding costs involved with hierarchical labels in Equation (3). We define the encoding cost $L(M)$ of the overall model (Section 3.3.1), the label encoding cost (Section 3.3.2) inside $L(M)$, and the encoding cost $L(\mathbf{E}^a)$ of labeling errors (Section 3.3.3).



(a) *Consistent* star s_1 (b) *Role-consistent* star s_2 (c) Inconsistentstar s_3

Fig. 5. Examples of label consistency at level-1 in GSHL.

3.3.1 Model Encoding Cost. To find a model that minimizes the total encoding length, we define the model encoding cost $L(M)$ for hierarchical labels as follows.

$$L(M) \;=\; L_{\mathbb{N}}(|M|+1) + \log \binom{|M|+|\Omega|-1}{|\Omega|-1} + \sum_{s\in M} (-\log(P(x(s)|M)) + L_t(s) + L_a(s)) \tag{4}$$

Note that we add $\sum_{s\in M} L_a(s)$ term which encodes the hierarchical labels of nodes, on top of Equation (1). The detailed encodings for hierarchical labels are described in the following section.

3.3.2 Hierarchical Label Encoding. We describe how to encode the hierarchical labels for the nodes in a structure s and derive its cost $L_a(s)$. Each node in our target graph has hierarchical labels as shown in Fig. 4: e.g., a character node may have three-level hierarchical labels "character-tanker-blade master". Furthermore, different nodes may have different depths of their labels: e.g., a dungeon node may have only two-level of hierarchy, like "dungeon-advanced", unlike the character node with three-level of hierarchy. When transmitting a graph to the recipient, we assume that the recipient also knows how the hierarchical labels are structured. In the following, we first propose a base label encoding cost applicable to a general structure, where the labels of all nodes in a structure are individually encoded. Then, we propose a label encoding cost by considering label consistency in a structure, which leads to smaller encoding cost than the base label encoding cost, since several nodes with the same label are grouped and encoded together. We determine which encoding cost to use based on label consistency. After that, we describe a full label encoding cost $L_a(s)$ for hierarchical labels.

Base encoding. In level-k, the base label encoding cost of a structure s is given by

$$L_a^{base}(s,k) = \sum_{m=1}^{|s|} \log(l_{k,m}^b + 1)$$

where $|s|$ represents the number of nodes in the structure s. $l_{k,m}^b$ is the number of unique *sibling labels* for the level-k label of the m-th node of the structure, where *sibling labels* of a label are those with the same upper-level label. For example, when the labels of the m-th node are *character-dealer-destroyer* (see Fig. 4), $l_{1,m}^b$, $l_{2,m}^b$, and $l_{3,m}^b$ are 3, 2 and 4, respectively.

Label consistency. We define the label encoding costs for the following two special cases: 1) *consistent* structure, and 2) *role-consistent* structure. The two special cases allow us to reduce the label encoding cost. A *consistent* structure is the one containing only nodes with the same label, as shown in Fig. 5(a). A *role-consistent* structure is the one where nodes with the same *role* have the same label, as shown in Fig. 5(b). We define 4 roles

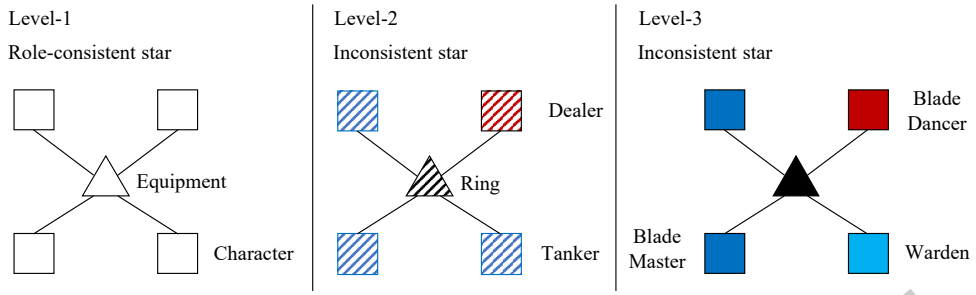


Fig. 6. A star structure of a subgraph in an MMORPG. At level-1, the structure is role-consistent since all the spokes have the same label. In contrast, the structure is inconsistent at levels 2 and 3.

in this work: 1) hub in a star structure, 2) spoke (neighbor of a star) in a star structure, 3) node belonging to the first set in a bipartite core, and 4) node belonging to the second set in a bipartite core. An inconsistent structure is the one that is neither consistent nor role-consistent, as shown in Fig. 5(c).

The label encoding cost for a *consistent* structure in level-k is given by

$$L_a^c(s,k) = \log(l_k^c + 1)$$

where l_k^c of a consistent structure is the number of unique *sibling labels* for the level-k label. For a consistent structure, substituting $L_a^c(s,k)$ for $L_a^{base}(s,k)$ reduces the label encoding cost.

Next, the label encoding cost for a *role-consistent* structure in level-k is given by

$$L_a^{rc}(s,k) = \log(l_{k,1}^{rc} + 1) + \log(l_{k,2}^{rc} + 1)$$

where $l_{k,1}^{rc}$ and $l_{k,2}^{rc}$ of a *role-consistent* structure are the number of unique *sibling labels* for the level-k labels of the two roles that a role-consistent structure can have, respectively. For a role-consistent structure, substituting $L_a^{rc}(s,k)$ for $L_a(s,k)$ reduces the label encoding cost.

We give an example of calculating the label encoding costs of the three level-1 structures described in Fig. 5. The cost of the consistent star structure s_1 in Fig. 5(a) is $L_a^c(s_1,1) = \log(3+1) = \log 4$, since there are three sibling labels (Character, Dungeon, and Equipment) of the label Account. The costs of the role-consistent star s_2 and the inconsistent star s_3 in Fig. 5(b) and 5(c) are $L_a^{rc}(s_2,1) = \log(3+1) + \log(3+1) = 2\log 4$, respectively. A structure with label consistency has a low label encoding cost as shown in the example: $(L_a^c(s_1,1) < L_a^{rc}(s_2,1) < L_a^{base}(s_3,1)) = 5\log 4$. Specifically, $L_a^c(s_1,1)$ and $L_a^{rc}(s_2,1)$ are much lower than $L_a^{base}(s_3,1)$.

Hierarchical label. We define the full encoding cost $L_a(s)$ for hierarchical labels as follows:

$$L_a(s) = \log\binom{|s| + l_1 - 1}{l_1 - 1} + \sum_{i=1}^{h_1} L_a^c(s,i) + \sum_{j=h_1+1}^{h_2} L_a^{rc}(s,j) + \left(\sum_{k=h_2+1}^{h} L_a^{base}(s,k)\right) + 2\log(h)$$

where h_1 and h_2 mark the level in which consistent and role-consistent structures end, respectively; h_1 and h_2 are set to 0 when consistent and role-consistent cases do not exist. l_1 is the number of unique labels in level-1 and h is the number of levels. We encode 1) the number of each label at level-1 by using an index over a weak number composition, 2) *consistent* cases' labels from level-1 to level-h_1, 3) *role-consistent* cases' labels from level-(h_1+1) to level-h_2, 4) the remaining levels using the base encoding for inconsistent cases, and 5) h_1 and h_2. We consider the cases in the order of *consistent*, *role-consistent*, and *inconsistent* cases when moving from higher (e.g. level-1) to lower (e.g. level-3) levels.

Algorithm 1: Graph Summarization with Hierarchical Labels

Input: A heterogeneous graph G with hierarchical node labels
Output: Model M

1: **Subgraph generation.** Given the graph G, generate subgraphs using a graph decomposition method.
2: **Subgraph identification.** For each subgraph obtained from step 1, we pick the structure which minimizes the local encoding cost of the subgraph.
3: **Structure segmentation.** For each candidate structure, we segment it if the segmentation reduces the total encoding cost by considering the trade-off between the label encoding cost and the structure encoding cost.
4: **Model summary.** We construct a model M using summary approaches Vanilla, Top-k, and Benefit.

We give an example of calculating the encoding cost $L_a(s)$ of hierarchical labels using the structure s in Fig. 6. In the first level where the structure is role-consistent, $L_a^{rc}(s,1)$ is equal to $\log(l_{k,1}^{rc}+1)+\log(l_{k,2}^{rc}+1) = \log 4 + \log 4$. In the second and the third levels where the structure is inconsistent, $L_a^{base}(s,2)$ and $L_a^{base}(s,3)$ are $\log 14 + 4\log 3$ and $\log 5 + 3\log 3$, respectively. Therefore, the full encoding cost $L_a(s)$ of the structure s for hierarchical labels is $\log \binom{5+4-1}{4-1} + (2\log 4 + \log 14 + 4\log 3 + \log 5 + 3\log 3) + 2\log 3$.

3.3.3 Encoding Labeling Errors. Given the summary M of G, it is vital to encode edges that are under or over-modeled by the model M which we refer to as errors. Specifically, there are two types of errors to consider: 1) connectivity and 2) labeling. The connectivity error $L(\mathbf{E})$ is handled as described in Section 2.2.1. We describe how to handle labeling errors $L(\mathbf{E}^a)$ where nodes that are failed to be included in at least a structure lack label information. Given the set *en* of nodes with labeling errors, our idea is to include all nodes in *en* into one structure and apply base label encoding. Labeling error encoding cost $L(\mathbf{E}^a)$ is as follows:

$$L(\mathbf{E}^a) \;=\; \log \binom{|en| + l_1 - 1}{l_1 - 1} + \sum_{k=1}^{h} \sum_{m=1}^{|en|} \log(l_{k,m}^{b} + 1)$$

where $|en|$ is the number of nodes in *en*, l_1 is the number of unique labels in level-1, and $l_{k,m}^{b}$ is the number of unique *sibling labels* for the level-k label of the m-th node in *en*.

3.4 Summary Generation

We present GSHL (Graph Summarization with Hierarchical Labels), our proposed summarization algorithm for heterogeneous graphs with hierarchical node labels. Algorithm 1 shows the overall process of GSHL.

3.4.1 Subgraph Generation. We first decompose a given large graph into several subgraphs using a graph decomposition algorithm. For the purpose we leverage SlashBurn [10] which has been used for many graph tasks [9, 19]; [11, 18] demonstrate that SlashBurn successfully decomposes real-world graphs for graph summarization. However, we note that using other algorithms like SUBDUE [6] and BIGCLAM [22] is possible.

3.4.2 Subgraph Identification. For each generated subgraph, we find an appropriate structure in Ω to minimize the encoding cost. For each subgraph, we 1) compute the encoding cost for all structure types, 2) compare the encoding costs, and 3) encode the subgraph as the structure type with the minimum encoding cost. We assume that each subgraph corresponds to one structure among the six types (star, clique, near clique, bipartite core, near bipartite core, and chain) as discussed in Section 2. The main difference from the previous work [11] is that we consider encoding cost $L_a(s)$ for hierarchical labels in a subgraph in addition to the connectivity cost $L_t(s)$.

We first obtain the structures from a subgraph as in Section 2.2.2. We then compute and compare their encoding costs. Given a subgraph G', we compute the local cost $L_t(s) + L_a(s) + L(\mathbf{E}_s)$ to encode it as a structure

Algorithm 2: Hierarchical Segmentation

Input: the set $\mathcal{C}$ obtained by subgraph identification in Section 3.4.2
Output: a modified set $\mathcal{C}'$

$\mathcal{C}' \leftarrow \emptyset$
for all structure $s \in \mathcal{C}$ **do**
  $\mathcal{C}' \leftarrow \mathcal{C}' \bigcup$ (Segmentation(s, 1))
end for

Algorithm 3: Segmentation of Structure

Function Segmentation(*structure s, level k*)
  if $k > 3$ **then**
    return $\{s\}$
  if *s is inconsistent at level k* **then**
    obtain a set $\mathcal{S}$ of sub-structures s'_i segmented from s at level k
    if *segmentation reduces the encoding cost* **then**
      return $\bigcup_i$(Segmentation(s'_i, $k+1$))
    else
      return $\{s\}$
    end
  else
    return Segmentation(s, $k+1$)
  end

s, where $\mathbf{E}_s = \mathbf{M}_s \oplus \mathbf{A}_{G'}$ is the derived error matrix, $\mathbf{A}_{G'}$ is the adjacency matrix of the subgraph G', and $\mathbf{M}_s$ is the approximate adjacency matrix of $\mathbf{A}_{G'}$. Since our objective is to compare the local encoding costs of all the structures for the subgraph G', we compute only $L_t(s) + L_a(s)$ from Equation (4). Note that the remaining terms except for $L_t(s)$ and $L_a(s)$ in Equation (4) are the same when we compare the encoding costs of all the structure with respect to the subgraph G'. In addition, the labeling errors are ignored since nodes in a structure are connected, and thus there are no uncovered nodes inside the structure. We compute and add $L_t(s)$ and $L_a(s)$ (e.g., $L_t(st) + L_a(st)$ for st). We also compute $L(\mathbf{E}_s)$ as in Section 2.2.2. We add the best structure s minimizing the local encoding cost $L_t(s) + L_a(s) + L(\mathbf{E}_s)$ to the set C of structures.

3.4.3 Hierarchical Segmentation. The goal of structure segmentation is to further reduce the label encoding cost for a structure considering the consistency and the hierarchy of labels. Segmenting an inconsistent structure s into substructures (e.g., s'_1 and s'_2) increases the encoding cost of connectivity while decreasing the label encoding cost; i.e., $L_t(s) < L_t(s'_1) + L_t(s'_2)$ and $L_a(s) > L_a(s'_1) + L_a(s'_2)$. Therefore, we need to select an option leading to the minimum cost by comparing the total encoding costs before and after segmentation. We perform the segmentation only when it reduces the total encoding cost. To achieve the goal, we propose a hierarchical segmentation algorithm (Algorithm 2) that segments each structure s of $\mathcal{C}$ into sub-structures by investigating from higher to lower levels. Since the consistency varies at each level as shown in Fig. 6, we need to consider segmentation for all levels. If a structure s is inconsistent at level k, we investigate whether we segment it or not at the level. Otherwise, we investigate the consistency of the structure and segmentation at the next lower level $k+1$. Substructures segmented from s require lower encoding costs than encoding the original structure s.

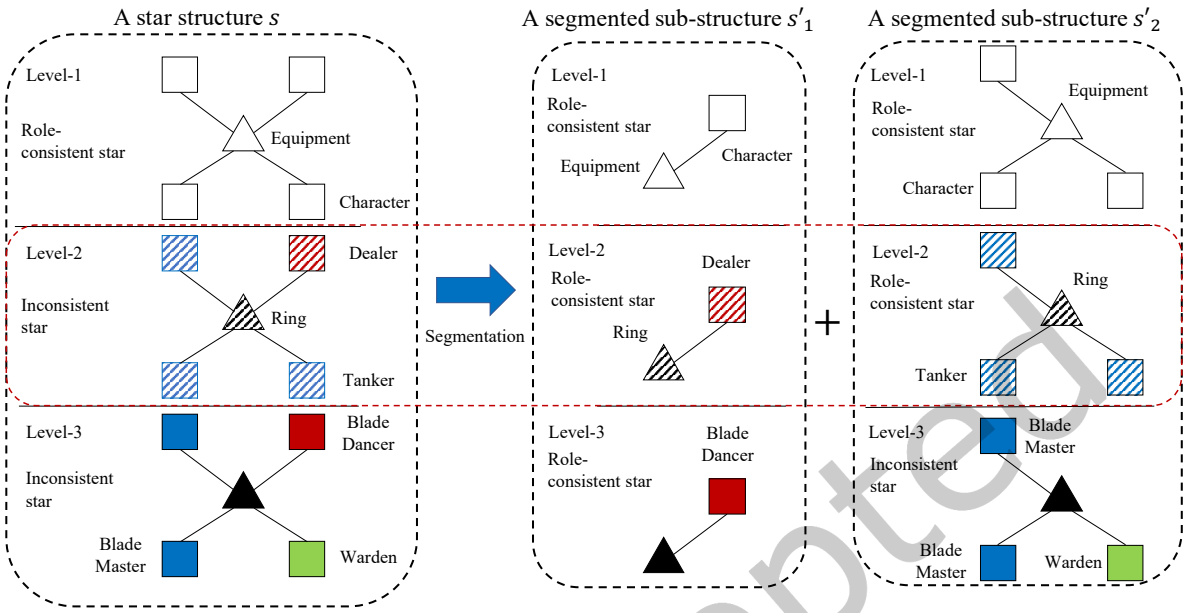


Fig. 7. [Best viewed in color] An example of hierarchical segmentation at level 2. Given a star structure s, we segment it into two sub-structures s'_1 and s'_2 based on the consistency at level 2.

Algorithm 2 describes how to segment structures of the set $\mathcal{C}$ into sub-structures of a modified set $\mathcal{C}'$. We investigate segmentation for each structure s, and insert sub-structures of s into $\mathcal{C}'$ when the segmentation reduces costs; otherwise, we insert the structure s into $\mathcal{C}'$. Before segmenting a structure s in $\mathcal{C}$, we initialize the output set $\mathcal{C}' \leftarrow \emptyset$ (line 1 in Algorithm 2). Then, we investigate all the structures in the set $\mathcal{C}$ obtained by subgraph identification in Section 3.4.2 (in lines 2 to 4 in Algorithm 2). We segment each structure s from level-1 to level-3 using a recursive function for segmentation in Algorithm 3. For each structure s, we investigate segmentation at level $k+1$ if s is consistent or role-consistent at level k (line 15 shown in Algorithm 3). When s is inconsistent at level k, we segment it into sub-structures s_i if the total encoding costs of the sub-structures are lower than those of it: $L_a(s) - \sum_i L_a(s'_i) > \left(\sum_i L_t(s'_i)\right) - L_t(s)$. Then, the segmented sub-structures are independently investigated at level $k+1$ (lines 4 to 8 in Algorithm 3). If s is not segmented or after investigation at level 3, we stop segmentation for s (lines 3 and 11 in Algorithm 3).

We give an example of the segmentation for Algorithm 3. In Fig. 7, it is unnecessary to segment a structure s at level 1 because it is already role-consistent at that level. However, the structure is investigated for segmentation at level 2 due to its inconsistency at the level. Suppose that a structure s can be segmented into two role-consistent sub-structures s'_1 and s'_2 for the consistency at level 2. It means $L_a(s'_1) + L_a(s'_2)$ is lower than $L_a(s)$ since $L_a^{rc}(s'_1, 2) + L_a^{rc}(s'_2, 2) < L_a^{base}(s, 2)$ where $L_a^{rc}(s'_1, 2)$, $L_a^{rc}(s'_2, 2)$, and $L_a^{base}(s, 2)$ are included in $L_a(s'_1)$, $L_a(s'_2)$, and $L_a(s)$, respectively. If $L_a(s) - \left(L_a(s'_1) + L_a(s'_2)\right)$ is higher than $\left(L_t(s'_1) + L_t(s'_2)\right) - L_t(s)$, we perform segmentation. At level 3, we need to investigate the sub-structure s'_2 due to the inconsistency, and the role-consistent sub-structure s'_1 is not investigated for segmentation. Consequently, sub-structures segmented from s require lower total encoding costs than s.

Segmentation at level k. The remaining issue in the hierarchical segmentation is to segment a structure at level k (line 6 in Algorithm 3). Structure segmentation at level k considers only labels at level k, reducing

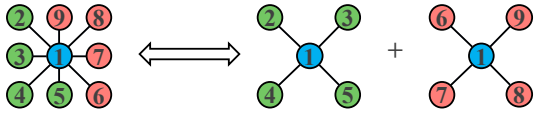


Fig. 8. GSHL chooses small segmented star structures in the right rather than a large inconsistent star in the left since the right ones minimize the encoding cost.

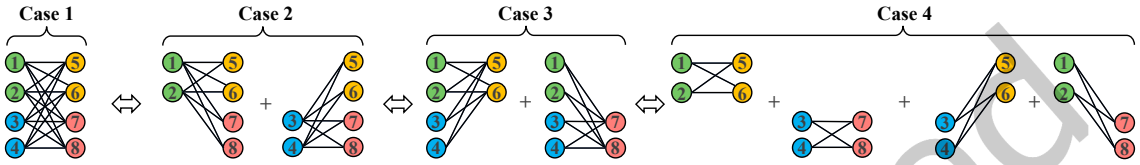


Fig. 9. GSHL chooses the best structure with the minimum encoding cost among the 4 possible cases of segmentation in a bipartite core.

$L_a(s,k)$ of a structure s which is inconsistent at that level; $\sum_i L_a(s'_i,k)$ of sub-structures s'_i segmented from s is lower than $L_a(s,k)$ since consistent (or role-consistent) substructures exist among them. We decompose a structure depending on its type, obtaining a set $\mathcal{S}$ of sub-structures segmented from it. Note that the segmented sub-structures have the same type as the original structure. We describe how we segment a structure at level k for each type.

- **Stars.** As shown in Fig. 8, we divide spokes of a star into two groups considering the consistency of labels at level k, and construct two stars which have the same hub, but different spokes. We assign spoke nodes of the majority label at level k to the first group and the remaining spoke nodes to the second group. For example, in the left structure of Fig. 7, the majority label at level 2 is *Tanker* and the label of the remaining nodes is *Dealer*. The first sub-structure with the majority label is consistent or role-consistent.
- **Bipartite cores and near bipartite cores.** Bipartite cores and near bipartite cores have two sets of nodes. For each set, we search for the majority label at level k and divide the nodes of the set into the nodes with the majority label and the rest. As shown in Fig. 9, we consider 4 cases depending on the segmentation for the two sets: 1) no segmentation, 2) segmenting only the first set, 3) segmenting only the second set, and 4) segmenting both sets. We choose the option with the minimum encoding cost among the 4 possible cases.
- **Cliques, near cliques, and chains.** Contrary to star and bipartite core structures, all the nodes in cliques, near cliques, and chains have the same role. We first find the majority label at level k, and then divide the structure into nodes of the majority label and the rest. The first sub-structure with the majority label is consistent or role-consistent.

Fig. 7 shows an example of structure segmentation in level 2. The leftmost structure s is inconsistent at level 2 since its spokes have *Tanker* and *Dealer* labels. Segmentation generates two role-consistent sub-structures s'_1 and s'_2 at level 2, reducing $L_a^{base}(s,2)$ to $L_a^{rc}(s'_1,2) + L_a^{rc}(s'_2,2)$.

3.4.4 Model Summary. We construct the final model which summarizes the given heterogeneous graph by carefully selecting structures among the candidate structures C. Finding the best model exactly by exploring all possible subsets of candidate structures is intractable. Thus we propose the following heuristics to choose a subset of candidate structures in a scalable way.

- Vanilla: All candidate structures are included in our summary, thus M is equals to the set C of the candidate structures.
- Top-k: We pick the top-k structures by sorting the local encoding gains of candidate structures in descending order. A local encoding gain is the number of bits reduced by encoding a subgraph as a structure x.
- Benefit: We pick all structures whose local encoding gains are greater than zero.

4 EXPERIMENT

We evaluate our proposed method GSHL to answer the following questions:

Q1 Graph summarization. How well does GSHL summarize an MMORPG graph?
Q2 Discovery. What are the discovery results of analyzing an MMORPG graph with GSHL?
Q3 Finding similar users. How can we find similar users using the summary from GSHL?
Q4 Scalability. How well does GSHL scale up in terms of the number of edges?

We run experiments on a workstation with an Intel Xeon E5-2630 v4 @ 2.2GHz CPU and 512GB memory. We describe our target data in detail and then answer the questions.

4.1 Data Description

Our target data is from Blade & Soul (https://www.bladeandsoul.com/en/), a popular MMORPG played in more than 60 countries worldwide and earning revenue worth USD 75 million in 2019.

4.1.1 Blade & Soul. Blade & Soul entails multiple characters simultaneously interacting with each other in an open-world environment. Each character enters various dungeons with other characters in order to acquire new equipment or items.

A user associated with an account can possess multiple in-game characters with different jobs. These jobs are divided into three types of play-styles: dealer, tanker, and buffer. A dealer mainly afflicts damage to the enemies of its party (a group of multiple characters), while a tanker taunts enemies so the party members can freely afflict damage, and a buffer aids other party members. A party needs to have balanced types of jobs to successfully enter a dungeon and fight with enemies.

4.1.2 Dataset. We collect 3 months of data from January to March in 2019, and extract the following four relationships.

- *account - character*: which character(s) each account possesses.
- *account - account*: friendship between accounts.
- *character - dungeon*: which dungeons each character visited.
- *character - equipment*: which equipment each character is equipped with.

We extract 4 types of entities: account, character, dungeon and equipment. An account represents a user, and a user can have multiple characters. A dungeon entity belongs to a type based on its difficulty where an advanced dungeon is a more difficult version of the corresponding normal dungeon.

Hierarchical Labels. Each label is further divided into sub-labels, and this leads to a hierarchical label structure summarized in Table 2. The four types of entities (account, character, dungeon, and equipment) extracted from the dataset represent the level-1 of the hierarchy. The account type has no sub-labels. The character type has level-2 sub-labels based on *play-styles*. Each of the level-2 label of the character has level-3 sub-labels. The dungeon type has 3 level-2 sub-labels. The equipment type has 14 level-2 sub-labels.

Graph. An entity, which is either account, character, dungeon, or equipment, becomes a node in the graph. A relationship between two entities forms an edge. There are four types of relationships: 1) (account - account), 2) (account - character), 3) (character - dungeon), and 4) (character - equipment) as shown in Fig. 3. Table 3

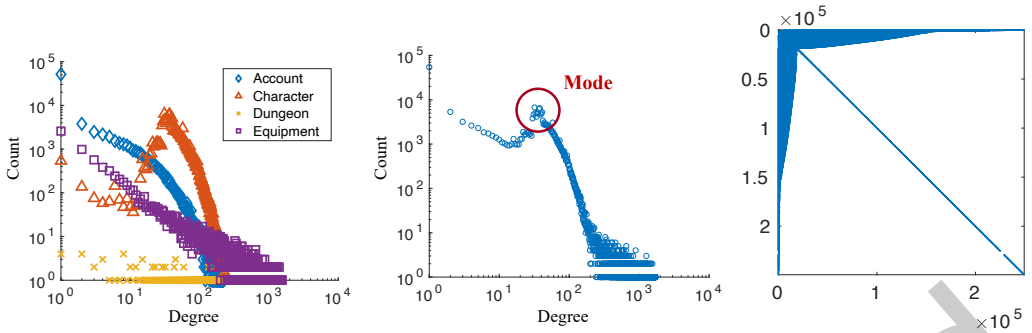
Table 2. Hierarchical labels in Blade & Soul graph.

Level-1	Level-2	Level-3	Node Count
Account	×	×	83, 970
Character	Dealer	Force Master	19, 147
		Destroyer	23,327
		Summoner	6,266
		Blade Dancer	5,822
		Zen Archer	11,023
	Tanker	Blade Master	11,854
		Kung Fu Master	17,845
		Warden	6,689
	Buffer	Assassin	18,460
		Warlock	15,868
		Soul Fighter	18,249
Dungeon	Normal	×	154
	Advanced	×	12
	Others	×	133
Equipment	Weapon	×	3,219
	Soul Shield	×	3,400
	Ring	×	431
	Bracelet	×	398
	Earring	×	455
	Belt	×	95
	Necklace	×	430
	Soul	×	171
	Heart	×	47
	Pet	×	269
	Glove	×	26
	Soul Badge	×	927
	Mystic Badge	×	739
	Talisman	×	29
Total			249, 455

Table 3. Number of edges between level-1 labels in Blade & Soul graph.

	Account	Character	Dungeon	Equipment
Account	229,338	154,550	0	0
Character	154,550	0	2,680,520	4,821,079
Dungeon	0	2,680,520	0	0
Equipment	0	4,821,079	0	0
Total				**7,885,487**

represents the number of edges between level-1 labels. The formed graph contains 249, 455 nodes and 7, 885, 487 edges.



(a) Degree distribution of nodes for each level-1 label type (b) Degree distribution of all nodes (c) Result of SlashBurn

Fig. 10. Blade & Soul graph shows a skewed degree distribution.

Table 4. Comparison of structures and costs from GSHL and the original graph. GSHL-Vanilla explains 99% of 7,885,487 edges using 25,740 structures. GSHL-Top-100 uses only 100 structures to explain 56% of edges. GSHL-Benefit uses 4,382 structures to explain 95% of edges.

Method	# of structures	Description cost (bits)	Relative cost	Unexplained edges
Original	–	106, 444, 727	100%	0%
GSHL-Vanilla	25, 740	52, 124, 308	49%	1%
GSHL-Top-100	100	75, 781, 149	71%	56%
GSHL-Benefit	4, 382	52, 237, 349	49%	5%

Fig. 10 shows the characteristics of the graph. The degree distributions of nodes for each level-1 label type are shown in Fig. 10(a). Note that the degree distributions of all node types except *character* follow power-law. The degree distribution of *character* type nodes has a mode around the degrees $30 \sim 40$ due to the following reasons: 1) the number of items a character can have is limited, and 2) the number of dungeons a character can visit is limited. Fig. 10(b) shows the degree distribution of all nodes. Even though there is a mode around the degrees $30 \sim 40$, the graph shows a skewed degree distribution which is also verified in Fig. 10(c) showing the reordered adjacency matrix from SlashBurn [10] method. Note that a reordered adjacency matrix with a thin arrow shape like Fig. 10(c) means there are few high degree nodes in the graph which can be used for decomposing (or 'shattering') it [10].

4.2 Graph Summarization (Q1)

We evaluate the description cost and the edge coverage of Blade & Soul graph by GSHL. Our baseline is Original model where the original edges are encoded with $L(\mathbf{E}^-)$ and the labels are encoded with $L(\mathbf{E}^a)$.

Table 4 compares the number of structures, the description cost, and the ratio of unexplained edges by Original and the three proposed models (GSHL-Vanilla, GSHL-Top-100, and GSHL-Benefit). GSHL-Vanilla summarizes the given graph with only 49% of the bits w.r.t. Original and explains all but 1% of the edges which are not modeled. GSHL-Top-100 explains 44% of the edges and summarizes the given graph with 71% of the bits w.r.t.

Table 5. Summarization results of Blade & Soul graph by GSHL. *st*, *fc*, *nc*, *bc*, *nb*, and *ch* indicate star, full clique, near clique, bipartite core, near bipartite core, and chain, respectively.

(a) GSHL-Vanilla

st	fc	nc	bc	nb	ch
22,787	21	–	–	2,467	465

(b) GSHL-Top-100

st	fc	nc	bc	nb	ch
100	–	–	–	–	–

(c) GSHL-Benefit

st	fc	nc	bc	nb	ch
3,917	–	–	–	–	465

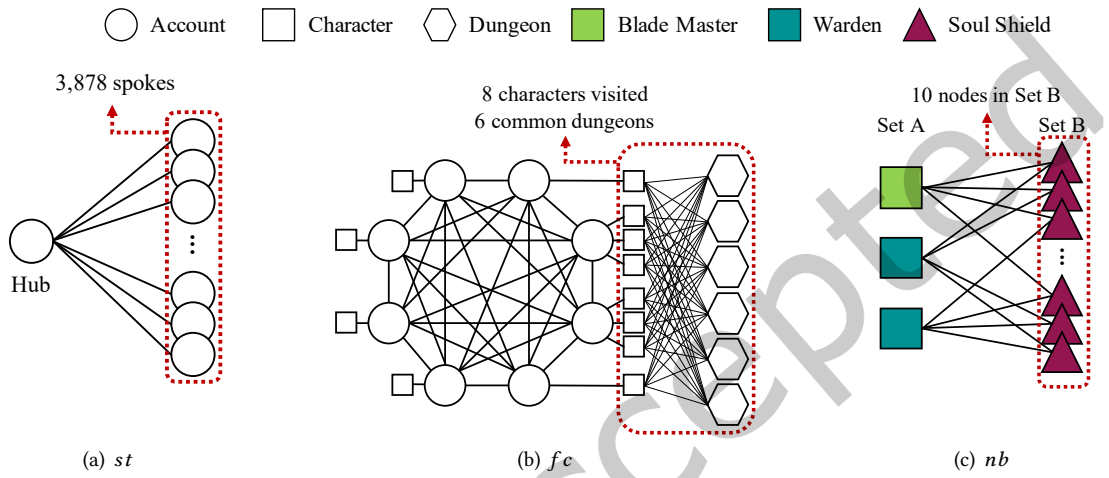


Fig. 11. Meaningful structures from GSHL. (a) *st* (star) structure consisting only of accounts, where the hub node represents an influential *account*. (b) *fc* (full clique) structure consisting of 8 *account* nodes constituting a small clan. The 4 *accounts* on the right possess characters which visited 6 common *dungeons*. (c) *nb* (near bipartite core) structure where a *blade master* and two *wardens* in set A (which are tankers) are equipped with several *soul shields* from set B, representing similar preferences for the three *characters* and similar properties for the 10 *soul shields*.

Original, while containing only 100 structures out of 25,740 structures. Each structure in GSHL-Top-100 contains a larger number of nodes and edges compared to those of the structures not included. Compared to GSHL-Vanilla, GSHL-Top-100 has fewer number of structures but a larger cost. GSHL-Benefit shows a reasonable balance between the description cost and the number of structures. Compared to GSHL-Vanilla, it has a similar description cost, but contains 5.9× smaller number of structures.

4.3 Discovery (Q2)

We present the discovery results of Blade & Soul graph using GSHL.

4.3.1 Major Structures. How does GSHL summarize the Blade & Soul graph, and which structures can we discover? Table 5 shows the structures from our proposed methods GSHL-Vanilla, GSHL-Top-100, and GSHL-Benefit. Note that GSHL-Vanilla contains the most diverse structures including star, full cliques, near bipartite cores, and chains. GSHL-Top-100 contains only stars as they are the dominant structures when using SlashBurn for graph decomposition. GSHL-Benefit contains mainly stars, but it also contains several chains; it means that there are clear chain-like structures in the decomposed subgraphs.

4.3.2 Meaningful Structures. What interpretations can we make out of the summary? We report meaningful structures of Blade & Soul graph discovered by GSHL.

Popular dungeons and equipment. We observe that the labels of hub nodes in star structures found by GSHL-Top-100 mainly consist of popular dungeon and equipment nodes, surrounded only by character nodes. This is due to the nature of Blade & Soul where characters cooperate with other characters visiting various dungeons, while having various equipment.

Influential account. Fig. 11(a) shows a star structure consisting only of account nodes. The hub node is an influential *account* connected to 3,878 other accounts.

Small clan. Fig. 11(b) shows 8 account nodes forming a full clique structure. The 4 accounts on the right possess characters which visited 6 common dungeons. This represents a small clan where the accounts play together frequently.

Near bipartite core of characters and equipment. Fig. 11(c) shows a near bipartite core. The set A in the left contains characters which share similar equipment contained in the set B in the right.

Segmented star. We analyze the structures segmented by considering hierarchical labels. Fig. 1 shows that a large star structure whose hub is an equipment node with soul label is divided into three smaller star structures. The spokes of each star structure consist of: a) zen archers, b) dealers except for zen archers, and c) all jobs in tanker and buffer. This segmentation occurs since 37% (760/2041) of the spokes in the star structure are zen archers (out of the 11 total jobs). Although any jobs can have this soul equipment, zen archers prefer this soul equipment the most.

4.4 Finding Similar Users (Q3)

How can we exploit the result of summarization for finding similar accounts? We measure the similarities of accounts as follows:

(1) **Matrix construction.** Given the found structures, we first construct a *node-structure* matrix **B**. The (i, s)-th element of the matrix **B** is set to $\gamma^{\alpha_{i,s}}$ where γ is set to 0.7 in this experiment and $\alpha_{i,s}$ is the minimum length of the shortest paths between the account i and the structure s. If node i is in structure s, $\alpha_{i,s}$ is set to 0.
(2) **Randomized SVD for features.** We compute a randomized SVD for the matrix **B** to obtain features of accounts.
(3) **Cosine similarity.** We compute the cosine similarity of the two account feature vectors.

We pick a target account with id 0, and find the two most similar accounts which have ids 3602 and 9. We observe that they share similar characters, equipment, and often friends. Fig. 2(a) shows the relation of accounts 0 and 3602. Note that they have the characters with the same jobs (destroyer and blade master). The corresponding characters have common equipment and visited the same dungeons; e.g., the two destroyers have 21 identical equipment, and visited 12 identical dungeons. Fig. 2(b) shows the relation of accounts 0 and 9. We observe the similar patterns of characters and their related equipment and dungeons. They also share 18 common friends as well.

4.5 Scalability (Q4)

We measure the running times of GSHL varying the number of edges for scalability experiments. We generate 4 synthetic subgraphs using Forest Fire Sampling (FFS) [13], where *(# of nodes, # of edges)* are (34203, 78000), (62339, 246480), (109764, 780000), (162759, 2464800) and (249455, 7885487). Note that the subgraphs generated from FFS follow the properties of real world graphs, including heavy-tailed distributions, densification law, and effective shrinking diameters [13]. As shown in Fig. 12, GSHL scales near-linearly with the number of edges.

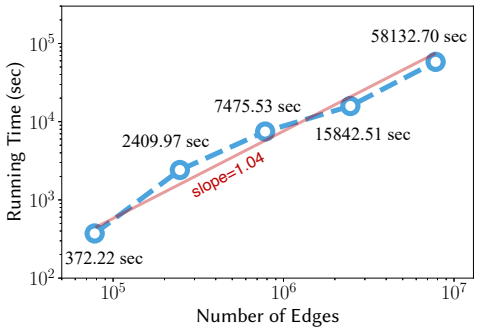


Fig. 12. GSHL provides near-linear scalability with regard to the number of edges.

5 RELATED WORK

We present related works on four main categories: minimal description length, graph compression and summarization, hierarchical graph, and game data mining.

Minimal description length. MDL [16] is a model selection method, where the best model is considered the one that gives the best lossless compression. Its use for compression [8] is related to summarization and pattern discovery. GSHL exploits the MDL principle for summarizing a heterogeneous graph with hierarchical node labels.

Graph compression and summarization. There have been several works including eigendecomposition [17], modularity-based optimization [3], and cross association [5] which find dense structures, such as cliques and bipartite cores. SUBDUE [6] discovers the best substructure that compresses a graph with node labels based on the MDL principle. Slashburn [10] reorders nodes of a graph to efficiently compress the graph by exploiting the characteristics of real-world graphs. Cebiric et al. [4] summarizes RDF graphs and Al-Dhelaan et al. [1] performs graph summarization for hashtag recommendation on social graphs. VoG [11] leverages the MDL principle to summarize a homogeneous graph with common structures. Shah et al. [18] propose TimeCrunch that extends VoG method for dynamic graphs. None of the above methods target a heterogeneous graph with hierarchical labels, which GSHL focuses on.

Hierarchical graph. Hierarchical graphs have been studied for node embedding and classification. HGP-SL [26] and DiffPool [24] improve the performance of predicting unknown graph labels using hierarchical representations of graphs. StructureNet [14] proposes a generator using hierarchical graph, which produces diverse and realistic mesh geometries. Zhang et al. [25] use hierarchical information of labels to improve multi-label classification performance. Wendt et al. [21] propose a hierarchical label propagation algorithm for document classification. Unlike GSHL, none of the above methods address the problem of hierarchical graph summarization.

Game data mining. There have been several works for analyzing game data. Drachen et al. [7] analyze the behaviors of players in MMORPGs using a clustering approach. Bernardi [2] propose a bot detection algorithm by analyzing the behaviors of players. Yang et al. [23] cluster game players by analyzing their purchase records. Thompson et al. [20] analyze chat messages of players in StarCraft 2 using a lexicon-based approach. We represent a popular MMORPG data as a heterogeneous graph, and analyze the graph for summarization.

6 CONCLUSION

We propose GSHL, a novel graph summarization algorithm for a heterogeneous graph with hierarchical node labels. Based on the MDL principle, GSHL decomposes a heterogeneous graph into subgraphs, identifies each subgraph as a structure, segments each structure by considering hierarchical labels of its nodes, and summarizes

the graph. Through extensive experiments on a large real-world MMORPG graph, we show that GSHL is a useful and scalable tool for graph summarization. Thanks to GSHL, we find key structures and similar users in the MMORPG graph.

ACKNOWLEDGEMENT

This research is supported from NCSOFT Co.

REFERENCES

[1] Mohammed Al-Dhelaan and Hadel Alhawasi. 2015. Graph Summarization for Hashtag Recommendation. In *3rd International Conference on Future Internet of Things and Cloud, FiCloud 2015, Rome, Italy, August 24-26, 2015*. IEEE Computer Society, 698–702.

[2] Mario Luca Bernardi, Marta Cimitile, Fabio Martinelli, and Francesco Mercaldo. 2017. A time series classification approach to game bot detection. In *WIMS*. 6:1–6:11.

[3] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[4] Šejla Čebirić, François Goasdoué, Pawel Guzewicz, and Ioana Manolescu. 2018. Compact Summaries of Rich Heterogeneous Graphs. (2018).

[5] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S Modha, and Christos Faloutsos. 2004. Fully automatic cross-associations. In *SIGKDD*. 79–88.

[6] Diane J. Cook and Lawrence B. Holder. 1994. Substructure Discovery Using Minimum Description Length and Background Knowledge. *J. Artif. Intell. Res.* 1 (1994), 231–255.

[7] Anders Drachen, Rafet Sifa, Christian Bauckhage, and Christian Thurau. 2012. Guns, swords and data: Clustering of player behavior in computer games in the wild. In *CIG*. 163–170.

[8] Christos Faloutsos and Vasileios Megalooikonomou. 2007. On data mining, compression, and kolmogorov complexity. *Data mining and knowledge discovery* 15, 1 (2007), 3–20.

[9] Jinhong Jung, Namyong Park, Lee Sael, and U. Kang. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *SIGMOD*. 789–804.

[10] U. Kang and Christos Faloutsos. 2011. Beyond 'Caveman Communities': Hubs and Spokes for Graph Compression and Mining. In *ICDM*. 300–309.

[11] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. 2014. VOG: Summarizing and Understanding Large Graphs. In *SDM*. 91–99.

[12] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. 2011. Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms. In *ECML-PKDD*. 245–260.

[13] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *SIGKDD*. 177–187.

[14] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. 2019. Structurenet: Hierarchical graph networks for 3d shape generation. *arXiv preprint arXiv:1908.00575* (2019).

[15] J. Rissanen. 1978. Modeling by shortest data description. *Automatica* 14, 5 (1978), 465 – 471.

[16] Jorma Rissanen. 1983. A universal prior for integers and estimation by minimum description length. *The Annals of statistics* (1983), 416–431.

[17] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. 2014. Spotting suspicious link behavior with fbox: An adversarial perspective. In *ICDM*. 959–964.

[18] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. 2015. TimeCrunch: Interpretable Dynamic Graph Summarization. In *SIGKDD*. 1055–1064.

[19] Kijung Shin, Jinhong Jung, Lee Sael, and U. Kang. 2015. BEAR: Block Elimination Approach for Random Walk with Restart on Large Graphs. In *SIGMOD*. 1571–1585.

[20] Joseph J. Thompson, Betty H. M. Leung, Mark R. Blair, and Maite Taboada. 2017. Sentiment analysis of player chat messaging in the video game StarCraft 2: Extending a lexicon-based model. *Knowl.-Based Syst.* 137 (2017), 149–162.

[21] James Bradley Wendt, Michael Bendersky, Lluis Garcia Pueyo, Vanja Josifovski, Balint Miklos, Ivo Krka, Amitabh Saikia, Jie Yang, Marc-Allen Cartright, and Sujith Ravi. 2016. Hierarchical Label Propagation and Discovery for Machine Generated Email. In *WSDM*. 317–326.

[22] Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM*. 587–596.

[23] Wanshan Yang, Gemeng Yang, Ting Huang, Lijun Chen, and Youjian Eugene Liu. 2018. Whales, Dolphins, or Minnows? Towards the Player Clustering in Free Online Games Based on Purchasing Behavior via Data Mining Technique. In *Big Data*. 4101–4108.
[24] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems 31*. 4800–4810.
[25] L. Zhang, S. K. Shah, and Ioannis A. Kakadiaris. 2017. Hierarchical Multi-label Classification using Fully Associative Ensemble Learning. *Pattern Recognition* 70 (2017), 89–103.
[26] Zhen Zhang, Jiajun Bu, Martin Ester, Jianfeng Zhang, Chengwei Yao, Zhi Yu, and Can Wang. 2019. Hierarchical Graph Pooling with Structure Learning. *arXiv preprint arXiv:1911.05954* (2019).