Correlation Constraint Shortest Path over Large Multi-Relation Graphs

Xiaofei Zhang*
University of Memphis
xiaofei.zhang@memphis.edu

M. Tamer Özsu
University of Waterloo
tamer.ozsu@uwaterloo.ca

ABSTRACT

Multi-relation graphs intuitively capture the heterogeneous correlations among real-world entities by allowing multiple types of relationships to be represented as entity-connecting edges, i.e., two entities could be correlated with more than one type of relationship. This is important in various applications such as social network analysis, ecology, and bio-informatics. Existing studies on these graphs usually consider an edge label constraint perspective, where each edge contains only one label and each edge is considered independently. For example, there are lines of research focusing on reachability between two vertices under a set of edge label constraints, or finding paths whose consecutive edge labels satisfy a user-specified logical expression. This is too restricted in real graphs, and in this work, we define a generic correlation constraint on multi-relation graphs from the perspective of vertex correlations, where a correlation can be defined recursively. Specifically, we formalize and investigate the shortest path problem over large multi-relation graphs in the presence of both necessity and denial constraints, which have various real applications. We show that it is nontrivial to apply conventional graph traversal algorithms (e.g., BFS or DFS) to address the challenge. To effectively reduce the search space, we propose a *Hybrid Relation Encoding* method, a.k.a. HyRE, to encode both topological and relation information in a compact way. We conduct extensive experiments over large real-world graphs to validate the effectiveness and efficiency of the proposed solution.

PVLDB Reference Format:
Xiaofei Zhang, M. Tamer Özsu. Correlation Constraint Shortest Path over Large Multi-Relation Graphs. *PVLDB*, 12(5): 488-501, 2019.
DOI: https://doi.org/10.14778/3303753.3303756

1. INTRODUCTION

Real-world entities are often correlated in multiple ways, either explicitly or implicitly. For example, two persons could be identified as friends when they follow each other on social media, and they could be business rivals working on similar projects, or potential collaborators in the future; STRING [42, 43] models the complex protein-to-protein interaction with six types of correlations statistically learned from existing protein databases, revealing that most protein-to-protein interactions are associated with at least two types of correlations. Other applications where these multiple relationships exist between entities include urban and transportation systems [13], ecology research [41], and recommender systems [31, 40]. The data in these applications are typically modeled as multi-relation graphs capturing the heterogeneous multiple correlations among real-world entities as entity-connecting edges. Compared to other emerging graph models, e.g., the property graph model [4, 3], which allows multiple attributes and values to be associated with vertices and edges, the multi-relation model is simplified to focus on the correlation of vertices. This graph model has been attracting increasing research interest in graph analytics with most work focusing on structural graph characteristics, e.g., reachability between two vertices under given edge label constraints [46], or finding paths that have consecutive edge labels that satisfy a user-specified logical expression [36, 23, 17]. However, the existing work considers only edge labels, which model the pairwise vertex correlation, often ignoring the correlation among a group of vertices. In addition, the existing work only considers positive constraints (what we refer in this paper as *necessity constraints*) that need to hold for query evaluation. However, considering negative constraints (what we call *denial constraints*) are equally important but far more complicated. In this paper, we consider both positive and negative constraints under the unified *correlation constraints*. We model the correlation between vertices recursively, such that the correlation between two vertices that are not immediately connected can be captured. To ground our discussion, we focus on the shortest path computing problem over multi-relation graphs although the method is more generally applicable.

Given a multi-relation graph, finding the shortest path between two vertices that satisfies a set of user-defined constraints has various practical applications. Examples include finding the connection of two individuals on a social network allowing only "*follower*" semantics, or calculating an optimal message routing strategy using all possible connections except those with latency greater than 5msec. All these predicates are necessity constraints imposed on the property or relation type of an edge, which has inspired several lines of research [36, 23, 17]. In this work, we focus on incorporating denial constraints into shortest path computing, which is needed in a number of applications:

Example 1.1. Functioning path in gene regulatory network. A gene regulatory network [6] depicts multiple types of interactions among a collection of molecular regulators and other substances in the cell for gene expression. Gene X regulates gene Y if certain coherence patterns are found between them. One major analysis task over this type of network is to find a functional path such that

*Work done while the author was with University of Waterloo.

Proceedings of the VLDB Endowment, Vol. 12, No. 5
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3303753.3303756

genes on the path are only regulated following the same coherence pattern [26]. Therefore, a denial constraint should be imposed to eliminate all other types of regulatory coherence in the search for functioning paths.

Example 1.2. Conflict-of-interests free search. Being free of conflicts-of-interest (COI) is critical in many applications: e.g., a peer review committee needs to be COI-free to deliver unbiased objective opinions. Committee members need to be experts of the particular field, which is the necessity constraint, and no two of them should be from the same institution or should have co-authored a paper together, which is the denial constraint. Given the size limit of the committee as well as an academic graph that encodes co-authorship, study interests, affiliation, etc., this can be easily transformed into a correlation constraint query. Another practical scenario is where the existence of a certain type of pairwise relationship is considered to be private, such that a COI-free path needs to meet certain privacy-preserving guarantees, e.g., the differential shortest path problem [38].

Example 1.3. Secure message passing. In communication networks, "eavesdropping" and message injection are easy between two peers who have been hacked. Thus, a message should not be considered legitimate if it is routed through peers who are suspected of being corrupted. A straightforward extension of this simple scenario is the k-anonymous shortest path problem [47, 48], where assuming a set of nodes are under control, limiting message passing through no more than k nodes in this set becomes a denial constraint, otherwise anonymity is broken.

One common feature of these examples is that a given type of relation must not exist among the vertices in the result, which is why they are called denial constraints. Handling denial constraints is more complicated than enforcing necessity constraints as the possible search space increases. Intuitively, if the correlation between two vertices matches the denial constraint, these vertices cannot appear simultaneously in the returned result. Thus, existing exploration methods, e.g., *breadth-first-search* (BFS) or *depth-first-search* (DFS), cannot be straightforwardly applied as the search space would change whenever a vertex is filtered out as a result of meeting the denial constraint, and determining the proper order of such changes in order to get the correct result is difficult.

There are other practical factors that make the problem challenging. First, given the size of real-world multi-relation graphs that scale to millions of vertices and billions of edges, it is necessary to incorporate any solution into parallel graph processing approaches such as the vertex-centric [30, 5] or the block-centric [51] computing. Second, many of these graphs are dynamic and experience regular updates. This makes exhaustive indexing techniques for conventional shortest path computing, e.g., tree decomposition [49], or landmarks [2, 21] infeasible or difficult due to index maintenance overhead. Moreover, the skew in the sizes and topological structures among subgraphs induced by different relations pose challenges to partitioning of the graph. It is worth pointing out that we explore the correlation constraint shortest path problem based on the assumption that the multi-relation graph has been constructed *w.r.t* correlation functions that can be user defined or application specific. The construction process of a multi-relation graph is beyond the scope of our discussion.

In this work, we study a general approach to handling correlation constraints (both necessity and denial) in the context of shortest path problem over multi-relation graphs. We propose a solution framework that includes three main components. First, we propose a novel vertex encoding scheme, namely *Hybrid Relation Encoding*, a.k.a. HyRE, which encodes the information representing both hierarchical structures and the skewed sizes of subgraphs induced by different relations. This serves as a lightweight index to effectively reduce the search space. Second, we investigate a non-trivial extension of the conventional BFS algorithm to answer the correlation constraint shortest path query. We prove that finding a shortest path with denial constraints is NP-hard and cannot be approximated within a constant factor; therefore, we further propose a time efficient heuristic algorithm that progressively adds "high value" vertices for evaluation that are promising to form a *relatively short* path. Third, we devise an index structure that can be computed cumulatively, which favors the scenarios where the set of denial constraints is fixed or frequently appears in the query log. Our proposed solution has the following advantages:

- It effectively deals with both necessity and denial constraints for shortest path evaluation on large multi-relation graphs;
- HyRE is a space-saving compact encoding scheme that can effectively reduce the search space and quickly reach an early stop condition when the query result is empty.
- Compared to an exploration-based method such as BFS, our proposed heuristic solution can return satisfactory results more efficiently; and
- Our solution can be straightforwardly adopted in a large scale graph processing framework with off-the-shelf dimensionality reduction techniques for data partitioning.

To validate the approach, we conduct extensive experiments on a collection of real-world multi-relation graphs on a moderately sized cluster. The rest of this paper is organized as follows. After a brief review of related literature in Section 2, we introduce the generic predicate-aware shortest path computing problem in Section 3. We introduce HyRE in Section 4 and explain how it can reduce the search space. In Section 5, we investigate both exact and approximate algorithms for query processing, followed by experiment results discussions in Section 6. We conclude with Section 7.

2. RELATED WORK

Multi-relation Graphs. A multi-relation graph, (also known as multi-dimensional network or multilayer network), has a wide range of applications in scientific computing and social network analysis [13, 41, 14]. However, most existing analyses performed on these graphs target finding specific feature metrics, e.g., eigenvalue spectrum analysis, information flow, or densely connected component. There hasn't been much research on answering selective queries like shortest path on such graphs.

Edge Constraint Shortest Path. Many works exist on shortest-path querying, e.g., [1, 9, 20, 37]. The work most related to ours is a line of research focusing on shortest path query on labeled graphs, called Edge Constraint Shortest Path (ECSP) problem [11, 36, 19, 23], where the edge constraints are user-specified labels or edge property values, which are all necessity constraints. Bonchi et al. [11] present approximate answers for ECSP queries based on landmark vertices, which are reference points. They propose two types of indexes, namely, PowCov and ChromLand that exhibit interesting trade-offs between index size and accuracy. However, neither of these indexes are suitable for applications that require exact shortest-path computations. CHLR [36] can answer ECSP queries exactly and has been extended to support more flexible edge restrictions [19]. CHLR adopts Contraction Hierarchies (CH, for short) [20] to provide an exact answer to the following query: Given a source s, a destination d, and a set of restricted labels R, retrieve the shortest path from s to d that avoids all the edges with labels in R. An ECSP query can be answered by CHLR

by computing the complement of set of labels listed as necessity constraints. EDP [23] is a traversal-based algorithm that supports dynamic graphs. The essential idea is to construct an index over subgraphs that are induced by each type of label to capture the overall topological structure such that cross partition exploration can be effectively supported. Although EDP claims that its solution can be extended to multi-relation graphs, an effective synchronization mechanism is missing when simultaneous traversals are performed in multiple subgraphs (induced by multiple relationships). Without proper synchronization, there would be considerable inefficiency in parallel computation due to significant data shuffling and imbalanced workloads. Variants of finding Regular-Language-Constraint-Paths (RLCP) have also been studied [8, 15, 33]. An RLCP query works on edge-labeled graphs where the concatenation of the labels of the found path satisfies a specified regular expression. While ECSP queries do not impose any order on the labels, RLCP queries assume that the user knows the exact order of the labels. Hence, RLCP queries are orthogonal to ECSP queries. Another line of research focuses on applying conjunctive regular path queries to vertex pairs, namely the CRPQs, which normally output a set of vertex pairs who are connected by the way defined in the query [12, 18]. Extended CRPQ [7] is proposed to improve the expressiveness of CRPQ, such that complex correlations between paths can be defined, and to have paths included in the query results. The fundamental difference between this class of queries and the one we define in this paper is that the semantics of our correlation constraint guarantees that every pair of vertices in the output results satisfies the given constraint, which cannot be guaranteed by CRPQ or its extensions.

Graph indexing. A plethora of index techniques have been proposed to facilitate different types of graph queries. For shortest path queries, a decomposed tree index is proposed [50], which has the advantage of reducing the search space by traversing only tree branches. Distance-aware embedding approaches [29] have proven effective for sparse graphs with large diameters. However, none of these works can be straightforwardly applied when correlation constraints are taken into consideration. As discussed in later sections, we introduce a correlation-aware layer-by-layer abstraction index structure to prune the search space. A recent work [28] also proposes a layered index to speed up graph similarity search under the graph edit distance semantics. Their approach is to devise a multi-layer filter, where each layer is independent and constructed based on a partition of the underlying graph so as to maximize the probability of pruning false positives. The major methodological difference of our approach is that the HyRE index is a tree-like structure, which has correlations among consecutive levels or layers.

Large Scale Graph Processing. A number of scale-out graph processing systems have been developed in the last decade. The most prominent is probably the vertex-centric BSP (*bulk synchronous parallel*) approach popularized by Pregel [30] and its open-source equivalent Giraph [5]. The so-called "think-like-a-vertex" paradigm allows algorithm developers to express their computation in terms of vertex programs that describe the operations to be executed on a single vertex and its incident edges and share vertex states by message passing among neighbor vertices. The BSP computation ensures that parallel computing nodes synchronize at the end of each iteration. Another programming model called "think-like-a-subgraph", as represented by Blogel [51], is a partition-based approach that allows sequential execution within each partition so that only cross-partition message passing is needed. Online graph query and transactional workloads are typically serviced by graph database systems such as Neo4j [45] and JanusGraph [25]. Both systems adopt the native graph storage model (i.e., adjacency list) and evaluate queries with traversals. Global and local indices are built to accelerate the vertex/edge access. Although a number of graph computing algorithms are developed within the context of the above computing frameworks and systems, no correlation constraint shortest path computing method has been investigated.

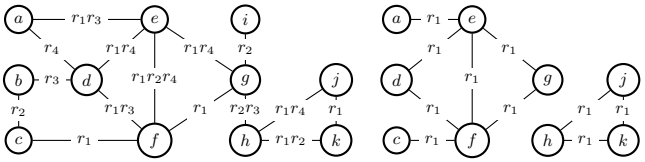


Figure 1: (a) Multi-relation graph G; (b) G_C, where $C = r_1$; singleton vertex b is omitted.

3. PROBLEM DEFINITION

Definition 1 (Multi-Relation Graph)**.** A multi-relation graph G is a four-tuple $G = \langle V, E, \mathbb{R}, F \rangle$, where V is the set of vertices, $E \subseteq V \times V$ is the set of edges, $\mathbb{R} = \{r_1, ..., r_k\}$ is the set of all relations allowed in G, and $F : E \rightarrow R$ ($R \subseteq \mathbb{R}$ and $R \neq \emptyset$) is a mapping function that assigns to each edge a subset of $\mathbb{R}$.

We assume that each vertex is identified by a unique vertex id. According to the definition, each edge represents at least one type of relation between two vertices. G could be a directed graph, in which case the set of relations connecting two vertices is not necessarily symmetric. In other words, $F(\langle s, t \rangle)$ and $F(\langle t, s \rangle)$ can be different. This represents real-world applications, e.g., people do not necessarily follow all their followers on Twitter. As a matter of fact, by differentiating the initiator (outgoing edge) and the acceptance (incoming edge) of a relation, our solution can be straightforwardly extended to cover asymmetric correlations. For simplicity, in the remainder we only consider the case where $F(\langle s, t \rangle) = F(\langle t, s \rangle)$.

Given a multi-relation graph G that describes complex correlations between vertices, an application dependent analysis may only focus on the vertices that participate in a subset of relation types. For example, in social media analysis, the number of likes of a post from users belonging to the same community suggests the influence of this post to a specific group of people. Thus, by extracting such posts and users connections, a subgraph of G can be derived to serve further analysis. Another example is the communication network, where the system load analysis would focus on dimensions like frequency, length, and transmission rate of phone calls, while a marketing strategy is designed based on portraits of users and communities. We define a group of vertices that participate in a set R of relations in G, denoted R-correlated subgraph, as follows:

Definition 2 (R-Correlated Subgraph)**.** Given a multi-relation graph $G = \langle V, E, \mathbb{R}, F \rangle$, a R-correlated subgraph (where $R \subset \mathbb{R}$), denoted as $g_R = \langle V_R, E_R \rangle$, is a maximal induced graph of a set of vertices $V_R \subset V$ such that $\forall V_x, V_y \subset V_R$, where $V_x \cap V_y = \emptyset$ and $V_x \cup V_y = V_R$, $\exists u \in V_x, v \in V_y$ and $R \subset F(\langle u, v \rangle)$.

A R-correlated subgraph is a connected subgraph of G that has relation set R covered on each edge. The maximal semantic indicates that no supergraph of g_R remains a R-correlated subgraph. Note that given R and G, there can be multiple non-overlapping subgraphs that satisfy this definition. For example in Figure 1, let $\{r_1, r_2, r_3, r_4\}$ denote types of relations in graph G, there are two $\{r_1\}$-correlated subgraphs, as shown in Figure 1(b).

Definition 3 (Vertex Correlation)**.** Given a multi-relation graph $G = \langle V, E, \mathbb{R}, F\rangle$, a set of vertices $V' \subset V$ is correlated under a relation set R (where $R \subset \mathbb{R}$) iff $\exists g_R = \langle V_R, E_R\rangle$, such that $V' \subset V_R$.

The above definition indicates that two vertices u and v are correlated *w.r.t* r_i when the following conditions are true: (1) if two vertices are adjacent in G and the edge between them has relation r_i; or (2) there is a path in G between u and v where each edge in the path has relation r_i. For example, in Figure 1, vertices from $\{a, c, d, e, f, g\}$ are correlated under r_1. However, vertices from two disjoint sets of vertices $\{a, c, d, e, f, g\}$ and $\{h, j, k\}$ are not correlated under r_1. Formally, we define the correlation constraint and its satisfying condition as follows:

Definition 4 (Correlation Constraint)**.** A correlation constraint C is either a singleton relation or recursively defined with an operator $o \in \{\wedge, \vee, \neg\}$:

$$C = \begin{cases} r_i \in \mathbb{R} \\ \neg C \\ C_i \wedge C_j \\ C_i \vee C_j \end{cases}$$

We say a set of vertices V' satisfies C, denoted $V' \vDash C$, iff

$$C = \begin{cases} r_i & \text{if } V' \text{ is correlated under } r_i \\ \neg C & \text{if } \nexists u, v \in V', \text{s.t. } \{u, v\} \vDash C \\ C_i \wedge C_j & \text{if } V' \vDash C_i \text{ and } V' \vDash C_j \\ C_i \vee C_j & \text{if } \exists V_x \cup V_y = V', \text{s.t. } V_x \vDash C_i \text{ and } V_y \vDash C_j \end{cases}$$

As noted earlier, a correlation constraint is a predicate that can consist of a *necessity constraint* (positive) and *denial constraint* (negative), represented by C_{N} and C_{D}, respectively, and $C = C_{\mathrm{N}} \wedge C_{\mathrm{D}}$. Following the above definition, C_{N} indicates the types of relations that are desired to connect a group of vertices. In contrast, C_{D} imposes a constraint on the types of relations that must not exist between any two vertices in a group. Note that a correlation constraint C does not necessarily have both C_{D} and C_{N}; either one can be empty, which implies that no such constraint exists. Although Definition 4 allows C to be recursively defined, C is evaluated based on set semantics as specified in the definition. Compared to resolving regular expression constraints in *Regular Path Query*, which typically targets validating a sequence of edge labels with automata- or search-based approach [17, 39], C is tested on a set of vertices.

Definition 5 (C-SAT Subgraph)**.** Given $G = \langle V, E, \mathbb{R}, F\rangle$, a correlation constraint C, a correlation-satisfying subgraph (C-SAT subgraph), denoted $g_C = \langle V_C, E_C\rangle$, is a maximal induced subgraph of G, such that $V_C \vDash C$.

Definition 6 (C-SAT Graph)**.** Given $G = \langle V, E, \mathbb{R}, F\rangle$, a correlation constraint C, a correlation-satisfying graph (C-SAT graph), denoted $G_C = \langle V_C, E_C\rangle$, is a maximal set of C-SAT subgraphs from G, $G_C = \{g_C^i | i = 1, ..., k\}$, $\forall g_C^i, g_C^j \in G_C$, $g_C^i \cap g_C^j = \emptyset$.

Note that the maximal set constraint indicates that no superset of a G_C can be a C-SAT Graph. Given a multi-relation graph G and a necessity predicate C_{N}, by scanning all the edges, it is straightforward to obtain a single C-SAT graph $G_{C_{\mathrm{N}}}$ on a fixed set of vertices, as shown in Figure 1(b). In contrast, when a denial constraint C_{D} is applied, there can be more than one C-SAT graph that meets the constraint even on a fixed set of vertices. The reason is that no two vertices that are correlated under the specified denial constraint can appear simultaneously in $G_{C_{\mathrm{D}}}$. To elaborate, following the same example graph shown in Figure 1(a), when $C = r_1 \wedge \neg r_2$, there are four sets of subgraphs of G which meet the constraint, as shown in Figure 2. Following the above definitions, we formally define the pairwise correlation constraint shortest path problem below.

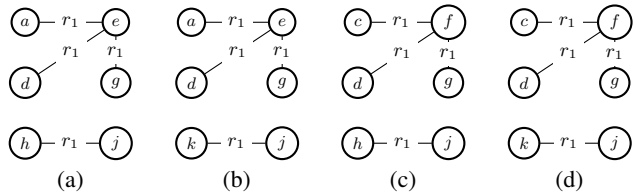


Figure 2: G_C, $C = r_1 \wedge \neg r_2$, singleton vertices omitted

Problem Definition 1 ($s \sim t$ CCSP)**.** Given a multi-relation graph $G = \langle V, E, \mathbb{R}, F\rangle$, a correlation constraint C, and two vertices $s, t \in V$, let p_{st} denote a path from s to t in G; $|p_{st}|$ be the number of hops in p_{st}; $p_{st}.V$ be the set of vertices involved in p_{st}. The problem of correlation constrained shortest path from s to t (denoted $s \sim t$ CCSP) is to find p_{st}^*, such that: (1)$|p_{st}^*|$ is minimized; and (2) $p_{st}^*.V \vDash C$.

Different from conventional shortest path problem, which only asks for minimum number of hops (first condition above), CCSP has an additional requirement on the correlation of returned set of vertices (second condition). Again we elaborate using the example shown in Figure 1(a). A query "*find the shortest path from a to c under constraint r_1*" (note the necessity constraint) can be straightforwardly evaluated using the graph in Figure 1(b). However, if we add a denial constraint $\neg r_2$ to the query, the result would be empty, since vertices e and f become mutually exclusive. Therefore there is no path from a to c that satisfies both the necessity and denial constraints at the same time. Such a definition imposes a restriction on the correlation of any pair of vertices in the returned results, which is desired in many real-world applications. Consider the protein-to-protein network, which depicts the interaction between different protein modules. Some key chemical interactions can be identified to be harmful, which are exactly the denial constraints that need to be met.

However, the above definition can be over restrictive. For example, in some applications, the strength of relation correlation may decay along the an increasing path length, e.g., two people may never know each other even if they are only two hops away on the social network, and the above definition can significantly narrow its application in practice. Therefore, we further investigate the shortest path problem with a relaxed denial constraint. Following the same context presented above, we formally define it below.

Problem Definition 2 ($s \sim t$ CCSP-R)**.** A relaxed $s \sim t$ CCSP shortest path problem (denoted $s \sim t$ CCSP-R) is to find p_{st}^*, such that: (1) $|p_{st}^*|$ is minimized; (2) $p_{st}^*.V \vDash C$; and (3) $\forall u, v \in p_{st}^*.V$, $|p_{uv}^*| < d_{uv}^{C_{\mathrm{D}}}$, where $d_{uv}^{C_{\mathrm{D}}}$ is the shortest path distance between u and v in $G_{C_{\mathrm{D}}}$.

$s \sim t$ CCSP-R relaxes the denial constraint based on the assumption that fewer number of hops implies stronger correlation. For example, two persons who are three-hop friends on a social network but are linked under the same working project, are more correlated as co-workers than as friends. We study the complexity of above problems with the following Lemma.

Lemma 1. The $s \sim t$ CCSP problem can be solved with $O(c|E|)$ edge traversals (i.e., messages), where c is the number of C-SAT subgraphs in $G_{C'}$, and $C' = \neg C_{\mathrm{D}}$.

Proof. Every vertex $u \in G_{C_{\mathrm{D}}}$ that is reached would filter out all other vertices reachable from u in $G_{C_{\mathrm{D}}}$. This can be done by forwarding an early stop signal message after reaching u during the graph traversal to make sure C_{D} is strictly met. As such an event can be triggered at most c times during the exploration, the total number of messages is at most $c|E|$. □

It is worth pointing out that $C_{\mathrm{D}} = \emptyset$ implies $C' = \emptyset$, meaning that no constraint applies in obtaining $G_{C'}$. Thus, there will be only one C-SAT subgraph, which is G itself, indicating $c = 1$.

Lemma 2. If $C_{\mathrm{D}} \neq \emptyset$, the $s \sim t$ CCSP-R problem is NP-hard.

Proof. We prove the Lemma with a reduction from the maximum clique problem. For simplicity, we consider the simplest denial constraint where $C_{\mathrm{D}} = r_d$. Given multi-relation graph G, a $\{r_d\}$-SAT graph $G_{\{r_d\}}$ can be derived based on Definition 6. We construct another graph G' as follows: 1) $\forall u, v \in G$ having $\{u, v\} \vDash C_{\mathrm{N}}$, connect u and v in G'; 2) let $e(u, v)$ denote an edge between u and v, then $\forall e(u, v) \in G_{\{r_d\}}$, if $\exists w \in G'$ such that $e(u, w), e(v, w) \in G'$, then remove $e(u, v)$ from G'; 3) remove all the vertices in G' that are not neighbors of input vertices s and t. The above process can be done in $O(|V|^2)$ time. Then the shortest path between s and t under correlation constraint is contained in the intersection of the maximum clique of G' and the C_{N}-SAT graph $G_{C_{\mathrm{N}}}$. Thus, if $s \sim t$ CCSP-R query can be efficiently answered, the maximum clique of G' can be efficiently computed as it must contain the vertices that reside on the $s \sim t$ path. Therefore, $s \sim t$ CCSP-R is at least as difficult as the maximum clique problem, which is both fixed-parameter intractable and hard to approximate [24]. Unless P=NP, there cannot be any polynomial time algorithm that approximates the problem with a factor better than $O(n^{1-\epsilon})$, for any $\epsilon > 0$. □

In this work, we study both $s \sim t$ CCSP and its relaxation $s \sim t$ CCSP-R. We investigate an extension of conventional graph exploration-based methods such as BFS to return the exact correlation constraint shortest path. In addition, we propose a heuristic algorithm to address the relaxed shortest path problem, since it remains an open question if a constant factor approximation exists for the general traveling salesman problem. In the following sections, we first present HyRE, a novel vertex encoding scheme that can effectively reduce the search space, then detail our proposed query processing strategies in Section 5.

4. HYRE INDEX

In this section, we introduce an indexing technique that encodes the types of relations that vertices participate in, as well as the hierarchical topology of a multi-relation graph. Ideally, such an index benefits the $s \sim t$ CCSP query processing in two ways: first, it enables an immediate decision as to whether s can reach t under a given correlation constraint; second, it significantly reduces the total number of vertex accesses. To achieve this goal, we devise a hybrid relation tree structure, where internal nodes are topological abstractions of graphs. We first present the index structure, and reason how it can be used to significantly reduce the search space for the $s \sim t$ CCSP problem in Section 4.1. Then we present our algorithm to construct the index in Section 4.2, followed by the encoding method in Section 4.3.

4.1 Hybrid Relation Tree

The concept of graph abstraction is defined as follows.

Definition 7 (Graph Abstraction)**.** Given $G = \{V, E\}$, a set of vertex sets $\mathbb{V} = \{V_{r_1}, ..., V_{r_k}\}$, where $V_{r_i} \subset V$, $\nexists V_{r_i} \subset V_{r_j}$ and $\bigcup_{i=1}^{k} V_{r_i} = V$, a graph abstraction of G is a graph $\mathbb{G} = \{\mathbb{V}, \mathbb{E}\}$, where $\mathbb{V} = \{V_{r_1}, ..., V_{r_k}\}$, edge $(V_{r_i}, V_{r_j}) \in \mathbb{E}$ iff $V_{r_i} \bigcap V_{r_j} \neq \emptyset$. We refer the vertices and edges in $\mathbb{G}$ as *abstracted vertices* and *abstracted edges*, respectively.

Intuitively, graph abstraction is another graph whose vertices are partitions of the original graph and an edge exists whenever two partitions overlap. It is straightforward to extend the above definition to a relation-based graph abstraction, where each $V_{r_i} \in \mathbb{V}$ can be a set of vertices correlated under relation r_i. Consider the example shown in Figure 1, where $V^1_{r_1} = \{a, c, d, e, f, g\} \vDash r_1$, $V^2_{r_1} = \{h, j, k\} \vDash r_1$, i.e., two disjoint sets of vertices are correlated under r_1, so the entire graph can be viewed as two abstracted vertices. Note that such an operation can be applied recursively within each abstracted vertex. For example, as shown in Figure 3(a), $V^1_{r_4} = \{a, d, e, f, g\} \vDash r_4$, and $V^1_{r_2} = \{e, f\} \vDash r_2$, two levels of recursive abstractions can be performed on $V^1_{r_1}$ that creates $V^1_{r_1,r_4}$ and $V^1_{r_1,r_2,r_4}$. Meanwhile, $V^1_{r_1}$ is updated to include all the vertices that are correlated under r_1 but not r_4: $V^1_{r_1,r_4} = V^1_{r_1} \setminus V^1_{r_4} = \{c\}$. Conceptually, such a recursive abstraction forms a tree structure. In this tree, every vertex in the child node also participates in the same type of relation as vertices in the parent node. Following Definition 7, we define a hybrid relation tree:

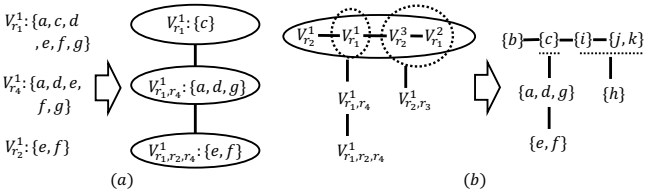


Figure 3: Example of recursive abstraction

Definition 8 (Hybrid Relation Tree)**.** Given a multi-relation graph $G = \langle V, E, \mathbb{R}, F \rangle$, let $S_{r_i} = \{V^0_{r_i}, ..., V^m_{r_i}\}$ be the set of vertex sets that are correlated under r_i, where $r_i \in \mathbb{R}$. A hybrid relation tree, denoted as $\mathcal{T}_G$ is a tree structure defined on $\{S_{r_i} | r_i \in \mathbb{R}\}$, such that (i) the root node is a graph abstraction of G, denoted as $\mathbb{G}^0$; (ii) an internal/leaf node on i-th level, $\mathbb{G}^i_{r_k}$, is a graph abstraction of $V^l_{r_k} \in S_{r_k}$, where $V^l_{r_k}$ is an abstracted node of a graph abstraction on the $(i-1)$-th level.

In the above definition, the subscript of $V^l_{r_i}$ denotes the type of relation that a set of vertices satisfies, and the superscript l denotes the ID of this set, as there may be multiple non-overlapping sets of vertices in G satisfying r_i. The high-level intuition of constructing a hybrid relation tree is to recursively perform the relation-based graph abstraction. To elaborate, consider a general multi-relation example shown in Figure 4. Let each circle represent a set of vertices that satisfies a type of relation, the entire graph can be partitioned into multiple sets of vertices. We capture the relation-based graph abstractions with a tree-like structure shown on the right. At the top level, a graph abstraction defined on relations r_a, r_b, and r_c forms the root node. Its three children are recursively defined within each partition set. For example, another graph abstraction can be derived from V_{r_a}, which are two sets of vertices that satisfy relations r_d and r_e respectively. Note that there can be multiple non-overlapping sets of vertices that satisfy a given relation. As shown in Figure 4, there are two groups of vertices satisfying r_d and r_e, and we differentiate them with $V^1_{r_d}(V^1_{r_e})$ and $V^2_{r_d}(V^2_{r_e})$.

The intuition of adopting such a recursively defined graph abstraction structure is to quickly prune a pair of query vertices (s, t) under a certain correlation constraint. For example, from the structure shown in Figure 4, we know that no two vertices are reachable when $C = \neg r_a \wedge r_d \wedge r_e$, because the set of vertices that satisfies $r_d \wedge r_e$ is correlated under r_a as well.

This hybrid relation index tree can be interpreted as a topological structure of all types of relations that each vertex in G participates in. For example, assume a vertex u is a member of V_{r_e} in Figure 4; following the recursive abstraction process, it is obvious that

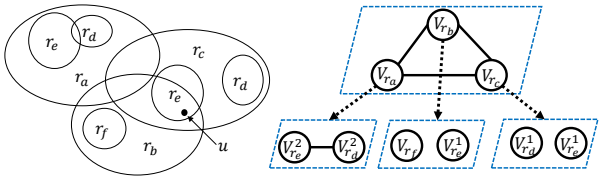


Figure 4: Relation-based graph abstraction

u participates in three relations, $\{r_b, r_c, r_e\}$, but this information is stored as two paths from the root to the internal node where u resides. It indicates that there are two paths to explore if u needs to be visited, implying more space costs on u's encoding to store such information. We call the path from the root to u as u's *prefix relation path*, as formally defined below:

Definition 9 (Prefix Relation Path)**.** A vertex u's prefix relation path in $\mathcal{T}_G$, denoted as $\mathcal{T}_G(u)_{Pre}$ is the path of graph abstraction from the root to the node where u resides, i.e., a sequence of vertex sets that are correlated under different relations, $V^{i_1}_{r^1_1} \rightarrow ... \rightarrow V^{i_k}_{r^k_k}$. We say $\mathcal{T}_G(u)_{PR} = \{r_1, ..., r_k\}$ is the prefix relation of u.

Note that although u may have multiple prefix relation paths depending on the way $\mathcal{T}_G$ is generated, it is always possible to construct a $\mathcal{T}_G$ that guarantees a unique prefix relation path for u, and this will be covered in Section 4.2. For simplicity, we now assume u has an unique prefix relation path for discussion. Following the definition of a vertex's prefix relation path in $\mathcal{T}_G$, we now investigate the pruning power of $\mathcal{T}_G$ for $s \sim t$ CCSP queries. Note that we only consider the following format of correlation constraints for simplicity: (i) $C_{\mathrm{N}} = r_i$; (ii) $C_{\mathrm{N}} = r_i \vee r_j$; and (iii) $C_{\mathrm{D}} = r_i$.

Lemma 3. Given $\mathcal{T}_G$, a pair of vertices (s, t), and a necessity constraint $C_{\mathrm{N}} = r_i$, s reaches t iff $V^{l_s}_{r_i} \in \mathcal{T}_G(s)_{Pre}$ and $V^{l_t}_{r_i} \in \mathcal{T}_G(t)_{Pre}$, where $l_s = l_t$.

Proof. s must participate in relation r_i since $V^{l_s}_{r_i} \in \mathcal{T}_G(s)_{Pre}$, and similarly for t. If $l_s = l_t$, then both s and t are members of a set of vertices V, such that $V \vDash r_i$. Therefore, s can reach t. On the contrary, if $\mathcal{T}_G(s)_{Pre}$ and $\mathcal{T}_G(t)_{Pre}$ do not share a common vertex set that satisfies r_i, then s and t are not connected under C_{N}. □

Lemma 4. Given $\mathcal{T}_G$, a pair of vertices (s, t), and a necessity constraint $C_{\mathrm{N}} = r_i \vee r_j$, s reaches t iff one of the following conditions hold: (i) $r_i \in \mathcal{T}_G(s)_{PR}$, or $r_j \in \mathcal{T}_G(s)_{PR}$; (ii) $r_i \in \mathcal{T}_G(t)_{PR}$, or $r_j \in \mathcal{T}_G(t)_{PR}$; (iii) $\exists V^{l_s}_{r_k}$ from $\mathcal{T}_G(s)_{Pre}$, and $V^{l_t}_{r_m}$ from $\mathcal{T}_G(t)_{Pre}$, such that $V^{l_s}_{r_k}$ and $V^{l_t}_{r_m}$ are reachable in an abstracted graph from $\mathcal{T}_G$.

Proof. The first two conditions guarantee that s and t do participate in relations r_i or r_j, which are implied from Definition 3. We prove the sufficiency and necessity of condition (iii) assuming that the first two conditions are already met. If condition (iii) holds, let $V^{l_s}_{r_k}$ and $V^{l_t}_{r_m}$ be two abstract vertices that are reachable from each other in an abstracted graph $\mathbb{G}$, based on the definition of graph abstraction. We know vertices in $V^{l_s}_{r_k}$ can reach vertices in $V^{l_t}_{r_m}$ through other sets of vertices that overlap with both of them. However, given the fact that no such graph abstraction exists, i.e., the abstracted vertices containing s and t in the root node of $\mathcal{T}_G$, then s and t must not reach each other. □

Intuitively, the above two Lemmas use the prefix relation path for necessity correlation constraint verification. The third condition of Lemma 4 can be relaxed to only perform the checking in the root node of $\mathcal{T}_G$. In addition to reachability test, they also indicate the solution search space of vertices that need exploring to answer the $s \sim t$ CCSP query. However, as we elaborated in Section 1, necessity correlation constraints can be evaluated straightforwardly during graph exploration even without the knowledge of $\mathcal{T}_G$. Therefore, we now focus on using $\mathcal{T}_G$ to effectively handle denial correlation constraints.

Lemma 5. Given $\mathcal{T}_G$, a pair of vertices (s, t), and a denial constraint $C_{\mathrm{D}} = r_i$, if $r_i \in \mathcal{T}_G(s)_{PR}$ or $r_i \in \mathcal{T}_G(t)_{PR}$, s and t are not reachable from each other.

Proof. If $r_i \in \mathcal{T}_G(s)_{PR}$, it implies that s must be correlated with each of its neighbors under relation r_i, therefore, following Definition 3, s cannot reach any other vertex for the given C_{D}. Similar reasoning applies to t. □

Lemma 5 describes a rather simple case where a vertex s cannot reach any other vertices if the given C_{D} rules out the possibility of having s and any of its neighbors appear together in the query answer. There is another case where C_{D} implies a split of G such that vertices from these two partitions become unreachable, as we elaborate below.

Definition 10 (Relation Cut)**.** Given two sets of vertices $V_{r_i} \vDash r_i$, $V_{r_j} \vDash r_j$, and $V_{r_j} \subset V_{r_i}$, if there exists two mutually exclusive vertex sets V_1 and V_2, where $V_1 \bigcup V_2 = V_{r_i} \setminus V_{r_j}$, such that V_1 and V_2 are not connected after removing V_{r_j} and its induced graph, we say r_j is a relation cut of r_i in V_{r_i}.

We define the relation cut concept similar to conventional graph cut, except that a relation cut is a set of vertices that are correlated under a certain type of relation. It is easy to prove the transitivity of relation cut.

Lemma 6. If r_i is a relation cut of r_j in V and r_j is a relation cut of r_k in V', then r_i is a relation cut of r_k in V'.

Proof. Prove by contradiction. Let $V'' \vDash r_i$ and $V'' \subset V$. Assume that r_i is not a relation cut of r_k in V', then there must exist two vertices $s, t \in V' \setminus V''$ that are connected. However, since r_j is a relation cut of r_k in V', by Definition 10, it indicates that $\forall s', t' \in V' \setminus V$, s' and t' are not connected. Since $V' \setminus V \subset V \setminus V''$, s and t must belong to $V \setminus V''$, which contradicts the fact that r_i is a relation cut of r_j in V. So r_i must be a relation cut of r_k in V'. □

Lemma 7. Given $\mathcal{T}_G$, a pair of vertices (s, t), and a denial constraint $C_{\mathrm{D}} = r_i$, s does not reach t if r_i is a relation cut of the root of $\mathcal{T}_G$, such that s and t reside in two separate vertex sets implied by the cut.

Following the transitivity introduced by Lemma 6, Lemma 7 is easy to prove by Definition 10. However, it is nontrivial to decide if a given relation r_i is a relation cut of a given abstracted vertex, or more complicated, to any necessity constraint defined subgraphs. In addition, extra information needs to be recorded in order to decide if s and t become unreachable once r_i is determined to be a relation cut of G. In the following section, we discuss the implementation of the hybrid relation tree and multiple optimization techniques that can be applied to improve the index's pruning power to yield faster query processing.

4.2 Construction Algorithm

Definition 8 defines the hybrid relation tree based on sets of vertex groups that are correlated *w.r.t* each single type of relation, which indicates a straightforward way to compute $\mathcal{T}_G$ following a top-down approach. We can construct $\mathcal{T}_G$ by recursively performing graph abstraction on a set of vertices that are correlated under multiple types of relations. However, this approach is highly inefficient in practice due to the process of choosing $\mathbb{V}$ for each abstraction. This becomes a bottleneck because all the vertex sets that

Algorithm 1: Constructing $\mathcal{T}_G$ (A bottom-up approach)

Data: $G = \langle V, E, \mathbb{R}, F \rangle$, $\mathcal{T}_G \leftarrow \emptyset$, queue $Q \leftarrow \emptyset$, relation vector $\overrightarrow{R}$
Result: $\mathcal{T}_G$
1 $\overrightarrow{R}(u) \leftarrow \langle l_1, ..., l_k \rangle, \forall u \in V$; $V_{tmp} \leftarrow \emptyset$;
2 **while** $V \setminus V_{tmp} \neq \emptyset$ **do**
3 Pick $u \in V$, assume $l = \text{argmin } \overrightarrow{R}(u)$;
4 $V^l_{r_i}$ = merge(v), where $v \in V \setminus V_{tmp}$ and $\overrightarrow{R}(v)_{r_i} = l$;
5 addNode($\mathcal{T}_G, V^l_{r_i}$); $V_{tmp} \leftarrow V_{tmp} \bigcup V^l_{r_i}$;
6 **while** $N \leftarrow \mathcal{T}_G$*'s nodes without a parent* $\wedge N \neq V$ **do**
7 Pick $V^l_{r_i} \in N$, assume $l = \underset{\forall u \in V^l_{r_i} \wedge \mathbb{R} \setminus \{r_i\}}{\text{argmin}} \{\overrightarrow{R}(u)\}$;
8 $V^{l'}_{r_j} = \text{merge}(V^l_{r_i})$, where $v \in V_{r_i}$ s.t. $\overrightarrow{R}(v)_{r_j} = l'$;
9 addNode($\mathcal{T}_G, V^{l'}_{r_j}$); addParent($\{V^l_{r_i}\}, V^{l'}_{r_j}$);
10 abstraction($\mathcal{T}_G$) and **return** $\mathcal{T}_G$;

satisfy a certain type of relation need to be scanned and examined at each time a graph abstraction is performed.

To eliminate this bottleneck, we propose a bottom-up $\mathcal{T}_G$ construction algorithm as shown in Algorithm 1. The overall process is divided into two stages. The essential idea is to recursively merge vertices into groups to form the tree structure at the first stage, then we perform graph abstraction at each tree node from the root to leaves at the second stage. A data structure named *relation vector* $\overrightarrow{R}$ is employed for vertex grouping. We define $\overrightarrow{R}(u)$ as the ID vector of vertex groups *w.r.t* each relation $r_i \in \mathbb{R}$ that u resides in. For example, in Figure 1(a), $\overrightarrow{R}(a) = \langle 1, null, 1, 1 \rangle$ indicates a participates in vertex sets of ID 1 *w.r.t* relation r_1, r_3, and r_4, respectively. It is worth pointing out that the IDs of such vertex sets can be easily computed with a Hash-Min algorithm [35] developed for connected component discovery in the MapReduce framework. It is guaranteed that $V^l_{r_i}$'s ID l is the smallest vertex ID within this set. Assume vertex ID is unique, then $V^l_{r_i}$ and $V^l_{r_j}$ must have overlap (at least the vertex of ID l). Therefore, we use this property in Algorithm 1 to merge vertex sets (Lines 4 and 8). The algorithm is divided into two parts. The first **WHILE** loop constructs all the leaves of $\mathcal{T}_G$. The second **WHILE** loop recursively generates parent nodes level-by-level, until the final node adding to $\mathcal{T}_G$ equals to the entire vertex set of G. Note that we simplify the second stage of performing top-down graph abstractions with abstraction($\mathcal{T}_G$) (Line 11) as it follows the same logic as discussed.

Lemma 8. Algorithm 1 constructs $\mathcal{T}_G$ with time complexity of $O(|V| \times M)$, where $M = \text{argmax}_{u \in V} |T_G(u)_{PR}|$.

Proof. Each vertex u is examined each time child nodes containing u are merged: a total of $|T_G(u)_{PR}|$ times during the bottom-up construction. During abstraction, u again can be visited at most $|T_G(u)_{PR}|$ times. Therefore, the overall cost is $\sum_{u \in V} |T_G(u)_{PR}| \leq |V| \times M$, where $M = \text{argmax}_{u \in V} |T_G(u)_{PR}|$. □

Lemma 8 indicates that the overall complexity of Algorithm 1 depends on the maximum size of prefix relation, i.e., the maximum number of relations that a vertex can participate in. It is worth pointing out that such a conclusion can be derived only because Algorithm 1 guarantees that each vertex has one unique prefix relation path following this bottom-up construction strategy.

Figure 3(b) shows a Hybrid relation tree built from the example shown in Figure 1 following the bottom-up construction. Intuitively, it can be viewed as a mapping from the abstracted graphs to individual vertices. Note that the abstraction can be conducted over a subgraph of the abstracted graph, like $V^1_{r_2,r_3}$ is derived from subgraph "$V^3_{r_2}$–$V^2_{r_1}$". We omit the relation information on the right part of the diagram as it is consistent with the left part.

One significant drawback of Algorithm 1 is that it allows the process of merging child nodes to a parent node only if the parent

Algorithm 2: Constructing $\mathcal{T}_G$ (A heuristic approach)

Data: $G = \langle V, E, \mathbb{R}, F \rangle$, $\mathcal{T}_G \leftarrow \emptyset$, queue $Q \leftarrow \emptyset$, relation vector $\overrightarrow{R}$
Result: $\mathcal{T}_G$
1 $\overrightarrow{R}(u) \leftarrow \langle l_1, ..., l_k \rangle, \forall u \in V$; $\overrightarrow{T}(u) \leftarrow \langle s_1, ..., s_k \rangle, \forall u \in V$;
2 $V_{tmp} \leftarrow \emptyset$;
3 **while** $V \setminus V_{tmp} \neq \emptyset$ **do**
4 Pick $u \in V$, assume $l = \text{argmin } \overrightarrow{R}(u)$;
5 $V^l_{r_i}$ = merge(v), where $v \in V \setminus V_{tmp}$ and $\overrightarrow{R}(v)_{r_i} = l$;
6 addNode($\mathcal{T}_G, V^l_{r_i}$); $V_{tmp} \leftarrow V_{tmp} \bigcup V^l_{r_i}$;
7 **while** $N \leftarrow \mathcal{T}_G$*'s nodes without a parent* $\wedge N \neq V$ **do**
8 Pick $V^l_{r_i} \in N$, assume $l = \underset{\forall u \in V^l_{r_i} \wedge \mathbb{R} \setminus \{r_i\}}{\text{argmin}} \{\overrightarrow{R}(u)\}$;
9 **for** $r_j \in \mathbb{R} \setminus \{r_i\}$ **do**
10 $V^{l'}_{r_j} = \text{merge}(V^l_{r_i})$, where $v \in V_{r_i}$ s.t. $\overrightarrow{R}(v)_{r_j} = l'$;
11 **if** $|V^{l'}_{r_j}| \geq \sum |V^l_{r_i}|$ **then**
12 addNode($\mathcal{T}_G, V^{l'}_{r_j}$); addParent($\{V^l_{r_i}\}, V^{l'}_{r_j}$);
13 $\overrightarrow{T}(u)_{r_i} + \leftarrow \overrightarrow{T}(u)_{r_i}, u \in V^l_{r_i}$;
14 abstraction($\mathcal{T}_G$) and **return** $\mathcal{T}_G$;

node is a superset of all these child nodes. Consider the example shown in Figure 4: following Algorithm 1, the last two leaf nodes need to be merged such that V_e is not split among V_b and V_c. This results in increasing the size of an abstracted node, in terms of the number of vertices represented by this node. Thus, it undermines the pruning power as more vertices need to be considered during the search. On the contrary, although smaller abstracted node size yields better pruning power, extra query processing overhead may incur when vertices can have multiple prefix relation paths, which we shall elaborate in the query processing section (§5). We realize that, to yield the most efficient $s \sim t$ CCSP query processing, there is a nontrivial trade-off between minimizing the size of abstracted nodes and reducing the potential prefix relation paths to explore. This is left for follow-up work. In this paper, we take a heuristic approach to decide the split of an abstracted node such that it always reduces the search space, as shown in Algorithm 2. The difference is that in Algorithm 2 a vertex can now have multiple prefix relation paths (result from the iteration on Line 9-13). In this algorithm, we use additional size information of each vertex group, as indicated by the vector $\overrightarrow{T}(u) \leftarrow \langle s_1, ..., s_k \rangle$, which represents the size of each vertex group that u participates in. This vector can be considered as a side-product of computing $\overrightarrow{R}(u)$. Compared to Algorithm 1, we duplicate a vertex group as much as possible as long as such duplication does not undermine the pruning power of newly formed abstracted vertices (Line 11). The size of each vertex group is updated when new redundancy is introduced (Line 13).

Lemma 9. Algorithm 2 constructs $\mathcal{T}_G$ with time complexity of $O(|V| \times M^2)$, where $M = \text{argmax}_{u \in V} |T_G(u)_{PR}|$.

Lemma 9 can be easily proven following a similar discussion as in the proof of Lemma 8. It is worth pointing out that in practice, the running time of Algorithm 2 is close to Algorithm 1 as the redundancy from vertex set duplication usually quickly outgrows the size of potential parent node, where the condition on Line 13 is evaluated to false in most cases. However, Algorithm 2 results in abstracted vertices of smaller size, which generally reduces the search space and improves the query time efficiency. Detailed evaluation is reported in the experiment section.

4.3 HyRE Encoding

The HyRE index suggests a natural partitioning of G. Vertices in the same abstracted node are most likely to be accessed together for query evaluation, so storing them together would certainly improve

the data locality and therefore reduce I/O swapping. Furthermore, vertices in the same abstracted node share the same prefix relation path, which only needs to be stored once for space cost saving. Thus, putting the vertices within each abstracted node into one partition group would yield a natural decomposition of G. Note that following Algorithm 1, each vertex in G would be assigned to exactly one partition group, while Algorithm 2 would result in a vertex residing in multiple partition groups. As we discussed above, there is a tradeoff between fewer numbers of vertex accesses to speed up query evaluation versus the space overhead introduced by vertex duplication among partitions.

We now describe the vertex encoding scheme to represent the hybrid relation tree. Two major components of this encoding are the relation tree metadata, which essentially is the prefix relation path information, and indexing each vertex's neighbors. The relation tree metadata is a combination of the relation path tree and the connectivity measure within each abstracted node. Consider the example shown in Figure 4 and assume a vertex v belongs to the partition defined in V_{r_f}; then v's prefix relation path is $r_b \rightarrow r_f$. Assume that r_f is a relation cut of r_b, which implies that vertices that reside in $V_{r_b} \setminus V_{r_f}$ do not form a single connected component. Therefore, this connectivity information should also be recorded to assist query evaluation. Our solution is to introduce an additional indicator to record the connectivity measure, which can be either *true* or *false*. Then the relation tree meta data of v would be recorded as r_b(*false*)$\rightarrow r_f$(*true*).

The second component of vertex encoding is to reorganize a vertex's neighbors according to its prefix relation path. Let a vertex v's prefix relation path be $\mathcal{T}_G(v)_{Pre} = r_i \rightarrow ... \rightarrow r_j \rightarrow ... \rightarrow r_k$. All of v's neighbors that correlate with v under r_j are first grouped together, then divided into two subgroups – one for the vertices that share the same prefix relation path of v from r_i to r_j and the other one for those that do not. The intuition of such an organization are two-fold. First, we group each vertex's neighbors *w.r.t* types of relations with which they are correlated, so as to speed up the lookup process (instead of scanning the entire list of neighbors). Second, such an organization is update-friendly. When there are newly added vertices or relations, only an append is necessary to maintain such an organization. Let H denote $\mathcal{T}_G$'s height and $|\mathcal{T}_G|$ denote the number of leaf nodes in $|\mathcal{T}_G|$, we study the overall space complexity of the HyRE index with the following Lemma:

Lemma 10. The space complexity of $\mathcal{T}_G$ is $O(V \times \log(H \times |\mathcal{T}_G|))$.

Proof. For each vertex, the length of its prefix relation path is at most H, and each prefix relation path needs only be stored once, at most $\log(H \times |\mathcal{T}_G|)$ index space is required per vertex. □

As we shall discuss in the experiment section, the space overhead of HyRE index is relatively small compared to the raw data size G. In addition, we can scale up the space cost of $\mathcal{T}_G$ by following the heuristic algorithm presented in Section 4.2 for query efficiency improvement.

5. QUERY PROCESSING

Following the HyRE index discussed above, we first describe a non-conventional exploration-based approach that uses HyRE to effectively prune the search space. Then we elaborate a heuristic approach for the $s \sim t$ CCSP-R problem given in Definition 2.

5.1 Exploration-based Solution

As elaborated in Section 1, the challenge of applying exploration-based algorithms for correlation constraint shortest path queries lies in incorporating the evaluation of denial constraints, which asks that no two vertices that are correlated under a denial constraint can appear in the returned results. In other words, multiple rounds of explorations are needed to ensure that the constraints are satisfied. Note that BFS has been proven to be a more preferable exploration method than DFS to compute hop-based shortest path distance, as discussed in [10]. Other heuristic algorithms for shortest path evaluation, e.g., Dijkstra [16] or A^* [22], are proposed for weighted graphs. These two algorithms behave exact the same as BFS when only a hop-based distance is considered. Therefore, we consider only the BFS exploration approach in this work.

Algorithm 3: Exploration with flags for $s \sim t$ CCSP (Baseline)

```
Data: G = ⟨V, E, ℝ, F⟩, s, t, C
Result: p_st
for v ∈ V do
    v_flag ← true;
    if v ← message from u & u_flag = true then
        dist(s, v) ← dist(s, u) + 1;
        for u ∈ V_{C_D} \ {v} do
            if dist(s, u) is not set then
                u_flag ← false; dist(s, u) ← ∞;
while t does not receive a message from v & dist(s, v) ≠ ∞ do
    v sends v_flag and dist(s, v) to its neighbors;
return p_st;
```

A baseline algorithm extends the breadth-first-search exploration for $s \sim t$ CCSP evaluation, as shown in Algorithm 3. The intuition is to maintain a flag on each visited vertex to indicate whether or not the vertex can appear in the final results. At each iteration of the exploration, whenever a vertex in V_{C_D} is visited from a vertex that does not belong in V_{C_D}, we flag all other vertices in V_{C_D} as "false" indicating that they cannot appear in the final result. In addition, we set the distance from s to these vertices to infinity. Note that we do not eliminate these vertices completely from exploration in the subsequent iterations; all vertices that are visited later from a vertex outside of V_{C_D} can still be considered as candidates. Flagging vertices in a specific V_{C_D} can be effectively supported by the HyRE index as vertices are partitioned based on the correlation groups, which facilitates such a batch processing. It is worth pointing out that Algorithm 3 eliminates all possible vertices that can violate the correlation constraint at each step of exploration, which guarantees the correctness of the returned answer. It is worth pointing out that the time complexity of Algorithm 3 meets the one given in Lemma 1, as Algorithm 3 exactly follows the methodology adopted in the proof of Lemma 1, which guarantees its correctness.

Algorithm 3 is simple and can be adopted without a change to a snapshot-based dynamic graph scenario, i.e., it does not need to wait for the HyRE index to be updated. However, when the graph update is not frequent and snapshot-based query evaluation is sufficient, the shortest path search can be significantly accelerated by considering the topological structure of HyRE, as we show in Algorithm 4. Essentially, we restrict the $s \sim t$ CCSP search space to vertices that reside in a root-to-leaf path in the hybrid relation tree $\mathcal{T}_G$. Given s and t, we need to search their precedents and descendants to find out all valid paths, which is guaranteed by the properties of $\mathcal{T}_G$ as discussed in Lemma 4 – 7. As shown in the algorithm, we explore both directions in one hop within in the WHILE loop until a shortest path is found. Therefore, Algorithm 4 is guaranteed to produce the correct results. Note that in this algorithm, a relation cut is determined during the execution, instead of pre-computed. One key difference to Algorithm 3 is that we no longer need a flag to rule out redundant vertex accesses, since this is already guaranteed when the search follows the topological structure of the HyRE index. It is worth pointing out that worst case running time of Algorithm 4 remains the same as Algorithm 3, which is $O(c|E|)$ given in Lemma 1. This happens when both s and t resides in the root

Algorithm 4: HyRE search for $s \sim t$ CCSP

```
   Data: G = ⟨V, E, ℝ, F⟩, T_G ← ∅, s, t, C
   Result: p_st
1  rc ← false;
2  while rc ≠ true do
3      for V_i ∈ Pre(V_C(s)) do
4          if Connected(V_i=false then
5              rc ← true;
6          for u ∈ V_i do
7              Update(dist(s, u)) at step 1;
8      for V_i ∈ Dec(V_C(s)) do
9          if Connected(V_i=false then
10             rc ← true;
11         for u ∈ V_i do
12             Update(dist(s, u)) at step 1;
13 return p_st;
```

of $\mathcal{T}_G$ and all the descendants need to be searched. On the contrary, the closer s and t locate to leaf nodes in $\mathcal{T}_G$, the smaller is the search space. Based on our bottom-up construction method, most vertices would reside close to leaf nodes, which is why we observe a significant speedup of query evaluation with Algorithm 4.

Consider a simple case study, "to find the shortest path between c and j with constraints $C_{\mathrm{N}} = r_1 \wedge C_{\mathrm{D}} = r_2$". Following the naive exploration approach, multiple vertices $\{f, c, g, h\}$ need to be visited. On the contrary, using a Hybrid relation tree shown in Figure 3(b), a **null** can be immediately returned as c and j are only connected with vertices that participate in relation r_2, indicating no valid path exists for this query.

5.2 $s \sim t$ CCSP-R

We define the $s \sim t$ CCSP-R problem in Section 3 to cover another class of applications where the strength of correlation, measured by the shortest path distance under that type of correlation, plays a more important role in the returned results. For example, to form a small jury group to give unbiased opinions, two people who are 4-hops away on the social network may have never met each other; therefore, a denial constraint on social linkage, like a friend-of-friend relation, is much less crucial than when two people are 1-hop or 2-hop connected. One challenge to address this issue is modeling the decay of correlation strength along the growth of linkage path, which requires domain knowledge and is beyond the scope of this work. As defined in Definition 2, we consider a linear decay-length model for correlation constraint, where two vertices that are correlated under C_{D} are allowed to appear in the results as long as their distance in $V_{C_{\mathrm{N}}}$ is shorter than the one in $V_{C_{\mathrm{D}}}$.

A heuristic algorithm is presented in Algorithm 5 that follows the same framework as in Algorithm 3. We highlight the key difference on line 8 and the **IF** condition on lines 9-10. In this algorithm, we further divide $dist(s, u)$ to two parts: $dist(s, v)$ and $dist(v, u)$. The former denotes the distance from s to a boundary vertex of $V_{C_{\mathrm{D}}}$, while the latter records the distance between two vertices that are correlated under C_{D}, which may be valid to form a result as indicated by the **IF** condition. Note that Algorithm 5 does not guarantee that optimal CCSP-R shortest path would be found. The reason is that it stops as soon as it finds one "short" path that satisfies the CCSP-R definition; however, it eliminates the possibility of discovering more distance-saving vertex pairs that are far from each other in $V_{C_{\mathrm{D}}}$. As discussed in Section 3, $s \sim t$ CCSP-R cannot be approximated within a constant factor. As we discuss in the experiments section (§6), such a heuristic solution can evaluate the $s \sim t$ CCSP-R query very fast and return satisfying results compared to optimal results we have computed in a brute-force manner.

Algorithm 5: Heuristic $s \sim t$ CCSP-R Evaluation

```
   Data: G = ⟨V, E, ℝ, F⟩, s, t, C
   Result: p_st
1  for v ∈ V do
2      v_flag ← true;
3      if v ← message from u & u_flag = true then
4          dist(s, v) ← dist(s, u) + 1;
5          for u ∈ V_C_D \ {v} do
6              if dist(s, u) is not set then
7                  u_flag ← false;
8                  dist(s, u) ← dist(s, v) + dist(v, u);
9              if idst(s, u) is set & dist(v, u) ≥ 2 then
10                 u_flag ← true;
11 while t does not receive a message from v & dist(s, v) ≠ ∞ do
12     v sends v_flag and dist(s, v) to its neighbors;
13 return p_st;
```

6. EXPERIMENTS

We examine our solution from the following three perspectives: (1) the time and space costs of building HyRE index on real-world multi-relation graphs, we vary the space budget for HyRE index and study the trade-off between query processing efficiency and the storage overhead; (2) the performance of answering $s \sim t$ CCSP queries using flag-based exploration and HyRE-based searching, as well as how our solution adapts to dynamic graph updates; and (3) the efficacy and efficiency of $s \sim t$ CCSP-R evaluation.

The highlights of our experimental results on multiple real-world and synthetic graphs are as follows: (1) the time efficiency of the heuristic approach for HyRE construction (Algorithm 2) is close to the bottom-up approach (Algorithm 1), but it can achieve over $5-8\times$ speedup of query processing at a cost of $2.5\times$ storage overhead; (2) Algorithm 2 exhibits more robust performance than Algorithm 1 in the presence of skew in distribution of correlation types (i.e., edge labels); (3) compared with the exploration-based solution (Algorithm 3), HyRE index can significantly reduce the search space, which results in over $5 - 20\times$ speedup in query evaluation on real-world graphs; and (4) the proposed solution demonstrates better performance in a scale-up setting.

6.1 Experimental Setup

We run all our experiments on a Linux server with 32 physical cores, 256GB memory and 2TB hard disk. Scalability tests are run on a cluster of 16 physical machines connected with a gigabit network, where each machine has 8 physical cores, 32GB memory and 1TB hard disk.

Implementation. The prototype system is implemented in C++, compiled using GCC 5.4.0, with O3 switch on. We incorporate the Intel Thread Building Block (TBB) framework to make use of multiple cores for HyRE construction, and MPICH 3.3 is used for the communication. Specifically, we implement parallel versions of Algorithm 2 and Algorithm 3. Algorithm 2 constructs $\mathcal{T}_G$ in a bottom-up fashion, where the computation tasks on each vertex is the same and therefore can be embarrassingly parallelized. The challenge of building $\mathcal{T}_G$ in parallel following the heuristic algorithm lies in the split validation process (given on lines 9 and 13), which requires a scan of other vertices that satisfy the correlation constraint. Our solution is to parallelize Algorithm 3 at the level of subgraphs (which is derived from an off-shelf graph partitioning toolkit [27]). In addition, we run the query processing algorithms following a subgraph-centric computing framework (similar to Blogel [51]), which executes the path searches serially within each subgraph. We set the total number of graph partitions the same as the number of physical cores.

Table 1: Real-World Graph Datasets.

G	$\|V\|$	$\|E\|$	$\|\mathbb{R}\|$	$\overline{\|R_v\|}$	$\overline{\|R_e\|}$	$\mu_{\frac{\|V_R\|}{\|V\|}}$	$\sigma_{\frac{\|V_R\|}{\|V\|}}$	HyRE (Bottom-up)		HyRE (Heuristic)		T(sec)
								Time (Sec)	Size (GB)	Time (Sec)	Size (GB)	
Youtube*(You)	15K	13M	5	3.2	2.4	0.27	0.02	9.2	0.025	9.7	0.037	18.6
Flickr*(Fli)	105K	2M	214	4.7	1.4	0.17	0.26	14.2	2.4	16.7	5.1	12.4
STRING (STR)	72.9M	117M	6	4.2	3.8	0.37	0.06	672	67.4	789	74.2	189.2
MAG*	166M	3.3B	8	5.1	1.9	0.74	0.03	1007	246	1147	297	97.4

Table 2: Correlation types and constraints adopted in query evaluation.

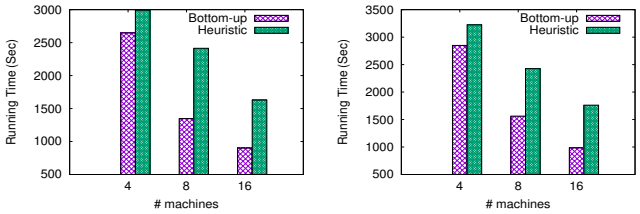
	Youtube	STRING	Flickr	MAG
$\mathbb{R}$	r_1: Contact r_2: Co-contact r_3: Co-subscription r_4: Co-subscribed r_5: Favorite	r_1: Conserved neighborhood r_2: Phylogenetic co-occurrence r_3: Co-expression r_4: Large-scale experiments r_5: Literature co-occurrence r_6: Combined score	r_1: SameTag r_2: SameCollection r_3: SameGroup r_4: SameGallery r_5: SameLocation r_6: SameUser r_7: TakenByFriend	r_1: AuthorInCommon r_2: SameYear r_3: KeywordsInCommon r_4: SameVenue r_5: SameType r_6: Cited r_7: Co-cited r_8: SameAffiliation
C	1.$C_{\mathrm{N}} = r_5, C_{\mathrm{D}} = \emptyset$ 2.$C_{\mathrm{N}} = r_5, C_{\mathrm{D}} = r_1$ 3.$C_{\mathrm{N}} = r_5, C_{\mathrm{D}} = r_1 \vee r_2$ 4.$C_{\mathrm{N}} = \emptyset, C_{\mathrm{D}} = r_4$ 5.$C_{\mathrm{N}} = r_1, C_{\mathrm{D}} = r_3$	1.$C_{\mathrm{N}} = r_2, C_{\mathrm{D}} = r_5$ 2.$C_{\mathrm{N}} = r_3, C_{\mathrm{D}} = \emptyset$ 3.$C_{\mathrm{N}} = r_3 \vee r_4, C_{\mathrm{D}} = r_1$ 4.$C_{\mathrm{N}} = \emptyset, C_{\mathrm{D}} = r_5$ 5.$C_{\mathrm{N}} = r_5, C_{\mathrm{D}} = r_4$	1.$C_{\mathrm{N}} = r_1, C_{\mathrm{D}} = r_6$ 2.$C_{\mathrm{N}} = r_2 \vee r_3 \vee r_4, C_{\mathrm{D}} = r_1$ 3.$C_{\mathrm{N}} = r_1 \wedge r_6, C_{\mathrm{D}} = r_5$ 4.$C_{\mathrm{N}} = r_1 \wedge r_6, C_{\mathrm{D}} = r_6$ 5.$C_{\mathrm{N}} = r_7, C_{\mathrm{D}} = r_1$	1.$C_{\mathrm{N}} = r_2 \wedge r_4, C_{\mathrm{D}} = r_1$ 2.$C_{\mathrm{N}} = r_1 \wedge r_3, C_{\mathrm{D}} = r_2$ 3.$C_{\mathrm{N}} = r_1 \wedge r_3, C_{\mathrm{D}} = r_4$ 4.$C_{\mathrm{N}} = r_5 \wedge r_3, C_{\mathrm{D}} = r_6 \vee r_7$ 5.$C_{\mathrm{N}} = r_8 \wedge r_3, C_{\mathrm{D}} = r_6 \vee r_7$

Datasets. As summarized in Table 1, we use four real-world graphs with different characteristics. Note that the graphs appended with "*" are undirected graphs. Youtube graph [44] depicts a user contact network, where vertices are users, edges are formed under different conditions, e.g., two users subscribe to the same channel Each condition is treated as a type of correlation in our experiments. Flickr graph [32] is an image relation network built from pictures posted online, where vertices are pictures and edges are the correlations among pictures. Over 200 types of relations are defined in this data set, and their frequency is highly skewed. STRING [42] is a Protein-Protein Interaction network, where vertices are proteins and edges represent different types of empirically learned chemical interactions. Microsoft Academic Graph [34] (MAG) is a collection of over 166 million papers.We identify 8 different types of correlations and construct the graph by determining how each paper connects to others from a random subset of the entire collection. In Table 1, $\overline{|R_v|}$ and $\overline{|R_e|}$ denote the average relation type counts on each edge and vertex, respectively. We also report the mean and standard deviation of the percentage of vertices that participate in a relation with $\mu_{\frac{|V_R|}{|V|}}$ and $\sigma_{\frac{|V_R|}{|V|}}$. We observe that when the per-vertex/per-edge relation counts are close, this indicates that correlated vertices are more likely to participate in the same types of relations, which is true for Youtube and STRING. The small $\sigma_{\frac{|V_R|}{|V|}}$ value of these two graphs also verifies this. In contrast, Flickr has a very skewed distribution of the types of relations among vertices, indicating an image often correlates other images under different types of relations. Although MAG also demonstrates relatively large difference on per-vertex/per-edge average relation count, the small $\sigma_{\frac{|V_R|}{|V|}}$ shows that these types of relations are evenly distributed in the graph.

Workload. For each graph, we design 5 different correlation constraints and first compute the ground truth results of 1000 random queries using the flag-based exploration method (Algorithm 3). This allows us to study the pruning power of HyRE for invalid input vertex pairs, as well as its efficiency in answering queries with valid inputs. We report the average running time over 5 runs with cold start. Table 2 summarizes the correlation types and constraints.

6.2 Computing HyRE

We summarize the computation of HyRE on four graphs in Table 1. We compare the space and time cost of HyRE construction using different algorithms. There are some observations we make from this set of experiments. First, the heuristic approach generally add less than 20% of extra construction time compared to the bottom-up approach, and the storage overhead does not grow exponentially when we allow a vertex to keep more prefix relation paths. As we shall show later in the query evaluation study, this enables the heuristic-based index to perform much better than the index built using the bottom-up approach. Second, the time complexity of building the HyRE index is mainly dominated by two factors: the number of edges, and the average number of correlations between vertices. For example, although Flickr has much fewer edges than Youtube, it takes longer to process because it is affected by the high average number of correlations (12 in our experiments). Note that the reported time costs in Table 1 include the vertex encoding process, which is only proportional to $|V|$. The last column in Table 1 reports the total execution makespan achieved using Algorithm 4 over 1000 queries. As detailed in the following section, two orders of magnitude speedup can be achieved using HyRE, which justifies the overhead of its construction time. To verify the scalability of the HyRE construction algorithm, we run two algorithms in a scale-out fashion on STRING and MAG. As shown in Figure 5, the bottom-up approach demonstrates better scalability due to its ease of parallelization. In contrast, the heuristic approach given in Algorithm 2 relies on more intermediate results, and this leads to significant data exchange in a distributed setting. Therefore, the heuristic approach favors a scale-up architecture.

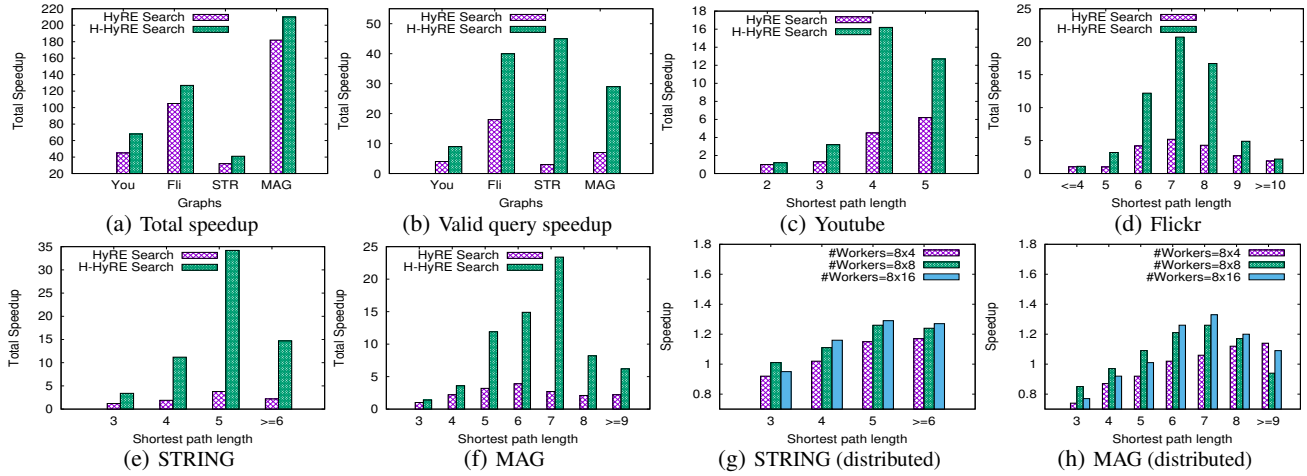


(a) STRING (b) MAG

Figure 5: Scale-out computation of HyRE

6.3 $s \sim t$ CCSP Query Processing

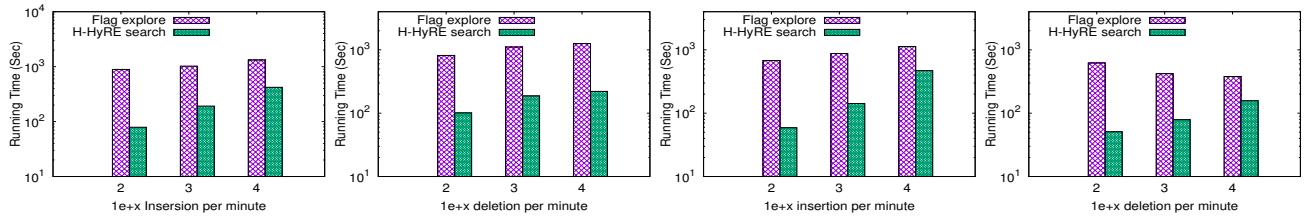
We summarize the $s \sim t$ CCSP query processing results in Figure 6. Note that we consider the flag-based exploration approach



(a) Total speedup (b) Valid query speedup (c) Youtube (d) Flickr

(e) STRING (f) MAG (g) STRING (distributed) (h) MAG (distributed)

Figure 6: $s \sim t$ CCSP evaluation: (a) Total speedup on 1K random queries; (b) Total speedup on queries with valid shortest path results; (c)-(f) Graph specific CCSP evaluation; (g)-(h) Scale-out $s \sim t$ CCSP evaluation.



(a) STR Insertion (b) STR Deletion (c) MAG Insertion (d) MAG Deletion

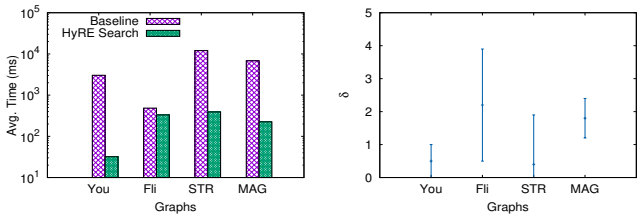
Figure 7: $s \sim t$ CCSP evaluation with graph updates. (Snapshot taken every minute)

(Algorithm 3) as a baseline and report the speedup of Algorithm 4 here. Figure 6(a) shows the overall performance improvement over 1000 random queries compared with the baseline, while Figure 6(b) shows the speedup for queries that do not return null. Algorithm 4 demonstrates much better efficiency, especially for random queries where HyRE can reach an early stop much faster if the result is null. Another observation is related to the performance of using HyRE for query processing varies on different graphs. For example, HyRE speedup on STR is much smaller than on MAG. The reason is that vertices in STR tend to be correlated under a similar set of relation types, which weakens the pruning power of HyRE.

We compare the query efficiency using different HyRE indexes built with both Algorithms 1 and 2. As discussed in Section 4, the heuristic-based HyRE construction approach enables the tradeoff between space overhead and much fewer vertex access. The experiments show that Algorithm 2 consistently beats Algorithm 1 in facilitating query processing. Although the overall speedup ratio varies on different graphs, we observe at least one order of magnitude of efficiency improvement. Note that evaluating queries without valid output using the baseline algorithm can be very time consuming, while HyRE guarantees an early stop property, which contributes the most in the speedup numbers shown in Figure 6(a). On the contrary, if we only look at the queries that do have valid results, the speedup ratio of our solution is very different, as shown in Figure 6(b). From this figure, we observe that allowing extra space in HyRE construction is rewarding. Table 1 shows that, with a $2-3\times$ increase in storage cost, we can achieve $5-10\times$ faster query processing. Figures 6(c)-(f) gives the $s \sim t$ CCSP query evaluation results with respect to the length of returned results. We find that the performance improvements are strongly correlated with two factors: (1) the length of returned results, and (2) the correlation distribution, or the density of the subgraphs that are traversed for query processing. For example, Flickr is much sparser than Youtube, therefore, when the shortest path distance is too long or too short, there is not much performance improvement, as the traversal does not incur much redundant vertex access. On the contrary, we observe significant performance gains when the length of shortest path increases in Youtube. The rationale behind this is that a longer path would incur much more vertex visits in this graph. A similar trend is observed on other graphs as well.

Among all four datasets, the best performance is observed on the STRING graph. The explanation is two-fold. First, compared to other datasets, the correlation types are relatively more evenly distributed among vertices. Second, this graph is featured with numerous dense clusters, which would benefit query processing if vertices within the same cluster are stored closely together, which is essentially guaranteed by our encoding scheme. Note that the results in Figure 6(a)-(f) are obtained by running the queries on a single server. In addition, we compare the running time of $s \sim t$ CCSP evaluation on STRING and MAG in a distributed setting, as shown in Figure 6(g) and 6(h). We compare using 4, 8, and 16 machines with 8 workers per-machine. The baseline in this case is set as the best performance reported from experiments conducted on a single server. When the total number of workers grows, slight efficiency improvements (15%-30%) are observed. This is because the complexity of shortest path search is dominated by the number of vertex accesses, which does not benefit much from parallel computing.

We also test how HyRE can accommodate graph updates in a dynamic setting. Note that we assume all queries are evaluated based on snapshots of the graph, therefore, we only consider batch updates to HyRE and do not consider streaming graphs. Specifically, we consider two atomic update operations: add/delete a relation to



(a) Efficiency of computing CCSP-R (b) Efficacy of computing CCSP-R

Figure 8: $s \sim t$ CCSP-R evaluation

an existing edge. The bottom-up approach (Algorithm 1) would require constructing the entire index structure from scratch, which can be time-consuming. Instead, there is the opportunity of trading off the re-construction time by allowing a vertex to have multiple prefix relation paths, as we demonstrated in Algorithm 2. Thus, given the updates, we would simply append a vertex's prefix relation paths if a new relation is added, or remove one of its prefix relation paths if a relation is deleted. The drawback of this approach is that it can increase the storage cost as well as the search space, which would degrade the query efficiency. Figure 7 shows the efficiency of $s \sim t$ CCSP query under graph updates on STRING and MAG. We compare the query efficiency of Algorithm 3 and Algorithm 4 over different patterns of graph updates. We observe that HyRE-based search is more sensitive to insertion, because such an operation continuously allocates more space for the index and potentially enlarges the search space. Another interesting fact observed for the MAG data set is that constant deletion leads to lower total query time. This is because the total number of valid paths that can be returned significantly reduces when edges are constantly deleted from such a sparse graph. As we trade off the update cost with the pruning power of HyRE, periodical re-construction of the index from scratch needs to be performed to satisfy the query efficiency requirement.

6.4 $s \sim t$ CCSP-R Query Processing

We define the $s \sim t$ CCSP-R problem in Definition 2 as a relaxation of the strict correlation constraint condition. As shown in Algorithm 5, we can again use HyRE as a guidance in evaluating $s \sim t$ CCSP-R queries. We empirically study the efficiency and efficacy of Algorithm 5 by comparing it to a brute-force approach that enumerates and verifies all possible shortest paths between s and t in an ascending order of the path length until the first qualified path is discovered. We run 1000 random queries and only report the average evaluation time of the ones that have valid returns. As shown in Figure 8(a), compared to the baseline approach, the heuristic search over HyRE is at least one order of magnitude faster on most graphs. This is because our search method would stop immediately after finding a path between s and t, which saves significant verification overhead. However, we observe that the efficiency improvement on Flickr graph is minor, which is mainly caused by two factors: the skewed relation distribution and the sparsity of the graph. Since the graph is sparse, and the average number of relations on each edge is low (as shown in Table 1), the pruning power of HyRE is limited. The efficacy of Algorithm 5, as shown in Figure 8(b), is reasonable given its huge savings on evaluation time. In this figure, δ on the y-axis depicts the difference between the true shortest path distance and the returned value. The reported results are based on HyRE built with the heuristic approach (Algorithm 2). We further observe that the performance of this index, in terms of efficiency and efficacy of $s \sim t$ CCSP-R evaluation, is not always superior to HyRE built with the bottom-up approach (Algorithm 1), due to the fact that all of the vertices (even the ones correlated under denial constraints) need to be followed for path examination.

We consider Q5 over MAG as a case study to demonstrate the real-world semantic of the $s \sim t$ CCSP and CCSP-R queries. The query is interpreted as "find a minimum set of papers including the given s and t that are of the same type and with keywords in common but do not cite or co-cite each other." Under the CCSP semantic, it could find papers that are relevant but missing references among others, which can be utilized to predict linkage or recommend citations. While under the CCSP-R semantic, where the co-/cite correlations are allowed among the returned results, it suggests a small set of publications that are generally related to the inputs, which should be examined to have a broad understanding of the particular research field.

7. CONCLUSION

In this work, we focus on the generic correlation constraint pairwise shortest path computing on multi-relation graphs, with the observation that existing edge label constraint solutions cannot be directly adopted to satisfy negative correlation conditions, which is defined and studied as denial constraints in this paper. We formally define the $s \sim t$ CCSP and its relaxation problem, which leads to a novel vertex encoding scheme that can effectively encode both topological structure and relation types, such that the shortest path workload can be evaluated with significantly fewer vertex access. Extensive experiments on real-world graphs are conducted to validate our proposed solution.

We used shortest path to demonstrate the approach, but the data structures and the techniques we propose can be applied more widely. Although this is beyond the scope of this paper, we highlight how the data structures and methods can be applied to two other workloads to demonstrate their wider applicability. The way we encode each vertex is derived from HyRE, which can serve as a generic logical structure for any type of correlation constraint path queries. For example, given a correlation constraint reachability query, we can evaluate it without traversing the graph, but by simply looking into the HyRE index since it records all the relation cuts which is sufficient to determine if two vertices are reachable under given relation constraints. In addition, traversal-based evaluation of online graph pattern matching is proven to be efficient in graph database systems. As HyRE reflects not only the relation types that a vertex participates in, but also its local neighborhood topology, it can be used as an effective filter to reduce the search space for pattern matching queries that incorporates correlation constraints.

Although HyRE is a generic indexing technique that can be applied to multi-relation graphs, there are multiple aspects we would like to investigate further in future work: (1) quantify the tradeoff between extra space cost of HyRE and its potential performance gain; (2) explore the opportunity of reducing the search space when a correlation function is available such that the pattern of vertex correlation can be taken into consideration; and (3) incorporate application scenarios where graph updates come in as a stream and the shortest path is monitored based on a sliding window semantic; and (4) study whether an optimal HyRE structure exists for a particular type of graphs.

8. ACKNOWLEDGMENTS

We thank reviewers' valuable comments. This research was supported in part by Natural Sciences and Engineering Research Council (NSERC) of Canada.

9. REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. F. Werneck. Hierarchical hub labelings for shortest paths. In *Proc. 20th Annual European Symp. on Algorithms*, pages 24–35, 2012.

[2] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 349–360. ACM, 2013.

[3] H. V. N. Y. Angela Bonifati, George Fletcher. Querying graphs. In *Synthesis Lectures on Data Management, Morgan Claypool*, 2018.

[4] R. Angles. The property graph database model. In *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2018.

[5] C. Avery. Giraph: Large-scale graph processing infrastructure on hadoop. *Proceedings of the Hadoop Summit. Santa Clara*, 2011.

[6] M. Baitaluk, X. Qian, S. Godbole, A. Raval, A. Ray, and A. Gupta. Pathsys: integrating molecular interaction graphs for systems biology. *BMC Bioinformatics*, 7(1):55, Feb 2006.

[7] P. Barceló, C. A. Hurtado, L. Libkin, and P. T. Wood. Expressive languages for path queries over graph-structured data. In *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 3–14. ACM, 2010.

[8] C. L. Barrett, K. R. Bisset, R. Jacob, G. Konjevod, and M. V. Marathe. Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the TRANSIMS router. In *Proc. 10th Annual European Symp. on Algorithms*, pages 126–138, 2002.

[9] H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. Proc. 9th workshop on algorithm eng. and experiments. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments*, 2007.

[10] S. Beamer, K. Asanovic, and D. A. Patterson. Direction-optimizing breadth-first search. *Scientific Programming*, 21(3-4):137–148, 2013.

[11] F. Bonchi, A. Gionis, F. Gullo, and A. Ukkonen. Distance oracles in edge-labeled graphs. In *Proc. 17th Int. Conf. on Extending Database Technology*, pages 547–558, 2014.

[12] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185. Morgan Kaufmann, 2000.

[13] A. Cardillo, J. Gómez-Gardeñes, M. Zanin, M. Romance, D. Papo, F. del Pozo, and S. Boccaletti. Emergence of network features from multiplexity. *CoRR*, abs/1212.2153, 2012.

[14] M. Coscia, G. Rossetti, D. Pennacchioli, D. Ceccarelli, and F. Giannotti. You know because i know: a multidimensional network approach to human resources problem. In *Advances in Social Networks Analysis and Mining*, pages 434–441, 2013.

[15] J. Dibbelt, T. Pajor, and D. Wagner. User-constrained multimodal route planning. *ACM J. Experimental Algorithmics*, 19(1), 2014.

[16] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[17] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. *Front. Comput. Sci.*, 6(3):313–338, 2012.

[18] D. Florescu, A. Y. Levy, and D. Suciu. Query containment for conjunctive queries with regular expressions. In *Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, pages 139–148. ACM Press, 1998.

[19] R. Geisberger, M. N. Rice, P. Sanders, and V. J. Tsotras. Route planning with flexible edge restrictions. *ACM J. Experimental Algorithmics*, 17(1), 2012.

[20] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proc. Int. Workshop on Experimental and Efficient Algorithms*, pages 319–333, 2008.

[21] A. Gubichev, S. Bedathur, S. Seufert, and G. Weikum. Fast and accurate estimation of shortest paths in large graphs. In *Proc. 19th ACM Int. Conf. on Information and Knowledge Management*, pages 499–508. ACM, 2010.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

[23] M. S. Hassan, W. G. Aref, and A. M. Aly. Graph indexing for shortest-path finding over dynamic sub-graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 1183–1197, 2016.

[24] J. Håstad. Clique is hard to approximate within $n^{1-epsilon}$. In *FOCS*, pages 627–636. IEEE Computer Society, 1996.

[25] Janusgraph: Distributed graph database. `http://janusgraph.org/`, 2018.

[26] A. Kanterakis, D. Kafetzopoulos, V. Moustakis, and G. Potamias. Mining gene expression profiles and gene regulatory networks: Identification of phenotype-specific molecular mechanisms. In *Proc. Artificial Intelligence: Theories, Models and Applications, 5th Hellenic Conf. on AI*, pages 97–109, 2008.

[27] D. Lasalle and G. Karypis. Multi-threaded graph partitioning. In *Proc. 27th IEEE Int. Parallel & Distributed Processing Symp.*, pages 225–236, 2013.

[28] Y. Liang and P. Zhao. Similarity search in graph databases: A multi-layered indexing approach. In *Proc. 33st Int. Conf. on Data Engineering*, pages 783–794, 2017.

[29] S. Ma, K. Feng, J. Li, H. Wang, G. Cong, and J. Huai. Proxies for shortest path and distance queries. *IEEE Trans. Knowl. and Data Eng.*, 28(7):1835–1850, 2016.

[30] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 135–146. ACM, 2010.

[31] M. Mao, J. Lu, G. Zhang, and J. Zhang. Multirelational social recommendations via multigraph ranking. *IEEE Trans. Cybernetics*, 47(12):4049–4061, 2017.

[32] J. J. McAuley and J. Leskovec. Image labeling on a network: Using social-network metadata for image classification. In *ECCV (4)*, volume 7575 of *Lecture Notes in Computer Science*, pages 828–841. Springer, 2012.

[33] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. on Comput.*, 24(6):1235–1258, 1995.

[34] Microsoft academic graph. `https://www.openacademic.ai/oag/`, 2015.

[35] V. Rastogi, A. Machanavajjhala, L. Chitnis, and A. D. Sarma. Finding connected components in map-reduce in logarithmic rounds. In *Proc. 29th Int. Conf. on Data Engineering*, pages 50–61, 2013.

[36] M. N. Rice and V. J. Tsotras. Graph indexing of road networks for shortest path queries with label restrictions. *PVLDB*, 4(2):69–80, 2010.

[37] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 43–54, 2008.

[38] A. Sealfon. Shortest paths and distances with differential privacy. In *Proc. 35nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 29–41, 2016.

[39] P. Selmer, A. Poulovassilis, and P. T. Wood. Implementing flexible operators for regular path queries. In *Proc. 18th Int. Conf. on Extending Database Technology*, pages 149–156, 2015.

[40] T. Shulong, B. Jiajun, C. Chun, and H. Xiaofei. Using rich social media information for music recommendation via hypergraph model. In *ACM Trans. Multimedia Comp., Comm., and Appl.*, volume 7S, pages 213–237. 2011.

[41] M. Stella, C. S. Andreazzi, S. Selakovic, A. Goudarzi, and A. Antonioni. Parasite spreading in spatial ecological multiplex networks. *J. Complex Networks*, 5(3):486–511, 2017.

[42] D. Szklarczyk, A. Franceschini, S. Wyder, K. Forslund, D. Heller, J. Huerta-Cepas, M. Simonovic, A. Roth, A. Santos, K. P. Tsafou, M. Kuhn, P. Bork, L. J. Jensen, and C. von Mering. String v10: protein–protein interaction networks, integrated over the tree of life. *Nucleic Acids Research*, 43(D1):D447, 2014.

[43] D. Szklarczyk, J. H. Morris, H. Cook, M. Kuhn, S. Wyder, M. Simonovic, A. Santos, N. T. Doncheva, A. Roth, P. Bork, L. J. Jensen, and C. von Mering. The STRING database in 2017: quality-controlled protein-protein association networks, made broadly accessible. *Nucleic Acids Research*, 45(Database-Issue):D362–D368, 2017.

[44] L. Tang, X. Wang, and H. Liu. Community detection via heterogeneous interaction analysis. *Data Min. Knowl. Discov.*, 25(1):1–33, 2012.

[45] The neo4j graph platform. `https://neo4j.com/`, 2018.

[46] L. D. J. Valstar, G. H. L. Fletcher, and Y. Yoshida. Landmark indexing for evaluation of label-constrained reachability queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 345–358, 2017.

[47] S. Wang, Y. Tsai, T. Hong, and H. Kao. k-anonymization of multiple shortest paths. *J. Soft Comput.*, 21(15):4215–4226, 2017.

[48] S. Wang, Z. Tsai, T. Hong, and I. Ting. Anonymizing shortest paths on social network graphs. In *Proc. 3rd Int. Conf. on Intell. Information and Database Syst.*, pages 129–136, 2011.

[49] F. Wei. Tedi: Efficient shortest path query answering on graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 99–110, 2010.

[50] F. Wei. TEDI: efficient shortest path query answering on graphs. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 99–110, 2010.

[51] D. Yan, J. Cheng, Y. Lu, and W. Ng. Blogel: A block-centric framework for distributed computation on real-world graphs. *PVLDB*, 7(14):1981–1992, 2014.