Visual Summaries for Graph Collections

David Koop* NYU-Poly
Juliana Freire* NYU-Poly
Cláudio T. Silva* NYU-Poly

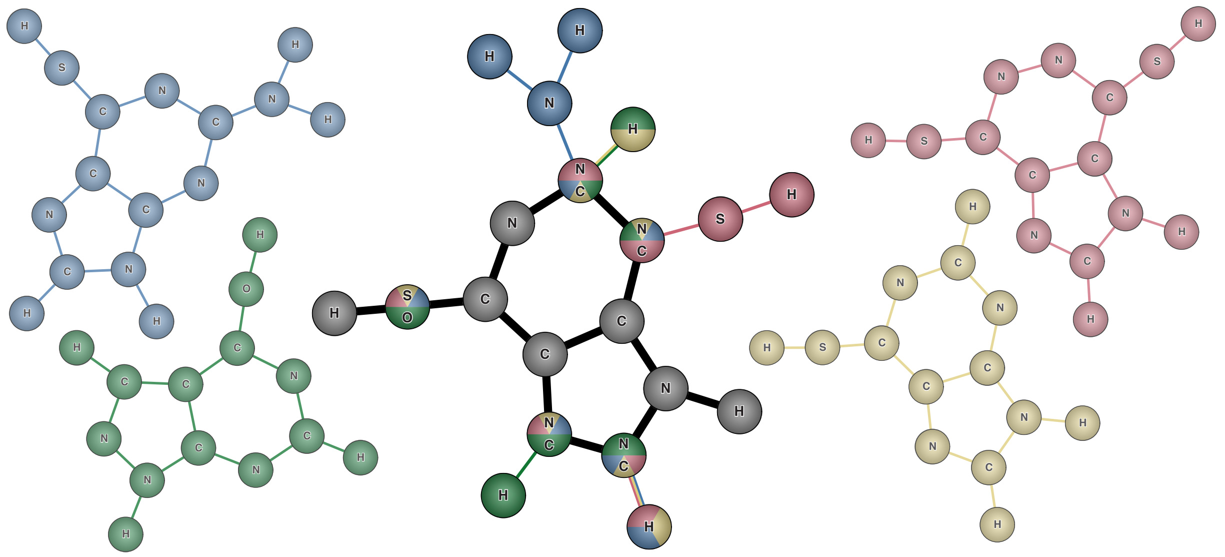


Figure 1: A summary graph (middle) constructed from four graphs of molecules; colors indicate which input graphs contain a given feature with gray indicating the feature is common to all graphs.

Abstract

Graphs can be used to represent a variety of information, from molecular structures to biological pathways to computational workflows. With a growing volume of data represented as graphs, the problem of understanding and analyzing the variations in a collection of graphs is of increasing importance. We present an algorithm to compute a single summary graph that efficiently encodes an entire collection of graphs by finding and merging similar nodes and edges. Instead of only merging nodes and edges that are exactly the same, we use domain-specific comparison functions to collapse similar nodes and edges which allows us to generate more compact representations of the collection. In addition, we have developed methods that allow users to interactively control the display of these summary graphs. These interactions include the ability to highlight individual graphs in the summary, control the succinctness of the summary, and explicitly define when specific nodes should or should not be merged. We show that our approach to generating and interacting with graph summaries leads to a better understanding of a graph collection by allowing users to more easily identify common substructures and key differences between graphs.

1 Introduction

Graphs provide a natural structure to represent a wide array of data including molecular structures, social networks, biological pathways, and computational workflows. The ability to analyze and visualize these graphs is essential to gain insight into the data and relationships represented. In many applications, users need to interact with collections of related graphs—graphs that while different share similar overall structure or have common substructures. In exploring such collections, one important task is to identify the similarities and differences between the graphs. For example, a medicinal chemist may be interested in comparing a set of molecules with similar properties to determine where specific drugs differ [44], or a visualization expert, given a series of pipelines for producing a given type visualization, may be interested in different strategies (or techniques) used. Such questions are difficult to answer when graphs are visualized independently, and this baseline approach does not scale well. We introduce summary graph visualization to display a collection of related graphs in a single, interactive view.

While there has been work on visual representations that support graph comparison, they have tended to focus on pairs of graphs where the correspondences are given [21, 4]. In addition, much of this work has been domain-specific (*e.g.,* workflows [22], provenance traces [6], or dependency graphs [8]). Our approach generalizes this idea to *create a visualization of more than two graphs in a single view*. A summary graph visualization is a single graph that represents data and structure from all of its input graphs.

One of the challenges in composing a collection of graphs into a single visualization is that there is no known computationally efficient method for comparing them. Testing whether one graph is an exact subgraph of another (subgraph isomorphism) is NP-Complete [27], and relaxing constraints to allow for greater freedom (*e.g.,* by allowing nodes that belong to the same class to match) complicates things further. However, in many cases, there is significant overlap among related graphs in a collection, and aligning similar substructures is less daunting than the general problem. We discuss methods that are effective at matching nodes and preserving common substructures.

We propose an algorithm that makes use of these methods to build a summary graph using hierarchical agglomeration, where intermediate summary graphs are constructed recursively from pairwise matchings. The final summary consists of a graph where each node (edge) represents at most one node (edge) from each of the input graphs. Using color, lightness, and edge thickness, we can display relative frequencies of common features and highlight specific differences. We note that the proposed algorithms and tools

*e-mail:{dkoop,juliana,csilva}@poly.edu

were designed to handle collections of (relatively) small graphs that are related. To use our approach for more diverse collections, it is best to pre-process the collection to cluster the graphs into subcollections, and then create a summary for each subcollection.

In addition, we propose tools for interactively exploring and manipulating the summary graph visualization. In a summary graph, there are tradeoffs between matching equivalent nodes and preserving structures. Thus, there are situations where a user may wish to modify the visualization to emphasize similar structures even if merged nodes are not as well related. We provide break and join operations to allow users to make such modifications. In addition, users can control the amount of summarization and examine specific relationships between a subset of the input graphs.

In this paper, we define summary graphs, show how they can be constructed and visualized, and present use cases and studies of their effectiveness. We begin by reviewing related work, then define a summary graph and give an overview of the technique in Section 3. In Section 4, we present algorithms to construct pairwise graph matchings, and in Section 5, we detail how to construct a summary graph from a graph collection. Next, we describe the display of summaries and how users can manipulate them (Section 6). In Section 7, we discuss specific applications, and then detail results of a user study in Section 8. We conclude with a discussion of future work in Section 9.

2 Related Work

There has been substantial work in the area of graph visualization, including layout algorithms, methods for visualizing large graphs, and techniques for interacting with them [31, 47]. In addition, there has been work on visualizing multiple trees and layout algorithms for multiple or evolving graphs. Our work focuses on the interactive visualization of a collection of graphs as a single graph where the relationships between the input graphs are not given.

While general graph collections have seen less study, there has been some significant work on visualizing collections of trees. Furnas and Zacks suggested Multitrees as a way to integrate sets of hierarchical information [24], and the InfoVis 2003 Contest generated significant work in tree comparison [39]. Much of it was rooted in work done for consensus trees used for phylogenies in the biological community [1]. Munzer et al.'s TreeJuxtaposer tackled the problem of comparing large trees using focus+context and visibility criteria [38]. Graham and Kennedy used directed acyclic graphs to agglomerate multiple trees [28] and also provide a comprehensive survey of work on in the area of visualizing multiple trees [29]. Recently, Bremm et al. introduced a visual approach for analyzing *multiple* hierarchies instead of the normal pairwise approaches, using linked hierarchy views [12].

Another related area is graph summarization and interaction techniques which focus on helping users analyze single graphs that do not naturally fit in reasonably-sized views. These demand some aggregation to allow scalable analysis [20]. Such graphs can be clustered or summarized with regions collapsed into smaller entities (*e.g.,* [19]). Other approaches have used topology [3] and interaction [25] to better navigate graphs. In addition, edges can be bundled to help users better show connectivity when graphs have large numbers of edges [33] . Level-of-detail can be used to more efficiently navigate large graphs [5]. There are also techniques for multivariate graphs that focus on relationships between nodes [49]. While our focus is on combining a collection of graphs into a single visualization, graph summarization could certainly be used to display the final summary graph, especially if that result is large.

For graph collections, previous work has examined the differences between graphs, layouts of related graphs, and the evolution of a graph. Archambault used difference maps in the context of dynamic graphs and grouped changes in hierarchies to reduce clutter [2], also showing that such maps help users better analyze differences between graphs [4]. Erten et al. describe three approaches for simultaneously visualizing a series of graphs with known correspondences [21]. Diehl and Görg used a foresighted layout to address offline dynamic graph drawing [18], and Beck et al. have investigated the aesthetics of dynamic graph visualization [7]. Other work has addressed "matched graphs" where correspondences are shown through the positioning of nodes [17] and simultaneous embeddings of planar graphs [9]. For large numbers of graphs, there are techniques that summarize the information in the graphs by finding features of graphs to present visually [23, 46, 11].

There has also been work to address similar problems in specific domains. The visualization of software evolution has also produced significant results including techniques for comparing dependency graphs [8], finding code differences across multiple versions using syntax trees [15], and locating areas of change over time [16]. Other work includes methods to highlight differences between two provenance graphs [6] and metabolic networks [10].

Summary graphs are constructed by aligning pairs of graphs such that substructures are preserved and related nodes are merged. Our algorithm is based on the similarity flooding work by Melnik et al. [37] and the analogy matching from Scheidegger et al. [42]. The formulation of the cost of a matching is based on Riesen and Bunke's approximation for graph edit distance [40]. Zeng et al. use a similar formulation for graph searching [51], and Heymans and Singh use another variant to compare metabolic pathways to generate phylogenetic trees [32]. There are also techniques for calculating node similarity (*e.g.,* [36, 43]) which seek to identify equivalent nodes rather than our goal of aligning entire graphs.

3 Definitions & Overview

A *graph* $G = (V, E)$ is a set of nodes V and set of edges E where each edge $e = (v_1, v_2) \in E$ connects two nodes $v_1, v_2 \in V$. A *labeled graph* is a graph plus labeling functions for nodes, $L_V : V \to L$, and edges, $L_E : E \to L$, where L is a finite set of labels. Many graphs that are analyzed are labeled graphs. For example, a molecule can be represented as a graph where the nodes are atoms, and each atom can be labeled with its element, distinguishing molecules that might have the same structure. With labels, we can better compare nodes and edges from different graphs, allowing a more concise representation of a collection of graphs.

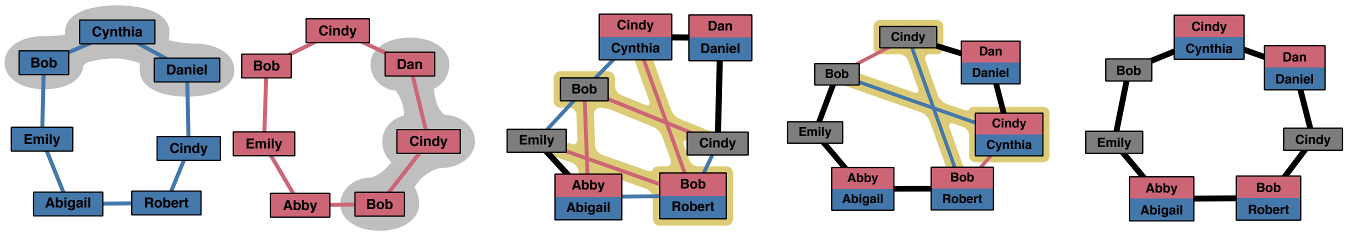
A *summary graph* of a collection of graphs is a single graph where each node (edge) represents at most one node (edge) from each of the graphs it summarizes. Thus, a node (edge) in the summary graph can be considered as a tuple of nodes (edges). For example, in the summary graph of four molecules shown in Figure 1, the "N–C" (nitrogen/carbon) node at the bottom represents an "N" node in the green molecule, and a "C" node in each of the blue, yellow, and red molecules. With related graphs, it is possible to have a large overlap, meaning each node or edge will correspond to many graphs in the collection. Thus, a summary graph can concisely express information about a set of graphs.

Formally, given a collection of graphs $\{G_1 = (V_1, E_1), \ldots, G_n = (V_n, E_n)\}$, a summary graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ is a graph where there exists a surjective map $M : \cup_{i \in [1,n]} V_i \to \mathscr{V}$ such that

- For each $e = (v_1, v_2) \in E_i$, there exists an edge $e^* = (v_1^*, v_2^*) \in \mathscr{E}$ such that $M(v_1) = v_1^*$ and $M(v_2) = v_2^*$.
- For each $(v_1^*, v_2^*) \in \mathscr{E}$, there exists some graph G_i with $v_1, v_2 \in V_i$ such that $M(v_1) = v_1^*$, $M(v_2) = v_2^*$, and $(v_1, v_2) \in E_i$.
- For $v_1 \in V_i, v_2 \in V_j$, $M(v_1) = M(v_2)$ implies $i \neq j$.

Then, a node $v^* \in \mathscr{V}$ represents a tuple $(v_{k_1}, \ldots, v_{k_n})$ where v_{k_ℓ} is a node in G_{k_ℓ}. Edges in the summary graph also correspond to tuples, but note that the connectivity defined in each input graph must be maintained in the summary graph.

To construct a summary of a collection of graphs, we need to identify which substructures are shared between input graphs and which differ. With two graphs, this is a problem of graph matching,



(a) Original Graphs (b) Node-Only Matching (c) Neighborhood Matching (d) Similarity Flooding

Figure 2: A comparison of graph-matching algorithms run on a pair of graphs (a); mismatched nodes and edges in (b) and (c) are highlighted. A node-only matching (b) errs because it cannot differentiate between similar pairs of nodes (*e.g.,* the Bob/Robert nodes). A matching that considers neighborhoods improves the result (c), but can still have issues when neighborhoods are similar (*e.g.,* the Cynthia/Cindy nodes). Similarity Flooding takes global structure into account, leading to an even better result (d).

finding a mapping between the nodes of the graphs that also induces matches between edges. A good matching pairs related nodes and edges while also keeping the total number of unpaired nodes and edges low. Such a matching can be used to construct a summary graph for two graphs: each matched pair of nodes (edges) and each unpaired node (edge) become a node (edge) in the summary graph. For larger collections, we can iterate, computing summary graphs of summary graphs. Thus, we will use hierarchical agglomeration to construct a single summary graph from a collection of graphs.

In the following sections, we describe how graph matchings are constructed, how pairwise graph matchings can be used to create a single summary graph, and how users can explore and edit an summary graph to better understand a collection of graphs.

4 Graph Matching

In order to build a single summary graph from a collection of labeled graphs, we need a method for merging graphs, and this depends on being able to match these graphs. The easiest case is pairwise graph matching, defining a relation between two graphs. Determining how graphs should be aligned depends on the desired type of matching. In an exact matching, as seen in graph theory, two graphs only match if there is a bijective mapping between nodes that also induces a matching between all edges so that the labels on matched nodes and edges are equal. In our approach, we are interested in inexact matchings where merged nodes may have different yet compatible labels or neighborhoods. For example, in chemistry, it is known that sodium and potassium appear in the same group in the periodic table and function similarly. There may also be cases where nodes appear in similar contexts and even if they are different, we may wish to match them to reduce the number of nodes.

Similarity between graphs can be defined in many ways, and in this work, we choose graph edit distance with user-defined cost functions. There are other definitions including, for example, the correspondence measure of trees discussed in [15]. However, computing the minimal graph edit distance can be slow and approximations do not necessarily produce meaningful matchings. Similarity flooding uses a Markov chain on a product of two graphs to diffuse similarities using connectivity information [37]. To generate both reasonable and flexible solutions for graph matchings, we use a solution of the assignment problem on a matrix composed of node edit costs.

4.1 Computing Graph Similarity

A *graph matching* for two graphs G_1 and G_2 is any map $h: V_1 \to V_2$. For simplicity, we will assume that matched nodes induce edge correspondences, but this could be changed to allow multi-edges if one wishes to also capture edge differences. Note that this definition allows a wide variety of matchings, ranging from an empty map to a full isomorphism. For these reasons, the quality of a matching is important, and we require a measure in order to compare matchings.

One measure of similarity, graph edit distance, is defined by the costs of edit operations that transform one graph to another. The possible edits, defined for both nodes and edges, are divided into three categories: substitutions, additions, and deletions. Each edit has an associated cost with the idea being that substituting a similar node n_1 for a node n_2 should cost less than deleting n_2 and adding n_1. For two graphs, their graph edit distance is the minimal sum of these costs over all valid sequences of edits. Then, the cost of a particular matching is the sum of the costs of the edits the matching induces. Specifically, mapped nodes are substitutions, unmapped nodes in V_1 are deletions, and unmapped nodes in V_2 are additions. A graph matching is *optimal* if has minimum cost.

4.2 Finding Graph Matchings

Finding a matching which induces the minimal graph edit cost is NP-Hard so an efficient solution for it is unlikely [51]. For small graphs, we might exhaustively check all possible matchings for one with minimal cost, but this will be very slow and quickly becomes infeasible. The complication here is that we wish to preserve both node similarity and connectivity. Thus, a good pairing of two nodes (a low cost node substitution) may induce poor pairings of edges (high cost edge additions and deletions). However, examining only the node costs (as if edge changes cost nothing) provides a starting heuristic for matching graphs, and adding information about the similarity of the neighborhoods of nodes allows us to improve these heuristics. We embed this into node costs because we can find the optimal solution for minimizing node costs in polynomial time.

Recall that edit distance defines costs for adding, deleting, and substituting nodes; each node must be associated with one and only one of these three operations—it cannot, for example, be both deleted and substituted for. We can write these costs in a matrix

$$\left(\begin{array}{cccc|cccc}
s_{00} & s_{01} & \cdots & s_{0n} & a_0 & \infty & \cdots & \infty \\
s_{10} & s_{11} & \cdots & s_{1n} & \infty & a_1 & \cdots & \infty \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
s_{m0} & s_{m1} & \cdots & s_{mn} & \infty & \infty & \cdots & a_m \\
\hline
d_0 & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\
\infty & d_1 & \cdots & \infty & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\infty & \infty & \cdots & d_n & 0 & 0 & \cdots & 0
\end{array}\right)$$

where the upper-left block contains all substitution costs (s_{ij}), the lower-left and upper-right blocks contain add/delete costs (a_k and d_ℓ), and the lower-right block is free. This matrix is designed so any valid sequence of edit operations corresponds to a selection of one entry from every row and one entry from every column [40]. We wish to find the sequence that minimizes the cost. Such constraints correspond exactly to the assignment problem [14], which has a polynomial-time solution via the Hungarian algorithm [35].

This provides a heuristic for computing a graph matching based solely on the node similarities, totally ignoring connectivity. While this solution may be efficient, it also produces poor matchings much of the time. Figure 2b shows an example where only node similarity was used to match the graphs, leading to a matching that is difficult to analyze. There are two possible avenues to leverage this heuristic: either use it to guide a search algorithm or try to encode connectivity information into the matrix.

Finding Low-Cost Matchings. Our goal is to find a matching with low cost. There are a number of search algorithms that serve to explore the space of all possible matchings. One of the most well-known search algorithms is A* which is a best-first search strategy that uses estimates of costs to guide exploration to an optimal solution [30]. Beam search uses similar estimates but unlike A*, it is a breadth-first search; at each step, each potential solution is extended by adding one new node substitution and calculating the cost of each new partial solution (both the known cost based on the edit distance costs and the remaining cost estimated from the node-only heuristic described above). Then, only the k partial solutions with the least cost are kept; the rest are discarded. At the end, we will have the lowest-cost solution among those that were not pruned, but note that because our node-based heuristic underestimates costs, it is possible that an optimal solution was pruned.

Using Neighborhood Information. We can also improve matchings by encoding connectivity and neighborhood information into the matrix our heuristic algorithm uses. Riesen et al. propose encoding the best case edge substitution, deletion, and addition costs into the node costs [40]. For example, a substitution $v_1 \to v_2$ means that our best case scenario is that all edges connecting to v_1 will match those edges connecting to v_2 at minimal cost. We can calculate the cost of this scenario by adding the costs of these substitutions to the costs of adding or deleting the remaining edges. This could also be extended to include matching the nodes on the other side of the edge. Figure 2c shows the result of including neighbor information in the heuristic; note that it still produces a sub-optimal solution because it only considers the immediate neighborhood.

Similarity Flooding. In order to better globally align graphs, we need to include more information about the full neighborhoods of each node into substitution costs. Similarity flooding [37] is an iterative process that propagates individual similarities between pairs of nodes across the entire graph. It works by using a direct product graph whose nodes are the possible pairs of nodes from the input graphs and whose edges indicate positive correlations between pairs of nodes. Unlike a Cartesian product graph, a direct product graph only includes an edge when both input graphs contain it.

The algorithm works by, at each step, calculating a new cost for each node in the product graph G based on its current cost and the costs of its neighbors. In this way, the cost of a given product node can eventually influence all of the other nodes it could be aligned with (some nodes will be incompatible). More formally, if we let π_k be the normalized vector of product node costs at the kth step, $A(G)$ be the adjacency matrix of the product graph, and $c(G)$ be the initial costs expanded as a vector, we can write the iteration as:

$$\pi_{k+1} = \alpha A(G)\pi_k + (1-\alpha)c(G)$$

where α is a parameter that controls the balance between diffusion and the initial costs [42]. This corresponds to the power method for eigenvector computation, and thus, this process will converge to π_∞ which can then be used as our improved node substitution costs.

With this solution, we can proceed to obtain a matching via the Hungarian algorithm as before. Because of the diffusion, the new matrix contains scores that better incorporate connectivity information. Figure 2d shows that this approach allows us to compute a better match than a node-only or neighborhood-based matching. Our tool allows the use of both beam search and similarity flooding to compute pairwise matchings.

5 Summary Graph Construction

A summary graph of two graphs is generated from a graph matching in a straightforward manner. If nodes are matched, we create a new compound node; if not, we create individual nodes representing the unmatched nodes. More formally, given a graph matching $h : V \to V'$ for graphs G_1, G_2, we can create a summary graph $\mathscr{G}$ by defining

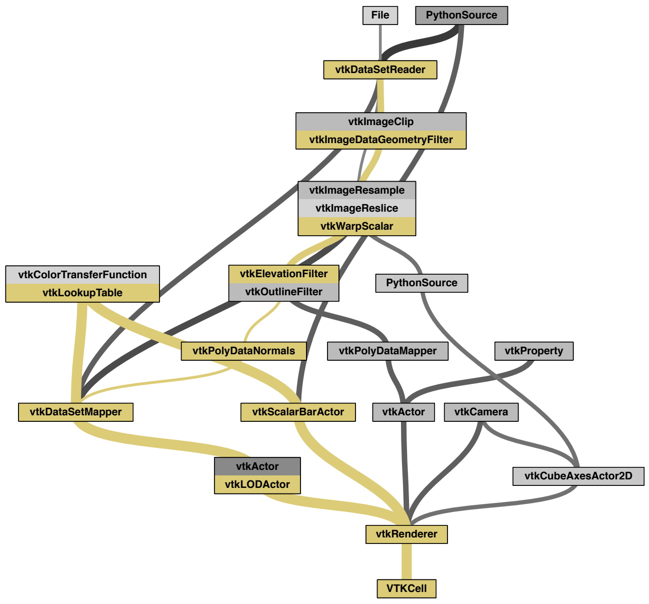


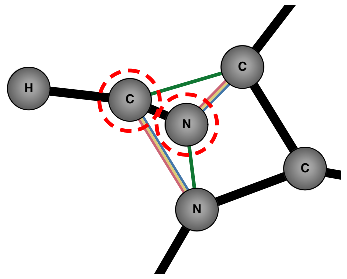
Figure 3: A summary of eight graphs representing visualization workflows generated by different students for a specific homework problem. A single student's work is highlighted in the context of the summary allowing it to be compared with the others as a whole.

the surjective map $M : V_1 \cup V_2 \to \mathscr{V}$ as

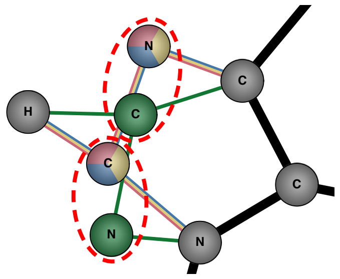
$$M(v) = \begin{cases} (v, h(v)) & v \in V_1 \text{ and } v \text{ is in the domain of } h \\ v & v \in V_1 \text{ when } v \text{ is not in the domain of } h \\ v & v \in V_2 \text{ and } v \text{ is not in the range of } h \end{cases}$$

Solutions to the assignment problem become NP-Hard in dimensions higher than two [14]. Thus, extending graph matching approaches to compute the best matching for n graphs becomes impractical. Instead, we use hierarchical agglomeration to build the summary graph using a series of pairwise matchings. At each step, we combine two graphs by computing their matching and build a summary graph according to that matching. Recall that hierarchical clustering requires similarities for each pair of graphs. Because computing matchings for all pairs is inefficient, we use heuristics based on feature vectors constructed from the node labels to order the agglomeration. From this ordering, we compute all matchings in the hierarchy to construct a summary graph. Note that each individual graph is recognizable in the final summary graph (see Figure 3). For more diverse graph collections, creating a single summary graph may be undesirable because the graphs are too dissimilar. In such cases, we can use clustering to identify similar subcollections and build multiple summary graphs.

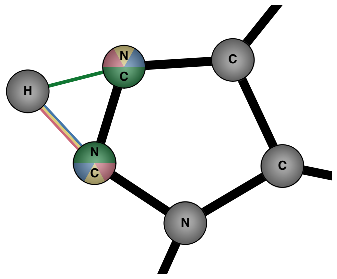
Each step of the construction is very similar to the two-graph construction, but higher levels require extensions to the cost functions. Specifically, we must determine the cost of substituting, deleting, or adding a compound node. Given two compound nodes v_i and v_j where $v_i = (v_{i_1}, \dots v_{i_m})$, $v_j = (v_{j_1}, \dots v_{j_n})$, and m need not equal n, we wish to define their substitution cost in terms of the costs of the simple nodes $\{v_{i_k}\}$ and $\{v_{j_\ell}\}$. Like the simple cost functions, we allow this computation to be domain-specific; for example, in chemistry a merged atom that already differentiates two molecules might be more likely to be merged with a third type of atom. By default, an overall average seems to give reasonable results as we are effectively determining the similarity of two sets of nodes, but the minimum cost might also be used. With these cost functions, we can compute the overall cost of a matching for each pair of graphs, regardless of whether they are already summary graphs.



(a) Original Graph



(b) Broken Nodes



(c) Joined Nodes

Figure 4: A piece of a molecular summary graph that shows how users can leverage edit operations to transform one summary into another via two break operations, (a) to (b), followed by two join operations, (b) to (c).

We can generate an order of node merges for each match based on their costs. We can make this ordering global by integrating the orders at each level. Because we only have costs based on the agglomeration tree, our ordering both depends on the order of merges as well as the individual costs. For example, we cannot place a merge of two compound nodes ahead of the merges of their underlying simple nodes. Thus, the global ordering is defined such that a merge m_1 is ordered before m_2 if m_2 depends on merge m_1 or m_1 has a higher cost than m_2. Later, we discuss how this ordering can be used when interacting with a summary graph.

6 Display and Interaction

A summary graph could be statically displayed as a normal labeled graph, but doing so would limit the amount of information a user could find. Instead, we wish to allow users to manipulate and explore the summary graph to discover similarities and differences. From colors to node labels to edge thickness, we seek to encode common information as well as unique features of the input graphs. In addition, we allow a user to explore relationships between subsets of input graphs as well as control the amount of summarization displayed. Finally, because there can be more than one effective summary graph for a set of input graphs (see Figure 7), we allow users to edit summary graphs via break and join operations.

Layout and Display. We use the dot and neato algorithms from the graphviz library [26] to layout directed and undirected graphs, respectively. These algorithms support some methods to guide layouts with position information so we can layout a summary graph in a similar way to one of the input graphs if desired. In addition, we can support modifications of the layouts as users change the amount of summarization shown or edit the summary by breaking or joining nodes. While these algorithms do not totally support dynamic graph layouts, we can preserve some structure as the graph changes, and we also use animation to help users follow the changes.

Recall that a node of a summary graph represents nodes from one or more input graphs. Because these input nodes may share the same label, we show each distinct label only once to reduce clutter. Labels are uniformly distributed in order to better show differences. We use color to show which graph(s) a given label belongs to; colors are also uniformly distributed behind the label. However, when a label is shared between many graphs, the number of colors can be distracting. In such cases, the node may be colored gray with its lightness representing the number of input nodes that share the label. Similarly, a summary edge encodes an edge from one or more input graphs. For edges, we use thickness or opacity to indicate how many graphs an edge appears in.

When there exist domain-specific techniques for visualizing a specific type of graph, it may be possible to extend that technique to display a summary. For example, a summary of molecules could be displayed using a ball-and-stick approach where balls have various colors according to which input graphs they represent. However, a summary graph could violate some constraints of the technique. For example, a summary of a set of planar graphs is not necessarily planar, and therefore any layout or rendering that depends on planarity would not be guaranteed to work for summary graphs.

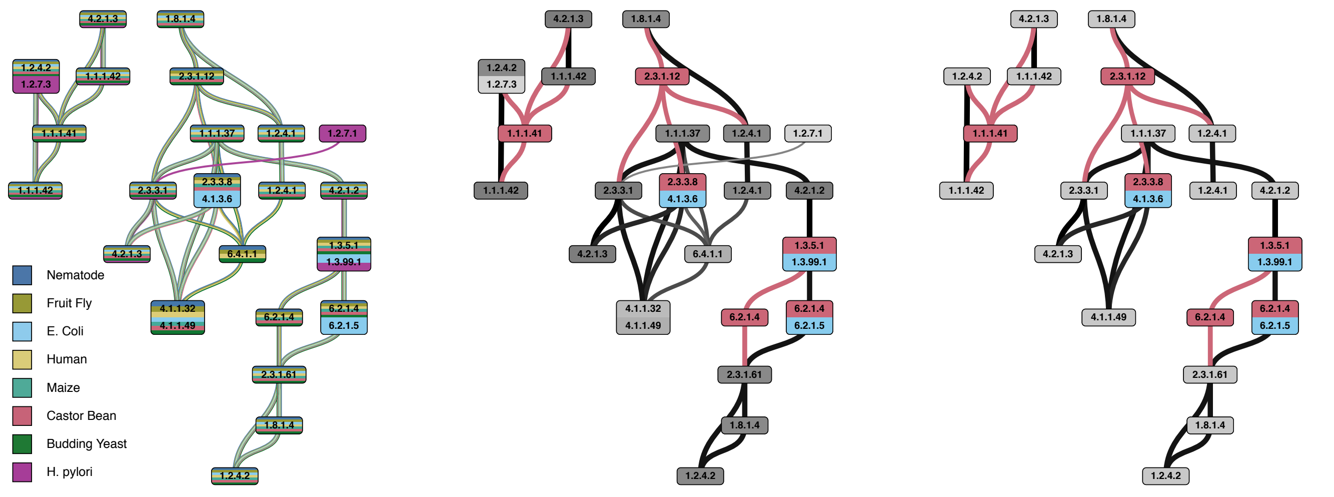
Edit Operations. As noted earlier, the initial summary graph result may not always grant a user the desired insight. This could be because too much was summarized or because summarized nodes and edges cluttered the layout. For example, a summary node that has edges connecting across the graph reduces the total number of nodes but may also have more edge crossings than if it were split into two or more nodes. Conversely, pairing nodes that do not belong to the same class may permit greater insight into structural similarities. For this reason, a summary graph should be *editable*; users should be able to explore and modify the visualization to gain better insight into the collection of graphs. We provide two operations that act on the nodes of the graph:

1. **Break.** Split a compound node by creating two new nodes that split the simple nodes represented.
2. **Join.** Join a set of nodes by creating a compound node that combines the simple nodes represented by each input.

Note that a user's edit operations can also be used to recompute the summary graph. For example, when a user breaks a compound node, it means that the two nodes should not be merged, and we can set the cost of this merge to the maximum and rerun the algorithm.

To break or join a node, a user selects the node(s) and chooses the appropriate operation (either via a global or contextual menu). One approach for breaking nodes is to use the ordered merge tree, breaking the node according to the last merge. However, when the node has multiple labels (it represents input nodes that do not all share the same label), it is more intuitive if it is split into two nodes with disjoint labels. Similarly, when there is only a single label, splitting the node based on the rarest edge that connects to that node allows a user to better untangle over-summarization. A join is more straightforward; the input nodes represented by the selected compound nodes are all added to a new single compound node. A join is not allowed when more than one node from the same input graph is involved. As mentioned earlier, we use animation when performing these operations to allow a user to see exactly where any new nodes are positioned in an updated layout. For breaks, new nodes are initially positioned at the compound node and moved (via linear interpolation) to new locations based on a layout of the new graph; joins are animated as the inverse of breaks.

Controlling Summarization. A summary graph is designed to compress the display of multiple graphs to a single graph that highlights similarities and differences. However, it is possible to have too much summarization. There may be times when separating nodes can unclutter a visualization. For example, two workflows that share similar overall structure but have different sequences of modules in a particular region could be shown with the sequences merged or unmerged. In some cases (*e.g.,* when the sequences are of different lengths), it may be easier to understand the differences when the modules are unmerged. For these and other cases, we allow users to control the amount of summarization displayed using a



(a) All Colors (b) Highlighting (c) Hiding Graphs

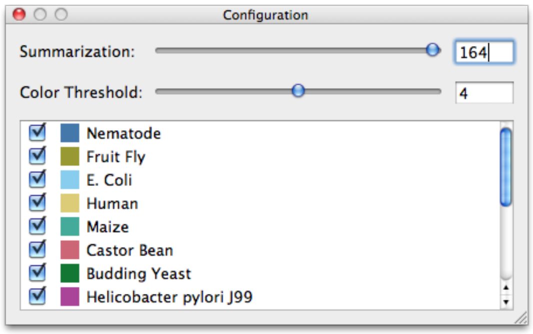
Figure 5: A summary graph of enzyme relation graphs from the citric acid cycle for eight organisms. Showing all colors (a) can be distracting. A user can interactively investigate a particular relationship, like that between E. coli and the castor bean, by either highlighting specific differences in the context of the summary (b) or hiding nodes and edges that do not exist in a selected subset (c).

linear slider. Each step up (down) the slider is a single join (break) of two existing nodes. See Figure 6 for an example of this control.

To construct the global order, we need the forest that corresponds to all merges involved in the summary graph construction. Each node in the forest results from a merge of two nodes, and each has a weight that corresponds to its substitution cost from the matching. A node is a leaf if it is a merge of two nodes from the input graphs. To compute the order, we use an iterative algorithm with a priority queue. The queue is seeded with all of the leaves of the tree and ordered by the similarity measure. At each step, the highest scoring node is removed from the queue, added to the final order, and if all siblings of the selected node have already been added to the final order, the parent node is inserted to the priority queue.

Selective Views and Color. Another feature of our interface is the ability for a user to examine specific input graphs by highlighting them in the context of the summary. A user can highlight a single graph to see exactly how it relates to all other graphs, or select two graphs to see the differences between exactly those two. Furthermore, a user can choose to hide graphs that are less similar to a set of others to better analyze a subset. In all of these operations, the layout of the graph remains the same, allowing the entire analysis to occur in the context of the summary graph.

Color is often used to differentiate categories, and in a summary graph, we use it to highlight specific differences or a selected input graph. As Figure 5a shows, using color to show every input graph causes an over-saturated visualization that is challenging to analyze. We allow users to change when nodes or edges are colored using a *color threshold* T; color is only shown when the node or edge occurs in fewer than T graphs. This helps highlight unique differences as shown in Figure 5b. Both the controls for selecting which input graphs are shown or highlighted as well as the color thresholds are available in the settings interface shown in Figure 6.


Configuration

Summarization: 164

Color Threshold: 4

☑ 🟦 Nematode
☑ 🟨 Fruit Fly
☑ 🟦 E. Coli
☑ 🟨 Human
☑ 🟩 Maize
☑ 🟥 Castor Bean
☑ 🟩 Budding Yeast
☑ 🟪 Helicobacter pylori J99

Figure 6: The interface allows users to control summarization, adjust the color threshold, and toggle the display of individual graphs.

7 Applications

Metabolic Pathways. In biology, the pathways that drive life by chemical processes are important components in our understanding of cells. During each process, compounds (metabolites) are processed into other compounds via different enzymes and substrates. Even for the same process (*e.g.,* glycolysis or the citric acid cycle), the components involved can vary across organisms. Understanding these differences can help in comparing organisms, especially with respect to their evolutionary history. Each pathway can be abstracted to only the orders of different enzymes to simplify this process [32]. Enzymes can be identified by their EC numbers, four-number tuples separated by dots, which are organized hierarchically [50]. Thus, the similarity between enzymes can be determined based on how many numbers match (*e.g.,* 1.2.3.4 and 1.2.3.5 are quite similar while 4.3.2.1 and 4.5.6.7 are close to completely different). In our examples, we used the Kyoto Encyclopedia of Genes and Genomes (KEGG) database [34] to obtain the pathway information. The summary graph for the Citric Acid Cycle, in Figure 5, shows that while complex organisms share most enzyme relationships, yeast and E. coli have some significant differences.

Visualization Pipelines. For programs that have a modular structure, it is often useful to visually examine this structure. Visualization pipelines are a good example of structured programs: they are defined as graphs whose nodes correspond to computational modules and edges represent how data flows between modules. The ability to construct summaries for workflow graphs is important for different applications. When exploring a shared collection of workflows (e.g., `http//www.crowdlabs.org`), users can pose queries to identify workflows with a particular substructure, or they can cluster together similar workflows [41]. Summarizing the query results or derived clusters can help users better understand a set of approaches used to solve a particular problem or how a technique is used in different problems. For example, one can examine different contexts where volume rendering is used, and even contrast these against other techniques such as isosurface extraction.

Another important application in this context is teaching. From the summary of a set of pipelines that were produced by students as solutions to a visualization assignment, we can obtain a "consen-

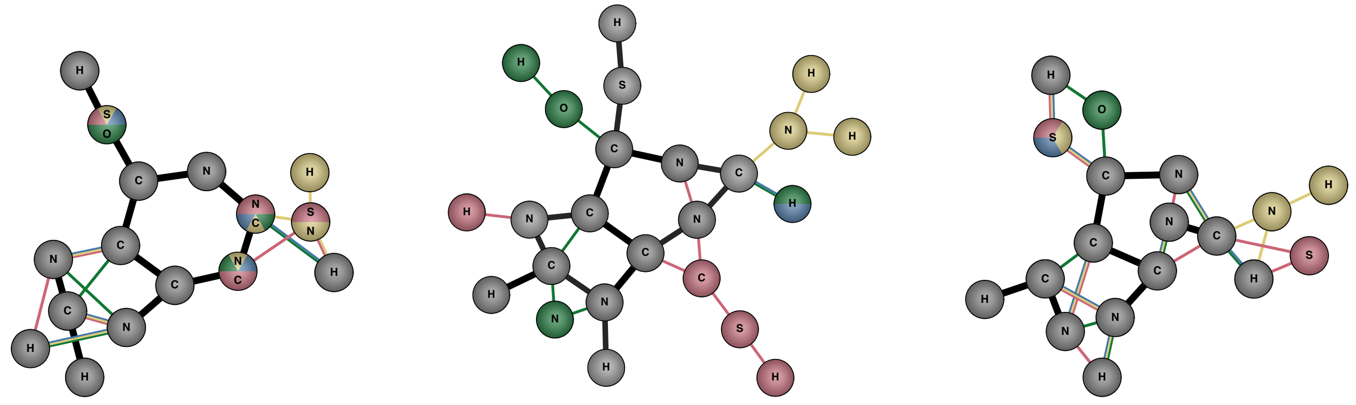


Figure 7: One of the questions in the user study asked participants to modify a given summary (left) constructed using the same graphs as in Figure 1 to create an informative visualization. Two results, (middle) and (right), show how user preferences vary.

sus" solution as well as identify the unique differences in individual solutions. Figure 3 shows a summary graph for eight different visualization workflows constructed in VisTrails [45] using VTK [48] modules. Most students followed the same core structure, but the filters and additional customizations used varied.

8 User Study

In order to evaluate the interactive features of our summary graph visualization tool, we conducted a user study with a set of eight tasks that ranged from determining differences in a graph collection to creating a summary from scratch. Our fifteen participants came from a pool of computer science students and researchers, most of whom described themselves as only familiar with graph visualization techniques on a basic level. For all tasks, we collected both timing information and, depending on the task, information about selected nodes or edges, the summarization hierarchy, and settings like the values of threshold levels and which graphs are highlighted. In order to minimize the amount of learning for each task, all graphs contained 10–30 nodes and were obtained from molecules, although users were not expected to know anything about this domain. At the end of the study, we asked the users to provide feedback about the software including questions about particular features like color and the editing tools. Overall, users were positive about the tool and thought it represented a useful direction for discovering relationships between similar graphs.

Tasks. The eight tasks were organized into four pairs. The first two tasks asked participants to identify differences between two graphs both using a summary and without. In order to evaluate threshold values, the second pair of tasks had users set color and summarization thresholds to their liking. The third group of tasks was designed to force participants to use the highlighting and threshold features to answer questions about differences between graphs in a summary. The final pair of tasks was creative, asking users to build or modify summaries to obtain an aesthetically pleasing visualization.

Results. The first pair of tasks was set up so one half of the participants answered a question about differences between graphs without a summary and the other half had the summary. This was flipped for a second pair of graphs. For the first graph, users were about 20% faster using the summary and for the second graph, 40%, taking 96.41 seconds to identify differences. One of the pairs had smaller graphs which we believe accounts for the difference. However, the speed-up is still smaller than expected, which may be due to the fact that these were the first questions after the warm-up exercise. Also interesting was that users preferred to see more color in a static summary; the average color threshold was set to 5.4 for a summary of eight graphs. The preferred summarization level for a summary that was "over-compact" was with nine merges removed, mostly to remove edge crossings from the displayed layout.

Results from the third pair of tasks were inconclusive, with the average times to complete the tasks ranging from one to two minutes. Users did utilize the ability to highlight and show subsets of graphs, but we were not able to measure if these features made their efforts easier. The results from the final pair of task showed that a "good" summary graph varies depending on the person viewing it. Figure 7 shows two representative summaries that users created by starting from the same given summary; one user preferred a planar layout while another valued a more compact representation.

User Opinion. Overall, users had a positive impression of the tool. There were a wide range of positive comments, and a common theme was that interactivity was useful in discovering relationships between graphs. On the other hand, layout and colors caused confusion at times. When the summary or summarization threshold changes, the layout must also be updated, and some nodes may be moved more than desired due to the current layout algorithm. Because color is used to distinguish which graphs are represented in a given node or edge, when the number of graphs was large, it was difficult for some users to determine which graphs were represented by a given edge. Highlighting and filtering should help with some of these issues and were not emphasized in the provided documentation. Suggested improvements included adding zoom capabilities and node tooltips that show the input nodes represented.

9 Conclusion & Future Work

Visual summaries for graph collections provide a compact representation for the similarities and differences between an entire set of graphs. With the ability to analyze and edit a summary graph, users can both explore and configure summaries to better understand relationships between the underlying input graphs.

Our approach and examples show results for relatively small collections of less than 30-node graphs. An important direction in graph visualization is scalability, effectively handling both large graphs as well as larger collections of graphs. One roadblock to scaling the number of graphs is that using color to distinguish individual graphs for collections with hundreds of graphs will be ineffective. For large graphs, it is often necessary to use single graph summarization to generate legible representations that fit on screen. We plan to investigate methods both for calculating and visualizing summary graphs in these cases.

Another direction that we would like to explore further is to more effectively exploit domain-specific and user-specified information. While costs for substituting nodes and edges can be generally defined based on label differences, the power gained from knowing when nodes are related allows us to calculate better summary graphs. Furthermore, when users edit graphs by identifying specific nodes that should be broken or joined in the summary, this information could be used to help refine similarity calculations. We might even be able to recalculate the summary based on a small set of user-defined matches as has been used in image registration [13].

References

[1] E. N. Adams, III. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21(4):390–397, 1972.

[2] D. Archambault. Structural differences between two graphs through hierarchies. In *Proc. of Graphics Interface 2009*, pages 87–94. Canadian Information Processing Society, 2009.

[3] D. Archambault, T. Munzner, and D. Auber. Topolayout: Multilevel graph layout by topological features. *IEEE Trans. Vis. Comput. Graph.*, 13(2):305–317, 2007.

[4] D. Archambault, H. C. Purchase, and B. Pinaud. Difference map readability for dynamic graphs. In *Proc. 18th Int'l Conf. Graph Drawing*, pages 50–61. Springer-Verlag, 2011.

[5] M. Balzer and O. Deussen. Level-of-detail visualization of clustered graph layouts. In *Proc. Asia-Pacific Symp. Vis.*, pages 133–140, 2007.

[6] Z. Bao, S. Cohen-Boulakia, S. Davidson, and P. Girard. PDiffView: Viewing the difference in provenance of workflow results. *PVLDB*, 2(2):1638–1641, 2009.

[7] F. Beck, M. Burch, and S. Diehl. Towards an aesthetic dimensions framework for dynamic graph visualisations. In *Proc. IEEE InfoVis 2009*, pages 592–597, 2009.

[8] F. Beck and S. Diehl. Visual comparison of software architectures. In *Proc. 5th Int'l Symp. Software Vis.*, pages 183–192. ACM, 2010.

[9] T. Bläsius, S. G. Kobourov, and I. Rutter. Simultaneous embedding of planar graphs. *CoRR*, abs/1204.5853, 2012.

[10] R. Bourqui and F. Jourdan. Revealing subnetwork roles using contextual visualization: Comparison of metabolic networks. In *Proc. IEEE InfoVis 2008*, pages 638–643, 2008.

[11] U. Brandes, J. Lerner, M. Lubbers, C. McCarty, and J. Molina. Visual statistics for collections of clustered graphs. In *IEEE Pacific Visualization Symposium, 2008*, pages 47–54, 2008.

[12] S. Bremm, T. von Landesberger, M. Hess, T. Schreck, P. Weil, and K. Hamacher. Interactive visual comparison of multiple trees. In *Proc. IEEE VAST 2011*, pages 31–40, 2011.

[13] L. G. Brown. A survey of image registration techniques. *ACM Comput. Surv.*, 24(4):325–376, 1992.

[14] R. E. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. SIAM, 2009.

[15] F. Chevalier, D. Auber, and A. Telea. Structural analysis and visualization of C++ code evolution using syntax trees. In *9th Int'l Workshop on Principles of Software Evolution*, pages 90–97, 2007.

[16] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *Proc. ACM Symp. on Software Vis.*, pages 77–86. ACM, 2003.

[17] E. Di Giacomo, W. Didimo, M. van Kreveld, G. Liotta, and B. Speckmann. Matched drawings of planar graphs. In *Proc. 15th Int'l Conf. Graph Drawing*, pages 183–194. Springer-Verlag, 2008.

[18] S. Diehl and C. Görg. Graphs, they are changing. In *Proc. 10th Int'l Symp. Graph Drawing*, pages 23–30. Springer-Verlag, 2002.

[19] P. Eades and Q.-W. Feng. Multilevel visualization of clustered graphs. In S. North, editor, *Graph Drawing*, volume 1190 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 1997.

[20] N. Elmqvist and J.-D. Fekete. Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Trans. Vis. Comput. Graph.*, 16:439–454, 2010.

[21] C. Erten, S. G. Kobourov, V. Le, and A. Navabi. Simultaneous graph drawing: Layout algorithms and visualization schemes. *Journal of Graph Algortihms and Applications*, 9(1):165–182, 2005.

[22] J. Freire, C. Silva, S. Callahan, E. Santos, C. Scheidegger, and H. Vo. Managing rapidly-evolving scientific workflows. In *Int'l Provenance and Annotation Workshop*, LNCS 4145, pages 10–18. Springer, 2006.

[23] M. Freire, C. Plaisant, B. Shneiderman, and J. Golbeck. Manynets: an interface for multiple network analysis and visualization. In *Proc. SIGCHI 2010*, pages 213–222. ACM, 2010.

[24] G. W. Furnas and J. Zacks. Multitrees: enriching and reusing hierarchical structure. In *Proc. SIGCHI 1994*, pages 330–336, 1994.

[25] E. Gansner, Y. Koren, and S. North. Topological fisheye views for visualizing large graphs. *IEEE Trans. Vis. Comput. Graph.*, 11(4):457–468, 2005.

[26] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Softw. Pract. Exper.*, 30:1203–1233, 2000.

[27] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proc. of ACM Symp. on Theory of Computing*, pages 47–63, New York, NY, USA, 1974. ACM.

[28] M. Graham and J. Kennedy. Exploring multiple trees through DAG representations. *IEEE Trans. Vis. Comput. Graph.*, 13:1294–1301, 2007.

[29] M. Graham and J. B. Kennedy. A survey of multiple tree visualisation. *Information Visualization*, 9(4):235–252, 2010.

[30] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. and Cybern.*, 4(2):100–107, 1968.

[31] I. Herman, G. Melancon, and M. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Trans. Vis. Comput. Graph.*, 6(1):24–43, 2000.

[32] M. Heymans and A. K. Singh. Deriving phylogenetic trees from the similarity analysis of metabolic pathways. In *ISMB (Supplement of Bioinformatics)*, pages 138–146, 2003.

[33] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Trans. Vis. Comput. Graph.*, 12:741–748, 2006.

[34] M. Kanehisa and S. Goto. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research*, 28(1):27–30, 2000.

[35] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[36] E. A. Leicht, P. Holme, and M. E. J. Newman. Vertex similarity in networks. *Phys. Rev. E*, 73:026120, 2006.

[37] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. 18th Int'l Conf. Data Eng.*, pages 117–128, 2002.

[38] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou. Treejuxtaposer: scalable tree comparison using focus+context with guaranteed visibility. *ACM Trans. Graph.*, 22:453–462, 2003.

[39] C. Plaisant, J.-D. Fekete, and G. Grinstein. Promoting insight-based evaluation of visualizations: From contest to benchmark repository. *IEEE Trans. Vis. Comput. Graph.*, 14(1):120–134, 2008.

[40] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.*, 27:950–959, 2009.

[41] E. Santos, L. Lins, J. P. Ahrens, J. Freire, and C. T. Silva. A first study on clustering collections of workflow graphs. In *IPAW*, pages 160–173, 2008.

[42] C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. Querying and creating visualizations by analogy. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1560–1567, 2007.

[43] K. Thiel and M. R. Berthold. Node similarities from spreading activation. *IEEE Int'l Conf. on Data Mining*, pages 1085–1090, 2010.

[44] S. Vilar, R. Harpaz, E. Uriarte, L. Santana, R. Rabadan, and C. Friedman. Drug–drug interaction through molecular structure similarity analysis. *J. Am. Med. Inform. Assoc.*, 2012.

[45] VisTrails. `http://www.vistrails.org`.

[46] T. von Landesberger, M. Görner, and T. Schreck. Visual analysis of graphs with multiple connected components. In *Proc. IEEE VAST 2009*, pages 155–162, 2009.

[47] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, and D. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6):1719–1749, 2011.

[48] VTK. `http://www.vtk.org`.

[49] M. Wattenberg. Visual exploration of multivariate graphs. In *Proc. SIGCHI 2006*, pages 811–819. ACM, 2006.

[50] E. C. Webb. *Enzyme nomenclature 1992. Recommendations of the Nomenclature Committee of the Int'l Union of Biochemistry and Molecular Biology on the Nomenclature and Classification of Enzymes*. Academic Press, 1992.

[51] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: on approximating graph edit distance. *PVLDB*, 2:25–36, 2009.