CommunityDiff: Visualizing Community Clustering Algorithms

SRAYAN DATTA and EYTAN ADAR, University of Michigan

Community detection is an oft-used analytical function of network analysis but can be a black art to apply in practice. Grouping of related nodes is important for identifying patterns in network datasets but also notoriously sensitive to input data and algorithm selection. This is further complicated by the fact that, depending on domain and use case, the ground truth knowledge of the end-user can vary from none to complete. In this work, we present CommunityDiff, an interactive visualization system that combines visualization and active learning (AL) to support the end-user's analytical process. As the end-user interacts with the system, a continuous refinement process updates both the community labels and visualizations. CommunityDiff features a mechanism for visualizing *ensemble spaces*, weighted combinations of algorithm output, that can identify patterns, commonalities, and differences among multiple community detection algorithms. Among other features, CommunityDiff introduces an AL mechanism that visually indicates uncertainty about community labels to focus end-user attention and supporting end-user control that ranges from explicitly indicating the number of expected communities to merging and splitting communities. Based on this end-user input, CommunityDiff dynamically recalculates communities. We demonstrate the viability of our through a study of speed of end-user convergence on satisfactory community labels. As part of building CommunityDiff, we describe a design process that can be adapted to other Interactive Machine Learning applications.

CCS Concepts: • **Information systems** → **Clustering**; • **Human-centered computing** → **User interface management systems**; **Visual analytics**; • **Computing methodologies** → *Ensemble methods*;

Additional Key Words and Phrases: Community detection, interactive machine learning, visualization

ACM Reference format:
Srayan Datta and Eytan Adar. 2018. CommunityDiff: Visualizing Community Clustering Algorithms. *ACM Trans. Knowl. Discov. Data.* 12, 1, Article 11 (January 2018), 34 pages.
https://doi.org/10.1145/3047009

1 INTRODUCTION

Community structure often provides useful insights about the underlying network. For example, communities in social network represent groups of people who share an interest, location, or other semantically meaningful attributes (e.g., were all "friends in high school") [38, 82]. In biological and brain networks, communities capture functional groups [16, 41]. Many other non-network problems in machine learning and data mining are recast as networks so that the network analysis

This work is partially supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center contract number D11PC20155.
Authors' addresses: S. Datta and E. Adar, University of Michigan, 105 South State Street, Ann Arbor, MI 48109; emails: {srayand, eadar}@umich.edu.

ACM 1556-4681/2018/01-ART11 $15.00
https://doi.org/10.1145/3047009

tools, including community detection, can be used. For example, transforming text to graphs in NLP [61] or user-item recommender models [44] allows us to use community detection to solve problems as diverse as document summarization or movie recommendations.

In practice, there are many community detection algorithms with very different properties: some work better at large or small scales, many use specific optimization functions (e.g., modularity), others contain randomization, some work better with weighted networks, and so on. The output *partitions* (i.e., *communities* or *community labels*) can be significantly different even when the input is the same. Because the space of possible algorithms and parameters is vast, selecting a satisfactory approach is difficult (let alone optimal). Even for algorithm experts, the selection can be a black art. For domain experts (e.g., social networks, biology) the choices can be overwhelming.

We are specifically interested in supporting domain experts who would like to partition a network into "communities" where labels are important. For example, the manual task of labeling blogs based on political party [1] or sub-communities in an ego-net [40]. Domain experts already have access to tools to support community detection. While they may be familiar with network analysis, the tools that are available are often generic and do not readily support community detection beyond executing the algorithms. Though we return to the issue of overlapping communities below, in many of these domains, "hard" partitions are desirable. This kind of community partitioning is one of the main functions of most network analysis packages used today (e.g., Pajek [9] and UCINET [14] in the social sciences; Cytoscape in biology [77]; and more general tools such as Gephi [8], GUESS [2], and NodeXL [39], etc.) or libraries such as *igraph* [23]. Each software or library contains multiple implementations and it is often difficult for an end-user to determine which is appropriate. Given the specific goal of "accurate" and well-understood partitions, the end user would like to be provided with a viable automated starting point and a facility to understand differences and quickly "fix" labeling mistakes with as few interactions as possible.

One approach to dealing with too many choices is simply not pick any specific one and to combine algorithm output. This is one of the motivations behind *ensemble techniques* [56, 73]. Network scientists have devised ways to apply this technique to community detection [24, 50]. Ensembles can be treated as weighted combinations of different algorithms or parameters: each algorithm "votes" on the output and their votes are combined in some way. The hope is that the ensemble will produce something closer to the "correct" answer. However, even with ensemble techniques achieving perfect clustering, accuracy (according to some ground truth) is often impossible. This may be due to the presence of noisy or ambiguous data (missing edges, incorrect weights, extra edges, etc.) or the assumptions of community detection algorithms (e.g., that nodes in the same community are connected).

It is here where domain knowledge is critical to make corrections to algorithm output. Domain knowledge can include anything from knowing how many communities should exist or knowing which nodes should (or should not) be in the same community. In practice, however, the "completeness" of this knowledge can vary greatly. For example, the expert may only know that there are five communities but not what should be in them, or the expert may only be able to point at only subset of pairs of nodes that belong in the same community (e.g., experimental evidence in a biological network may only provide clear functional groups for some of the data). Ideally, an analytical tool would allow the end-user to: determine that the "facts" they know are captured in the current partition (or not); make corrections to bring the partition in line with known facts; and make decisions on those cases where there is no background knowledge. The tool we describe here combines both sensemaking and data-wrangling. This is in the sense that the high level objective is to create "clean" groupings (the wrangling part) that requires understanding the network in various ways (the sensemaking part). The particular activities vary depending on the prior knowledge of the end-user but both are entwined and both should be supported.

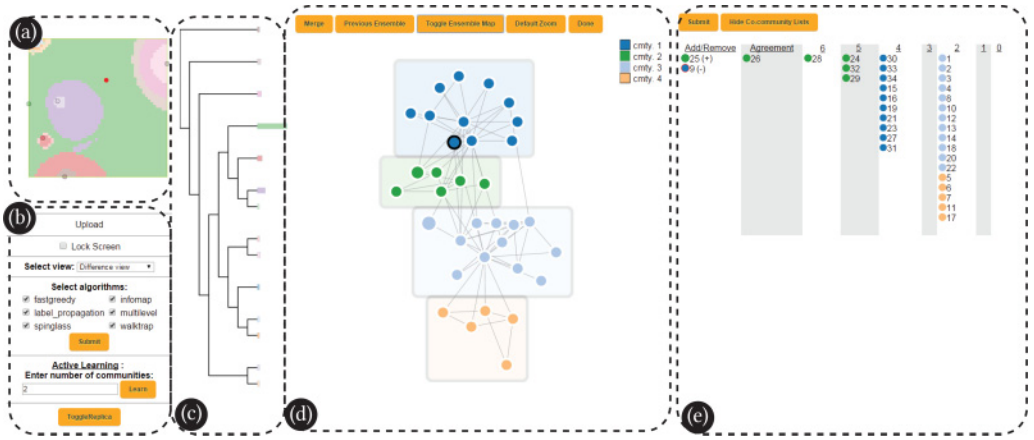

Fig. 1. CommunityDiff: (a) The *Ensemble Space Heatmap* showing different algorithms as dots and different ensembles achieved using these algorithms. (b) A panel showing different features and controls for the *Ensemble Space Heatmap* and the active learning panel. (c) A dendrogram showing relations among and proportions of different ensemble outputs. (d) The network diagram. (e) The co-community lists. Note that this view was used in evaluation and some advanced features were hidden (e.g., downloading the network).

Distilling this analytical pipeline, we can abstractly view our workflow as being comprised of two main "phases" (1) finding the best initial partitioning of the data (the best starting communities) and then (2) making the necessary corrections (with or without the software's help). From the perspective of the interface, this requires supporting the end-users' decision making about whether a partition is "good" and then making decisions about corrections and refinements.

CommunityDiff (see Figure 1) was designed with this pipeline in mind. The tool supports interactive analysis of community structures by coupling a novel visual analytics environment and decision-making tools[1]. CommunityDiff supports two main task types. The first is rapidly labeling all nodes in the network when ground truth is known (e.g., quickly partitioning handwriting samples by digit type or classifying the end-user's own social network egonet). The second mode is when the knowledge is incomplete and the end-user must compare different partitions in making decisions. Both tasks are supported by a visualization of *ensemble spaces*—the output of individual algorithms as well as mixtures. Through this view, the end-user can rapidly compare different ensembles and quickly resolve on partitions that are close to their final objective. Additionally, we introduce a second set of tools that help the end-user judge between alternative labels for specific nodes. Finally, a novel AL framework ensures that end-user actions can be integrated into the machine learning process to constantly improve results. CommunityDiff 'signals' to the end-user which labels would lead to better classification (visually, by making certain nodes more salient). Any operation done by the end-user—ranging from moving nodes between groups, specifying the number of clusters, excluding nodes from group membership, and many others—is taken into account as a form of supervision. Decisions are supported by focusing on uncertainty of community labels. For example, nodes with uncertain labels are larger in size and there are explicit functions to show agreement of different algorithms for nodes in the same cluster.

We demonstrate, through both automated and lab experiments, that CommunityDiff allows for rapidly constructed, high confidence, and accurate community labels. Currently, we only focus

[1]A brief video demonstrating the system is available at https://youtu.be/KNdQqWXTT8w.

on disjoint community detection rather than the overlapping variant as overlapping community detection algorithms are generally poor performers. In fact, discrete community detection often performs better than custom overlapping-community algorithms for overlapping community detection benchmarks [70]. For this reason, use of overlapping community detection algorithms is far less common compared to their hard-partition counterparts. We find that the most common social network analysis tools focus on hard partitions. While abstractly CommunityDiff can model overlapping communities (e.g., the overlap of two communities A and B can be modeled as a third community C that is the intersection of the two) and hierarchical communities (e.g., a community can be further partitioned into sub-communities), the current visualizations of CommunityDiff are not optimized for these variants.

The community-detection task we describe is an instance of a broader set of analytical pipelines. In many other Data Mining and Machine Learning tasks there are numerous alternative implementations (clustering algorithms, classification systems, etc.). With any of these, the domain expert must compare different models or solutions and make corrections in order to arrive at a satisfactory answer. In addition to demonstrating the effectiveness of CommunityDiff for the particular community-detection problem, we capture both design guidelines and visualization techniques that can be directly applied to other Interactive Machine Learning (IML) systems.

In this article, we contribute CommunityDiff, an interactive community detection analysis tool that allows end-users to easily compare, combine, and modify communities generated by multiple algorithms. We demonstrate how the idea of "ensemble-spaces" and specific heatmap-based representations can allow the end-user to identify common communities, where there is disagreement between algorithms, and to easily correct errors in labels. Along with other visual elements (e.g., conventional node-link diagrams, dendrograms and a novel co-community list), CommunityDiff also integrates active learning (AL) to visually suggest those nodes that should be labeled and which readily adapts the community structures based on end-user feedback. CommunityDiff is intended to support a common task for IML systems, where the human must interpret the output of the algorithm and make corrections.

2 RELATED WORK

Related work to CommunityDiff falls broadly into two categories: interactive features to support humans as part of the machine learning 'use' pipeline; and the algorithm side that produces better results based on human intervention.

2.1 Interactive Machine Learning

There are many visual techniques to analyze the output of machine learning and data mining algorithms. Simple scatter plots and histograms (for understanding data distributions) or precision/recall, ROC curves, or confusion matrices (for evaluating output) are a common representation available to developers in most platforms [11, 26, 60, 85]. To improve the designer's ability to debug different pipeline components a number of specific visualization techniques have been developed to provide an enhanced view at different outputs. Examples include visual model comparison [65, 78] and model interpretation [10, 19, 52, 62, 83, 89]. These range from generic [83, 89] to model and data specific (e.g., graphs [78] or text [62]), to algorithm-specific (e.g., Bayesian text classification [10] and SVMs [19]). Often these focus on static representations or basic comparisons (e.g., ROC or precision/recall) at the end of the analysis pipeline.

A good example of general visual model analysis is LoVis [89], which compares different linear models on any kind of data visually to identify local patterns in the data. In graph specific visual model comparison, Sharara et al. [78] introduced G-Pare, a visual analytics tool, which compares output of different machine learning algorithms on networks and provides both a global view of

algorithm outputs and difference of algorithms on specific subsets of nodes. This is similar in intent to our ability CommunityDiff's functions, comparing outputs of different community detection algorithm on the whole graph or a specific subset of nodes, though in our case we emphasize the use of ensembles and not simply comparisons. For algorithm-specific models, Becker et al. [10] provides a visualization of naive Bayes classifier outputs, specifically focusing on probabilistic importance of each feature and importance of each feature in a specific example. However, these systems often focus solely on comparison and not the interactive corrections.

Recognizing that by adding interactivity, visualizations could also support better analytical pipelines, researchers have turned to IML [29, 31, 69]. Through interaction with visualizations and other means of "dialog" (e.g., questions to the end-user), IML systems can obtain new training examples for classifications [32, 33], refine features [15, 58], identify trade-offs the designer is willing to make [46], modify individual algorithm behavior (e.g., Hidden Markov Models [25] or classifiers [7], and create effective ensembles [47, 81]. Such solutions are often an extension to visual workflow designs such as the Wekinator [32] extension to Weka [85] or plugins for Orange [26] or KNIME [11].

IML systems often focus on improving training data. For example CueFlik by Fogarty et al. [33] allows end-users in an image search engine to create their own rules for image search and improve upon searched image ranking in a personalized AL framework. Other IML system, such SmartStripes [58], focuses on interactive feature selection that allows user to explore dependency between different feature and entity subsets. Ensemble classifier algorithms can also interactively benefit from human input as demonstrated by Kapoor et al. [47] (with the aid of misclassification cost visualization for a multi-class ensemble classifier). These approaches are often singular in focus and reinforce traditional "black box" models that do not support the different modalities under which the end-user can operate (labeling, exploring, comparing, etc.). Our contention is that even simple analytical pipelines (e.g., generating community labels or clustering) require multiple modes of operation and multiple steps. To be effective, systems need to support different types of analytical work which requires varied, but integrated, visualization components. Through CommunityDiff we attempt to provide such a solution for domain-experts by building support for different steps through different visual elements.

A final area of related work is in intelligent data wrangling [37, 45]. Generating and correcting community labels on a graph can be viewed as a form of data wrangling. The end-user is "fixing" and validating some element of the data, in this case labels. Though data wrangling work is most often focused on more standard columnar data (e.g., [37]) work in the field has moved to incorporate visualizations, AL, and mixed-initiative frameworks. In this way, CommunityDiff can be situated in this broad space.

2.1.1 The Human Side of ML. The IML community has developed a number of design guidelines for eliciting feedback from human participants [5]. Horvitz [42] provided some key guidelines (for example, considering uncertainty about and user's goal and employing "dialog" to resolve key uncertainties) for designing mixed-initiative user interfaces that allows efficient communication between humans and the user interfaces. Our goal is to leverage these lessons as a starting point for our interactions. For example, Cakmak et al. [18] found that individuals providing feedback dislike acting as oracles for active learners (e.g., answering a constant stream of tedious questions) and Stumpf et al. [80] found that end-users wanted variety in the feedback they provided (e.g., modifying features, weights). Others have focused on the necessary level of interpretability in representations [49]. In part inspired by this work, we devise a set of guidelines specifically intended to guide the construction of effective visual analytics solutions.

2.2 Ensembles and Community Detection

CommunityDiff functions, in part, by allowing end-users to build ensembles of community-detection algorithm outputs as these often represent a "better" solution. Lancichinetti et al. [50] showed that use of consensus clustering (a kind of ensemble technique that uses voting) can boost accuracy and stability of a community detection algorithm. In this method, a single algorithm is run over the input network a number of times. The edge-weights of this network are modified using the results of different runs. The original algorithm is run over the modified network again. This works well in case of fast, but unstable, algorithms such as Fastgreedy [63]. Dahlin et al. [24] used a node-based fusion of communities algorithm which applies hierarchical clustering of nodes with a special linkage rule. Burgess et al. [17] create ensembles by generating variations of the same graph using randomized link-prediction and treating each run of the community detection algorithm as input to an ensemble.

Our idea of creating an ensemble space for community detection is very much inspired by the system described by Grimmer and King [36]. Their system focused on generating ensembles of multi-dimensional clustering algorithms. A metric space of these approaches allowed them to identify "holes" in this space and to generate ensembles that proved to be effective in finding the correct classification. We expand on this visualization to integrate additional "heat-maps" that capture differences between algorithms, key community-structure metrics (e.g., modularity). This ensemble visualization is used in combination with a number of other visual and algorithmic elements to support the end-user's labeling and partitioning tasks.

2.3 Active Learning

AL has seen a number of advancements in recent years from the algorithmic perspective [76] though it is rare for the focus to be on graph structures. More critically, very little attention has been paid to the interplay between visualization and AL. From the algorithmic perspective a notable exception is Leng et al. [54] who used a label-propagation based approach to incorporate AL in community detection. Macskassy et al. [57] used graph features for AL in networked datasets which can be viewed as a form of community detection in networked data. Within the HCI context, Amershi et al. [6] demonstrated the use of an AL scheme for grouping individuals by using node properties such as "age" or "place of birth" (rather than network structure). The Apolo system [20] utilized AL to help end-users collect and label related papers. The task was framed around an iterative process of collecting and labeling a small set of papers which could then focus on the "expansion" of this set.

3 THE TROUBLE WITH COMMUNITY DETECTION

Before describing the design of CommunityDiff in detail it is worth considering why interactive tools might offer an advantage when performing community detection. To understand why, we consider *how* community detection algorithms work, and therefore, fail. Community detection is dependent on structure and metadata (on nodes or edges). If edges were only to exist between nodes that were in the same community, or nodes contained explicit metadata that identified which community they belong to (e.g., a node identified as "group 1"), the solution would be obvious. Because networks are never this well structured, community detection algorithms attempt to find communities by optimizing on some metric (e.g., modularity or being part of a k-core [82]). Because networks are noisy and community detection algorithms can at best utilize an approximation of "optimality" (based on their metrics), there can be no guarantee of a "correct" output.

This should be unsurprising as community detection is a form of clustering. Specific studies on networks have shown that a "perfect and general" community detection algorithm is likely

out of reach either because the networks are noisy [17] (i.e., sampled nodes and edges do not reflect reality) or the algorithms make assumptions about structure [28] (e.g., overly focusing on metrics such as modularity) or metadata [66] (e.g., that metadata reflects ground truth). The consequence of these assumptions is that fully-automated community detection, even in simple networks (e.g., [88]), can fail to produce the "correct" community assignments.

Even when we tolerate imperfect partitioning (e.g., the down-stream use of the communities will still work) being able to evaluate different algorithms is extremely useful. Different algorithms can have systematic biases that need to be understood (e.g., restrictive models of communities that produce too many partitions with very few nodes). Thus, human feedback and interaction is a crucial part of generating high quality, believable, and usable partitions.

Even those developing community detection algorithms may find it difficult to evaluate the algorithms due to scarcity of ground truth partitions. Algorithm designers often rely on synthesized data or external node metadata in lieu of ground truth community assignment [87]. However, recent research suggests that metadata partitions in the network may not align with community assignments suggested by network structure [43, 66]. Thus, it may be hard to fully characterize a community detection algorithm's performance. Because of this, even experts using community detection algorithms may utilize multiple algorithms (or ensembles) to better understand the quality of the partition. The intuition is that if multiple algorithms arrive at the same result, the researcher can infer that the resulting partition is stable. On the other hand, if each algorithm produces widely varying results from each out, one gets a sense of instability.

Despite the various ways community detection algorithms can fail, the manifestation of those failures are somewhat consistent. Broadly, algorithms will produce a different distribution of errors, but we observe the following three categories:

(1) Split community—A single community split into two (or more) sub-communities. Each sub-community is "valid" (in that there is no incorrectly placed node) but a more valid labeling has all nodes in the same community. Often, variants of the Louvain [13] and InfoMap [74] community detection algorithms generate these kinds of errors more than other algorithms due to "field-of-view" limits [28].
(2) Single-node misclassification—This occurs when a single node is assigned to the wrong community. This occurs most often for "bridge" nodes, where the node has strong connections to two (or more) communities and is mislabeled in the end. These mistakes tend to be less frequent than the split communities but potentially critical as such nodes often have high centrality scores, making them "critical" to the network. Many algorithms "fail" in this way on networks such as Zachary's Karate Club [88].
(3) Merged community—Communities can be mistakenly merged (often due to strong connections between them). This results in large communities and overall less number of communities. Label propagation, for example, tends to merge different communities to a single one. Algorithms that rely on modularity [64] may also run into resolution limits that are unable to pull out smaller sub-communities from larger ones [34].

Figure 2 depicts the three kinds of errors shown in community detection. It is possible to think of split and merged community errors as being composed of multiple single-node misclassifications. That is, one could resolve these errors using many single-node relabeling (i.e., moves). If errors were completely random, our end-users would need to perform single-node corrections, one at a time, to build the correct structures. However, this is not realistic in the context of community detection (and most other clustering problems). Split communities are a very common occurrence (merged communities, slightly less so). Thus, we would like to ensure that our end-user can quickly

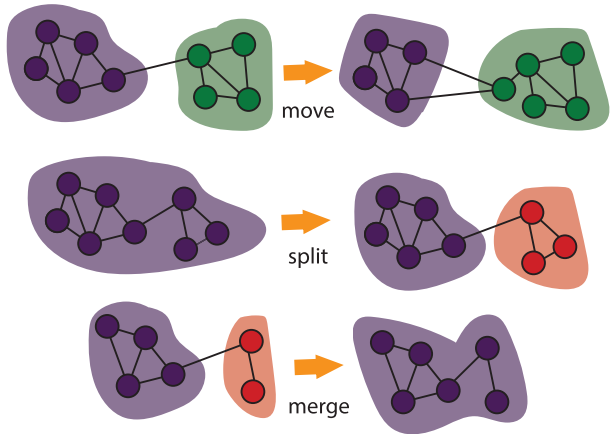

Fig. 2. Different kinds of errors in community detection. (a) Single-node misclassification. (b) Merged community. (c) Split community. On the right hand side, the incorrect community assignment is shown and on the left hand side the corrected version through a specific action (shown in figure) is presented.

correct these errors without moving one node at a time. These three errors are ones that a human agent would either need to correct or understand (i.e., mentally model).

3.1 The Analytical Pipeline

Recall that the end-user is using two "pipelines" in our scenario: a *sensemaking* activity that helps the end-user model the space (i.e., possible communities) and a *data wrangling* activity by which the end-user corrects the algorithmic output (or manually produces labels). These are clearly not independent of each other, and one may view sensemaking as including wrangling decisions in organizing data and conversely wrangling decisions can be aided by analysis. Depending on the state of the network and background knowledge of the end-user, they may focus more on one task or another. For example, if they know all the community labels, this is largely a wrangling task. The less complete the knowledge, the more the end-user must rely on sensemaking techniques to make or validate labeling decisions. CommunityDiff is designed to help with both tasks.

Wrangling work, which in some sense is a more limited task, is supported through automation. AL, mixed-initiative interactions, supervision, and other human-in-the-loop paradigms can take advantage of human action to accelerate the wrangling process [45]. Wrangling is naturally "easier" when the end-user has a perfect mental model of the ground truth. At worst, the end-user could label each node—one at a time. Automation speeds this up. However, with incomplete knowledge, the end-user may need guidance to help decide between different labels which is where the broader notion of sensemaking comes into play.

Work with data often requires iterative loops of sensemaking, where the analyst engages in cycles of data collection, organization, hypothesis testing/analysis, back to data collection, and so on [67] (eventually terminating on a "satisfying" answer). For a community detection task, we can view this process as: the analyst starts with an unlabeled graph and through a sequence of decisions and "wrangling" actions produces a graph where nodes are annotated with a community label. It is this framing that drives our design. The many algorithmic and design decisions we made are motivated by a need to support this progression. In the context of community detection, the sensemaking and analysis process might be aided by high-level information (e.g., how many communities were found? how do the community sizes range?) and low-level information (e.g.,

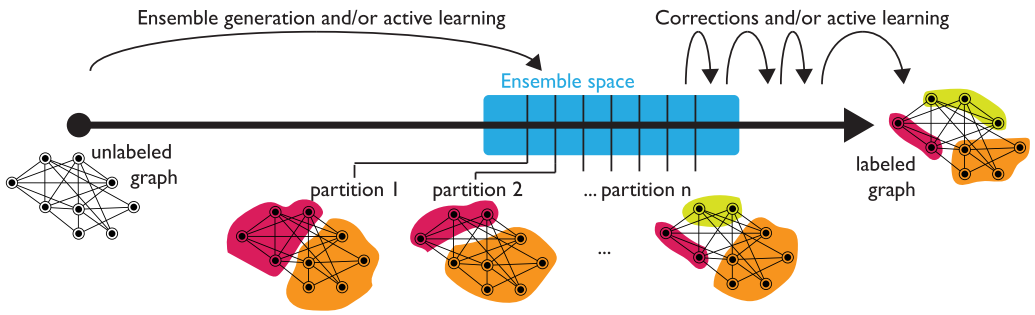

Fig. 3. Exploration process from an unlabeled graph to the final partition. The end-user moves from the unlabeled graph to a reasonable starting point generated by the initial execution of the partitioning algorithms and ensembles, followed by corrections through the use of direct manipulation and active learning.

is this pair of nodes in the same community?). In some cases, the domain-expert cannot evaluate the answers to these questions (e.g., they may not know if the two nodes should be in the same community or may only have a sense that the number of communities range from 3 to 5). Here, information about algorithm or output confidence—which in our case can be derived by using multiple algorithms—can guide a decision.

We graphically depict (a partially linearized) view of this idea in Figure 3. As we argue below, CommunityDiff is intended to capture viable analytics start states by producing a solution space (the ensemble space) and presenting information that help progress the analyst in making decisions. By providing "views" into this space, the end-user may pick solutions that are *close* to the final target (both an act of sensemaking and data-wrangling). Our goal is to make these good solutions highly salient and the differences between solutions obvious. From this point, the analyst can make smaller decisions to allow them to reach the target labeling. When the analyst has a clear model of ground truth such decisions should not require significant manual work (e.g., clicking and dragging individual nodes). When the answer is less clear, CommunityDiff is designed to provide a high level overview of options and a fast mechanism for acting on these options (e.g., through the Co-Community View). Furthermore, CommunityDiff constantly uses the human feedback to improve the classification of nodes that still need to be labeled (aiding the data-wrangling activity).

4 THE COMMUNITYDIFF DESIGN

Having described the ways in which algorithms can fail, and the mechanisms by which a domain scientist can navigate this space, we can describe the key features of CommunityDiff. We also briefly illustrate its use with an example and offer a set of design guidelines that informed our specific implementation.

Figure 1 depicts a typical configuration of CommunityDiff. Upon loading up a new network, Blocks 1–4 are visible. Block 5 is initially hidden from view and is activated to support "fine" manipulation of the network structure. The network is shown in Block 4 (referred as the network diagram), with an initial community assignment of nodes, which CommunityDiff interprets as the best community assignment for the network without any input from the user (further elaborated in next paragraph).

In the standard node-link view, each community is shown inside a bounding box for easy identification of the communities. The nodes and the bounding boxes of different communities are colored differently. CommunityDiff also highlights the nodes (by making them larger in size) for which the system is less confident of the community assignment. This allows the end-user to focus

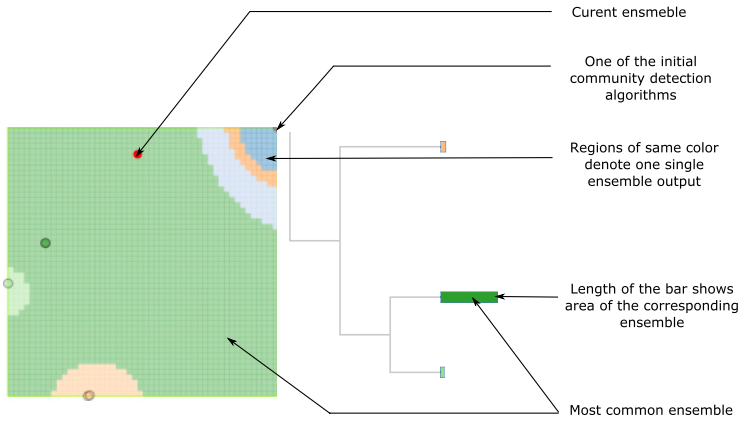

Fig. 4. Elaboration of different components of the ensemble heatmap and the dendrogram. Initial community detection algorithms are shown as multi-colored dots (except the red dot) in the heatmap. Each differently colored region in the heatmap represents a different ensemble output. The red dot corresponds to the current ensemble output shown in the network diagram. The output of different algorithms can be viewed by clicking corresponding multi-colored dots. The dendrogram shows relationship among different ensembles. The interactive bars in the dendrogram correspond to the ensemble that is colored the same in the heatmap and the length corresponds to the area of that region. Output of the corresponding ensemble algorithm can be viewed by either clicking the corresponding region in the heatmap or the corresponding bar in the dendrogram.

on these specific nodes (elaborated in 5.3.3). This interface allows the end-user to correct specific failures in the community detection output (e.g., moving specific nodes, splitting and merging communities). However, this may be costly if there are many errors or difficult if the end-user does not have a mental model of the ground truth. To address both problems, CommunityDiff offers alternative community detection "outputs."

CommunityDiff initially takes six different community detection algorithm and their combinations to identify possible communities. The algorithms can be dynamically and selectively enabled and disabled and CommunityDiff is architected to support the addition of new algorithms. The user can choose any subset of the six algorithms (with the constraint that there must be at least two algorithms for the ensemble operations). For purpose of this article, we chose six very well known community detection algorithms as a starting point. We compare and contrast these algorithms and their different combinations (the ensembles) using an abstract metric space where each algorithm is represented by a dot and each point in this space represent an unique ensemble algorithm based on the point's distance from the "dots" (shown in Figure 4). The distance between two algorithms (dots) is proportional to how different their output is. We call the visualization of this space the *ensemble space heatmap* or simply *ensemble heatmap* (explained in detail in Section 5.2). Output of each of the original algorithms and each ensemble can be viewed by clicking on the point. The community assignment is dynamically changed accordingly in the network diagram panel with minimum number of community reassignments from the previous figures. Nodes which changed communities are highlighted temporarily for easy identification. However, not all ensemble algorithms produce different output. In fact, we see very few unique outputs compared to the vast number (2,500) of ensembles calculated. Thus, in the ensemble map, ensemble algorithms that produce the same output are colored the same, i.e., the ensemble map is partitioned by different colored regions where each region produce a different output. The largest region in

the ensemble map produce the most stable community assignment and this assignment is initially reflected in the network diagram.

While the heatmap view makes salient broad areas of stability and the relationship between algorithms, it is not the most effective view for certain tasks. For example, these ensemble map regions can have arbitrary shape and it is not always easy to tell which is the largest region if there are multiple large regions. Due to their shape, it is difficult to rank other large regions. Moreover, from the ensemble map, we have no idea how different the outputs of these different regions are. For this purpose, we provide a dendrogram (shown in Figure 1(c) and Figure 4) based on the similarity of ensemble outputs. The dendrogram also provided bars to show the size of different ensemble regions so it is easy to identify and rank large regions. Bars are colored according the color of the region in the ensemble map and highlights the corresponding region when hovered on. This is further detailed in Section 5.3.2.

CommunityDiff also provides the user with a choice of different algorithms, multiple views of the ensemble space (explained in Section 5.3.1) and AL with specified number of communities with the option of manually labeled nodes (elaborated in Section 5.4.2) in the panel shown in Figure 1(b).

The user can bring the node-specific co-community lists (shown in Figure 1(e)) by shift-clicking a particular node in the network diagram (the node with thicker borderlines in Figure 1(d)). These lists show how many original algorithms put all the other nodes in the highlighted nodes community. This gives a sense of how robust a particular community in the network is. These list also allow the user to iteratively add/remove nodes from the highlighted nodes community to achieve their desired partition. The user can multiple add/remove constraints at once. It is to be noted that these lists are different for each node and this panel is not initially visible. The user can hide this panels by clicking the "Hide Co-community Lists" button.

To help with navigation, CommunityDiff supports additional functions such as zooming, toggling node labels, hiding particular communities, and so on. which is further detailed in Section 5.3.5. Overall, CommunityDiff provides an end-user with an easy way to compare and contrast different algorithms and ensembles, to incorporate own knowledge about the network/communities without intricate knowledge about the algorithms and have a sense of how good or bad the overall partitioning of the network or a particular community is.

4.1 Example Interaction

Let's follow a political scientist, Alice, as she looks at a small social network she has collected by mining messages on a specific issue between politicians on Twitter. She knows there are a two to four main groups developing different legislation on the issue, but not necessarily which politician has committed to each group (she does know which party they belong to, and some past interactions). Her goal is to generate groupings of politicians that are likely to push for the different legislation. CommunityDiff can help, by giving Alice a workable starting point, the ability to correct errors produced by the algorithms (broadly, wrangling), and information to help decide on labels when there is disagreement or uncertainty (broadly, sensemaking).

Upon loading the network Alice sees four blocks in the interface (Figure 1(a–d)) as co-community lists are initially hidden from view. CommunityDiff automatically executes a number of community finding algorithms and generates the ensemble space view in the heatmap (see Figure 1(a)). The heatmap represents a view of the six algorithms as well as the ensembles of those algorithms (each dot is a base algorithm and each color is a unique ensemble). In this view, Alice can quickly identify the most stable ensemble (the large green surface, here we refer ensemble algorithm outputs as ensemble or ensemble output). This is reinforced by the dendrogram view (Figure 1(c)) that not only shows that this particular ensemble is common, but it is also similar to

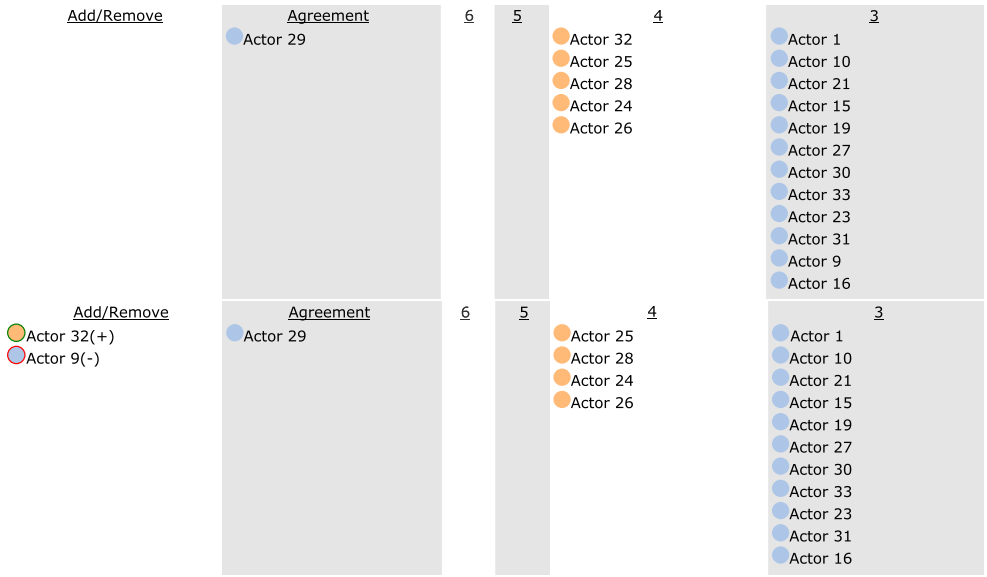

Fig. 5. A partial view of the co-community lists before (figure on top) and after (figure on bottom) Alice's decisions. In both figures, the "Agreement" column shows actor 29, whose co-community lists Alice is working on. Initially, actor 32 is on column "4," which means 4 algorithms agree it should be in the community of actor 29. On the other hand actor 9 is on column "3." After Alice decided to add actor 32 to actor 29's community and remove actor 9 from the said community, these nodes move to "Add/Remove" column.

two other very stable ensembles. Upon clicking on a point in this ensemble inside the heatmap view, the network map (Figure 1(d)) dynamically changes to show the current partitioning. Alice can click on other points in the ensemble space to compare different partitions. CommunityDiff makes the differences salient by briefly highlighting nodes that have "moved" (i.e., changed community assignment or moved from one community to another after performing certain actions) thus supporting visual comparison between algorithms or ensembles. Alice can quickly test different partitions of her network to identify a good starting point for further analysis (the one that is closest to what she knows is true about this network).

Alice can further configure the ensemble space by adding or removing algorithms in the control panel (Figure 1(b)) or change the heatmap view. This panel also allows her to force CommunityDiff to provide feedback to the learning system. Alice can constrain CommunityDiff by either setting a specific number of clusters or using direct manipulation to move nodes between communities. For example, Alice may know that the two communities are aligning with the two main political parties so she may set the number of communities to two. These changes become constraints for the machine learning system and will be adhered to by the newly proposed partitioning.

The underlying AL system can focus Alice on nodes that seem to be unstable. These are made more salient by size. Alice clicks on one of these nodes (politician 29) to see the *Co-Community View* (Figure 5). A number of columns show how often different algorithms agree on which other politician should be in the same community as politician 26. For example, four of the algorithms agree that politician 32 belongs in the same community, but only 3 of the 6 believe that politician 9 should be. Given her background experience, Alice quickly selects politician 32 (which was in the high-agreement column, i.e., majority of algorithms agree that this node should be in the community of the highlighted node) to force it to be in the same community as politician 29. Alice

knows that politician 9 cannot possibly belong in the same community as he is a core member of the opposing party. She indicates to CommunityDiff that it is a negative example (Figure 5). Upon submitting her changes, CommunityDiff propagates these changes and constraints. Once satisfied with any particular grouping, Alice can hide it form the network diagram and focus only on the remaining groups. Alternatively, she could manually label the unstable node and asked the classifier to get to a new partitioning.

4.2 Design Guidelines

Before describing our implementation we define a set of design guidelines that motivated our decisions. These were produced based on the analysis pipeline model described above.

(1) *Respect the Mental Model—The system should seek to quickly bring the system state in line with end-users background knowledge.* The decision of whether the result is "in line" requires end-user feedback. Thus, whenever possible, the end-user should be able to quickly and accurately judge the current results relative to their mental-model [55]. For example, Alice, our political scientist, can quickly reject partitions that produce more than the two to four communities she was expecting.

(2) *Respect the Unknown—The system should help the end-user resolve gaps in their mental model* [55]. When an end-user cannot answer a question such as, "does node *a* belong in community *C* or *D*," the system should provide evidence to support this decision. Options that are more likely to be correct should be made more salient (e.g., large bars in the dendrogram or areas in the heatmap to indicate stable partitioning). The system should also make obvious where feedback from the end-user would be useful. In CommunityDiff, we implement node stability (explained in 5.3.3 in detail), which gives the user a sense of how certain the system is about the node's community assignment. Visually, unstable nodes are made larger in size. The larger the node, the more unstable the node is. This also gives the user a sense of which nodes to focus on. Moreover, using the co-community lists, the user can explicitly analyze a node and its current community assignment with exact knowledge of how many algorithms agree on this assignments, which nodes are more likely to be in the same cluster as a current node, etc. Alice, for example, would be able to quickly see how likely a politician is to vote with one group or another given their past connections to both. At a community level, co-community lists also tell the user if there is smaller coherent cluster in a large community or if there is strong connection between two communities. In general, via the ensemble heatmap and dendrogram, CommunityDiff tells the user that results are likely and how likely they are.

(3) *Respect Decisions—Assume that an end-user will not likely reverse a decision. Once a decision is made, it should be persistent, i.e., it should not be necessary to make it again.* [42, 79] If the end-user indicates that there are only three communities or that nodes *A* and *B* should always be in the same community but *C* and *D* should never be in the same group, this should be respected by the system. CommunityDiff iteratively learns from the user's previous decisions and maintains all previous constraints whenever a new constraint is added. Once Alice decides that there are three communities or that politician 32 belongs with 29, new executions of community detection should not break this (and should, in fact, leverage the information to produce better output).

(4) *Assume Greedy Progression—Actions that create the biggest change should be available and be explicit.* Through a combination of direct manipulation on "batches" of nodes or communities and targeted AL, the system should help the end-user minimize the number of steps needed to complete the labeling. CommunityDiff provided a number of ways to

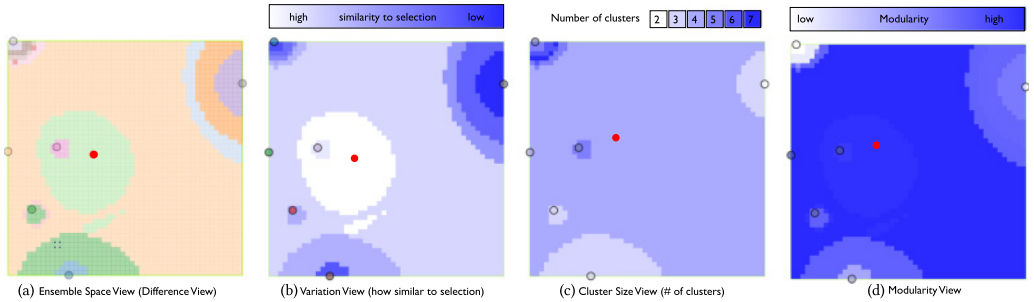

Fig. 6. Four different views for the ensemble space heatmap. (a) Difference view: each differently colored region represent a unique ensemble output, (b) Variation view: shows how similar (white or light blue) or different (darker shades of blue) different ensemble outputs are compared to the current ensemble (red dot), (c) Number of clusters view: shows how many clusters are there in each ensemble output and (d) Modularity view: shows the relative modularity scores of different ensemble outputs.

minimize user effort and ease including merging multiple communities, AL with or without labeled nodes, fixing number of communities, processing multiple constraints at once via the co-community lists. Alice, for example, can focus her data-wrangling effort on the largest legislative group before moving onto the next one.

(5) *Acknowledge Outliers—The interface should provide a way for capturing outliers.* We follow the "NetViz Nirvana" criteria for network visualization by Dunne et al. [27] (specifically, the fourth point: "Clusters and outliers are identifiable") . While we assume that most nodes will have a strong affinity to one community or another, occasionally a node will fit into multiple groups or to none. When this determination is made algorithmically, these outliers should be obvious to the end-user. Node size (determined by node stability) clearly points out outlier nodes. Using co-community lists these nodes can be further analyzed.

5 SYSTEM DETAILS

Having established the high level goals and interface, we focus on the specific implementation and details of the visualizations.

5.1 System and Interface Architecture

CommunityDiff is composed of a back-end which processes the graph data (implemented in Python), and a Web-based front-end (implemented using D3 and Javascript). Recall that the front-end consists of four major panels: an *Ensemble Panel* (consisting of a heatmap projection dendrogram in, panels a and c, respectively, in Figure 1), a *Control Panel* (Figure 1(b)), a *Network Visualization Panel* (Figure 1(d)), and a *Co-Community Panel* (Figure 1(e)). The panels are roughly placed in an ordered way (from left to right) to match the likely workflow: identifying a "good starting point," validating the automated label choices, and then refining labels. In all the views, the decisions of the end-user (selection of algorithms, movement of nodes between groups, labeling groups, etc.) directly influence the learning algorithm.

The *Ensemble Panel* contains two visualizations. The first is the *ensemble heatmap* projection (see Figure 6), which is generated by calculating a similarity matrix (M) for each of the community detection algorithms. A cell in this matrix, $M_{alg1,alg2}$ quantifies how similar two partitions of the network are. Using Multidimensional Scaling (MDS), the similarity matrix is projected into two dimensions. Thus, algorithms generating similar partitions appear close together in this projection

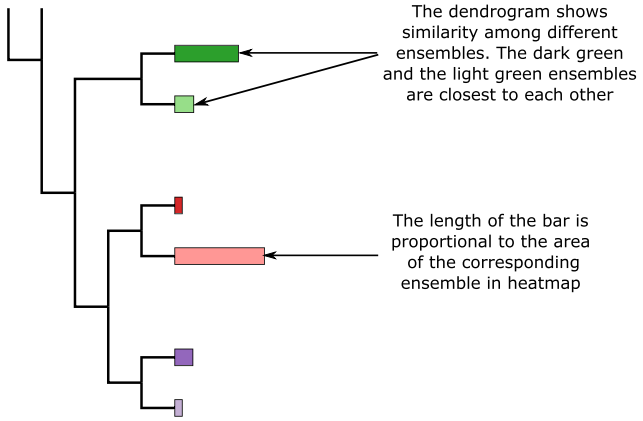

Fig. 7. A small portion of the Dendrogram visualization. The dendrogram shows hierarchical similarity between different ensemble outputs. For example, the dark violet and the light violet ensembles are more similar compared to the pink and the dark violet ensembles. The length of the bar corresponding to an ensemble represents the area of the region covered by the corresponding ensemble in the ensemble heatmap.

(represented as colored dots). From this projection, an ensemble is built for each discretized cell in the heatmap (we describe this metric space in more detail below). Because different ensembles may still produce the same partitioning, there are far fewer ensembles then there are cells, leading to the contours that can be seen in Figure 6. This view is interactive and can control many of the other panels (e.g., clicking on the point for the "fastgreedy" algorithm will load that partition into the network view as will clicking on any other point in the space). The colors in the heatmap can also be set to encode different features of the space including the distribution of cluster sizes.

The second Ensemble Panel visualization is the *dendrogram view* (Figures 1(c) and 7). The view shows the relationship between ensembles based on hierarchical clustering of their similarities. The length of the colored bar provides another view of the prevalence of each ensemble (proportional to the area in the heatmap view). As with the heatmap, the dendrogram is interactive (e.g., click to change ensembles or hover to highlights the point in the heatmap). This view enables the end-user to find related ensembles or vastly dissimilar ones and are worth exploring. Broadly, the ensemble panel allows the end-user to identify a partition of the graph that is near to their mental model and to support the exploration of alternatives when a classification is unknown.

A *Control Panel* (Figure 1(b)) allows the end-user to upload new files, add or remove different algorithms, and control different AL properties. On occasion, it is helpful to compare two heatmap encodings (e.g., one capturing cluster size and the other capturing a partition "score" such as modularity). The control panel allows the creation of a duplicate view for this purpose.

The *Network Visualization Panel* (Figure 1(c)) is a standard node-link diagram which displays the current "working" partition. Layout is done through a standard force-directed algorithm. In the current implementation, nodes are assigned colors based on which community they belong to. Interactive features include: display of labels, "collapsing" of communities to hide them (used when the end-user is satisfied with the nodes assigned to that community). Collapsing of the nodes has the benefit that completed "work" can be removed from view to allow the end-user to focus on the rest of the graph. Communities are enclosed in a rectangle to double-encode the community structure. Through this view, the end-user can make corrections such as moving nodes between communities, merging communities, or creating custom labels. The network visualization, in part, is designed to support the identification of "outliers" (i.e., nodes that might be unstable).

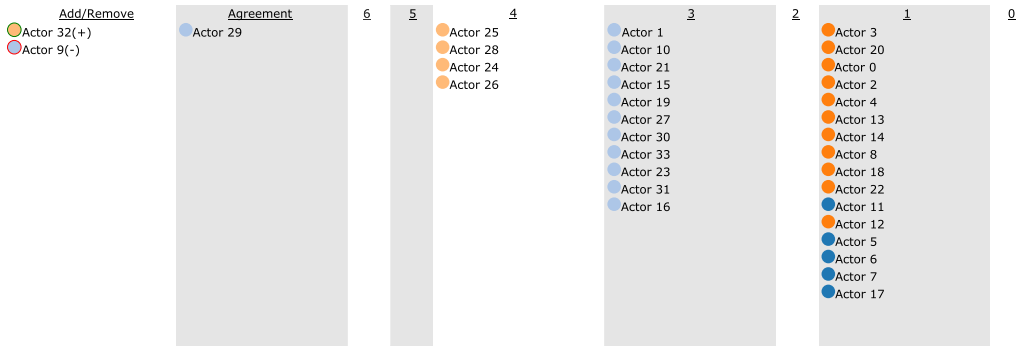

Fig. 8. The Co-Community Lists: The node in the "Agreement" column shows whose co-community lists are being shown. For the columns "0" to "6," column j shows that j of the initial community detection algorithms agree that the nodes shown in column j should be in the community of the node in "Agreement" column. The "Add/Remove" column shows which nodes the user thinks should be added to the community of the node in "Agreement" column (depicted with "+") and which nodes should be removed (depicted with "−").

The end-user may also enable a Co-community View (Figures 1(e) and 8) by clicking on a specific node. This view allows the end-user to find which communities a node could belong to as well as rapidly making corrections and engaging the AL system. For example, in Figure 8, the end-user has selected "Actor 3" (this node appears in the agreement column). Initially, all other nodes appear on the right in one of the six columns. Column 6 includes all nodes that have been found to be in the same community as Actor 3 by all six algorithms (e.g., all six algorithms agree that Actor 14 should be in the same community as Actor 3). No nodes appear in columns 5, 4, and 3 as there is no consensus for nodes at this level. Column 2 contains a number of nodes that two of the algorithms would connect to Actor 3. Column 0 are all nodes that should *not* be in the same community as Actor 3 (by consensus of all 6 algorithms). The end-user can click on nodes to either explicitly indicate they belong with Actor 3 or explicitly should not be (these appear in the leftmost column with +'s or −'s next to them). Here the end-user indicated that both Actor 4 and 28 should be in the same community as 3. Once the end-user is done, clicking on a submit button causes AL to relabel the network based on these new constraints. This view is also explicitly designed to allow the end-user to make decisions when they do not know where nodes belong but also to rapidly correct algorithmic mistakes.

5.2 Algorithmic Details

Below, we describe specific details in generating ensembles and metric spaces that are visualized in the interface. Much of the computation is done at the server level as the calculation of communities and ensembles can be parallelized.

Algorithmic Inputs: For the CommunityDiff prototype, we chose six popular community detection algorithms: FastGreedy [21, 63], InfoMap [75], Label Propagation [71], Multilevel [59], Spinglass [72], and Walktrap [68]. These algorithms are some of the most popular community detection algorithms in use today and capture a broad range of objective functions and underlying techniques. Note again, that nothing prevents the addition of new algorithms.

Fastgreedy, as its name suggests, greedily merges communities iteratively by maximizing modularity, a measure of "modular strength" of a network. Modularity, Q is captured as

$$Q = \frac{1}{2m} \sum_{vw} \left[A_{vw} - \frac{k_v k_w}{2m} \right] \delta(c_v, c_w),$$

where v and w and two nodes, k_i is the degree of node i, and c_i is the community label for node i, m is the total number of edges in the graph, A is the adjacency matrix representation of the graph (i.e., $A_{vw} > 0$ if an edge exists between v and w), and δ is the Kronecker delta—an indicator for testing if the communities are equal. The intuition for this function is that we are testing the number of edges within a community versus the number expected with random assignment. Stated differently, a strong community contains more edges between its members than expected by chance.

InfoMap, on the other hand, follows a very different approach, and aims to provide the shortest description length of a random walker trajectory. The description length is measured by the expected number of bits per vertex to encode the random walk path. This algorithm uses the minimum description length principle in information theory and follows the idea that a random walk within a community is likely to stay within the same community as the number of intra-community edges are higher compared to the number of inter-community edges.

The remaining four algorithms have additional variability and we refer the interested reader to the source publication. While the algorithms tend to agree on high-confidence communities, in practice, they generate enough variety in identified communities that they are a good mechanism for building ensemble spaces. All six algorithms can operate over weighted networks (a common requirement in network analysis).

5.2.1 A Metric Space for Ensembles. In order to define a "space" for our ensembles, it is necessary to define a suitable metric to represent relative distances between different partitions of the graph (i.e., the outputs of two or more community detection algorithms). Ideally, the distance metric will be bound within some fixed range, preferably between zero and one. We are equally satisfied with a dissimilarity metric as one that measures similarity (we can simply subtract the value from 1). As a specific example, if two partitions agree on every node pairing (i.e., they are in the same community or are not) the metric should return zero. On the other hand, if the two algorithms disagree on every pair we would expect to find the distance to be one (normalized).

The most popular metric for this purpose in the community detection literature is Normalized Mutual Information (NMI) [82]. Let X and Y be two arrays that denote the community partitions determined by two different community detection algorithms on the same network. The NMI for two partitions X and Y is calculated by determining the entropy for the two partitions (i.e., $H(X)$ and $H(Y)$) and mutual information, $I(X;Y)$ as

$$H(X) = -\sum_{x \in X} p(x) log(p(x)),$$

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log\left(\frac{p(x,y)}{p_1(x)p_2(y)}\right).$$

The normalized for $(NMI(X;Y))$ is then calculated as

$$NMI(X;Y) = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}}.$$

Roughly, we are capturing a partition as a probability distribution of a node falling into a community. The entropy reflect the information contained in this distribution and the mutual information is the shared information between the partitions. NMI lies between zero and one, and can compare two algorithms which produces different number of communities unlike some other metrics. The inverse, $(1 - NMI)$, can be used as a suitable distance metric as it satisfies all the properties discussed above.

Given a graph G and a set of community detection algorithms $CD = \{cd_1, cd_2, \ldots, cd_n\}$, let op_i correspond to the set of clusters obtained by running cd_i over G. We calculate the distance of op_i

for each of the outputs produced by the different community clustering methods, so that for each method, we have an n-dimensional distance vector, where n is the number of different community detection algorithms.

5.2.2 Multidimensional Scaling (MDS). For visualizing the relationship between the outputs of the six algorithms we would like to project them into a 2D plane where their distances in this plane is proportional to their (dis)similarity. We have opted to use MDS [22] for this purpose as our qualitative experience is that it is effective in this context (fast and accurate). Other dimensionality reduction techniques (e.g., PCA, t-SNE) can also be used. By projecting the algorithms into a 2D plane we are now able to create ensembles that combine the algorithms by using Euclidean distance to generate a set of ensemble weights (i.e., how much each algorithm's "vote" counts towards the ensemble).

5.2.3 Creating Ensembles. Having a suitable representation of the metric space (the MDS projection) enables us to automatically calculate an ensemble for each point in this space. Our objective is that any selected point in the MDS projection will have an associated ensemble (i.e., partition) that is similar to our original algorithms in proportion to the distance to those algorithms. For example, note the red dot in the leftmost panel of Figure 6. We would like the ensemble at this point to be very similar to the algorithm immediately to its left (the light purple dot) and very dissimilar from the algorithm at the bottom of the space (in blue).

There are many possible ways to combine different community detection algorithm to form an ensemble algorithm. Ideally, CommunityDiff would allow the end-user to select different ensemble algorithms. However, we have currently implemented one based on a variant of the "co-community graphs" method, which also takes the original graph structure into account (leveraging our prior work [17]). An edge between two nodes in a co-community graph indicate that those two nodes belong in the same community. The absence of such an edge indicates that the two nodes are not in the same community. Within the context of ensembles, the weight on an edge between any two nodes is determined by the number of algorithms that put them in the same community (e.g., if 3 of 6 algorithms place nodes A and B together the edge between them has a weight of 0.5). With one algorithm, the co-community graph will contain a number of disconnected components (each component capturing one community). However, a co-community graphs that is generated by multiple algorithms will often contain only one connected component which will be extremely dense. The intuition is that with many algorithms, at least one algorithm will put each pair in the same community. To mitigate this problem we only retain co-community edges that were also an edge in the original graph. The output is a graph that is no more dense than the original input graph but has co-community weights on the edges.

Briefly, to find the co-community graph for a given ensemble point (x, y in our MDS projection) we apply the following algorithm: given a graph G and a set of community detection algorithms $CD = \{cd_1, cd_2, \ldots, cd_n\}$, where c_i has weight w_i, first run the algorithms over G. Let w_i be the distance of the ensemble "point" (x, y) in the metric space to algorithm i (recall that each algorithm is placed at some point in the MDS projection). Intuitively, if the ensemble point is close to i's position, then w_i is high (this is determined by the inverse of the Euclidean distance). For each edge e joining vertices v_i and v_j in the network, initialize weight of e as zero and if v_i and v_j are put in the same cluster by cd_i, then increase the edge weight by w_i. The total sum of weights for all community clustering algorithms is normalized to 1. For an edge e joining vertices v_i and v_j, if very few algorithms put v_i and v_j in the same cluster or the algorithms that put v_i and v_j in the same cluster have low weight (we do not want the ensemble algorithm output to be similar to result of these algorithms), e has a low weight (see Figure 9 for a graphical illustration). We prune these edges by removing edges below a certain threshold to improve the ensemble output. A low

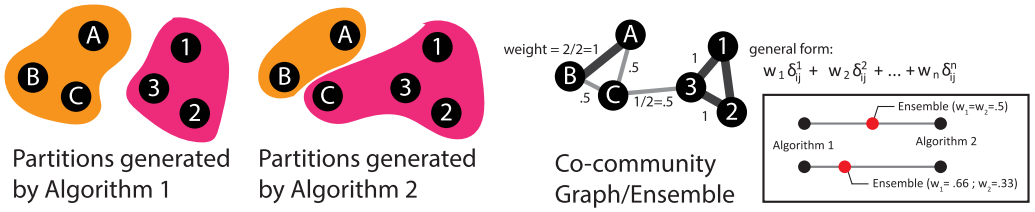

Fig. 9. A graphical example of ensemble generation through the co-community graphs. Algorithms 1 and 2 generate two unique partitions: {{A,B,C},{1,2,3}} and {{A,B},{1,2,3,C}}. The co-community graph (right) has an edge between each pair of nodes that were in the same community (in any of the partitions). The weight of the edge is proportional to the number of times the nodes appear together among the algorithms (e.g., both algorithms 1 and 2 agree that nodes A and B should be together, hence a weight of 1; however, they disagree on node C, once placing it with A and B and once with 1/2/3). Each algorithm is thus a vote. When generating the ensemble, this vote is weighted by the position of the ensemble relative to the algorithms in the ensemble metric space (MDS). The inset shows two different weighting schemes based on ensemble placement (the weights must sum to 1; δ_{ij}^n is the Kronecker delta set to 1 if nodes i and j are in the same community for according to paritioning algorithm n, 0 otherwise).

threshold would not improve the results by much and a high threshold decomposes the network into too many connected components. As we normalize the clustering algorithm weights, the edge weights are restricted to range between 0 (no algorithms put the vertices joined by the edge in the same cluster) and 1 (all algorithms put the vertices in the same cluster). In practice, a threshold value of 0.33 does not over-split the graph and improves the clustering results. Note that, for different networks, different threshold values may improve the results further, but in general, our chosen threshold value works on a wide variety of networks.

Because the co-community graph with multiple algorithms does not properly disconnect components we run one final community detection algorithm on the co-community graph to split it into communities. Our experience is that InfoMap works most effectively for the kind of data produced by co-communities. Other algorithms, like Louvain or Walktrap, can also be used for the final run. In principal, any community detection algorithm can be chosen as long as the algorithm performs reasonably on the original graph.

As we described above, not every ensemble will be unique. Different combinations of algorithms can still produce the exact same output. In CommunityDiff, each *unique* ensemble receives an identifier. In most cases, the same ensemble will appear contiguous in the ensemble space resulting in the contour plots seen in the figure.

From the perspective of implementation, though it would be possible to generate an ensemble for every pixel in the 2D-space this is somewhat impractical due to computational costs. It is also unlikely that ensembles vary dramatically at those scales (a shift of one pixel does not cause weights to change enough to impact the final ensemble). For CommunityDiff, we perform a grid-search on a discretized grid. Specifically, we assume a 300×300 space and use a non-overlapping grid of size 5×5. Ensembles are only computed once per grid.

In the control panel the end-user can eliminate certain algorithms from consideration in the ensemble space. Conversely, they can use the current ensemble (manually generated or through AL) as a new "anchor" point in the metric space. This partition is treated the same way as the output of the other algorithms when calculating the map (a new dot is added to the heatmap for this ensemble).

5.3 Visualization and Interface Elements

We have selected a number of visualizations and interactive elements to support the end-user. We have opted for largely conventional forms (e.g., trees, heatmaps, node-link diagrams) as these are familiar and effective for the task.

5.3.1 Ensemble Space Heatmaps. Once created, our ensemble space heatmaps can be used for a number of tasks related to the different design objectives we defined. We propose four different kinds of views of the metric space to help the user choose a suitable algorithm or ensemble. These different views align with our first design guideline as they allow fast and accurate selection of a desired algorithm or ensemble. When clicking on the heatmap (in any configuration) the network diagram is dynamically changed (this clicked on point becomes the "working" ensemble).

Difference view (Figure 6(a)): This is the most general static view of the metric space. Each unique ensemble has a unique (orthogonal) color. Though rare, it is interesting to note that two separate (i.e., disconnected) regions in the metric space may produce the same clustering (visible because they have the same color). The goal of this view is to help the end-user identify the "jumps" in the initial labeling phase (see the first big "jump" in Figure 3).

Variation view (Figure 6(b)): This view is the only dynamically generated heatmap. When the end-user clicks on a specific ensemble point, all other grid points are colored by comparing their associated ensemble to the current selection (using the NMI metric described above). In the figure, the end-user clicked to place the red dot (the selected ensemble). The white region surrounding that point has a distance of zero (meaning high similarity). The further one moves from that point the more dissimilar the ensembles get. This allows the end-user to identify other, subtly-different ensembles, but also ones that are radically different. This view supports our design guidelines in helping resolve knowledge gaps and identifying outliers.

Cluster Size view (Figure 6(c)): Recalling the "mental model guideline," domain experts often have a sense of the number of clusters they should expect to see. This view plots the number of clusters detected by the ensemble at each of the grid point. The end-user can readily choose ensembles or algorithms that have the same number of clusters as the goal or close to the number of clusters of the goal.

Modularity view (Figure 6(d)): There are a number of measures of community "quality." Modularity is one that is often applied. This view depicts the modularity score for each ensemble cell in the space. This allows easy choosing of high modularity ensembles or confirming that the chosen ensemble produces high modularity output. In the future, we would like to add additional such metrics. As with the variation view, access to known metrics (i.e., modularity) supports decision making by the end-user.

The goal of all these views is to support exploration within the ensemble space to find a good starting point for additional labeling. This corresponds to allowing the user to see, and decide between, possible starting points in the blue "ensemble space" in Figure 3.

5.3.2 Dendrogram. In the construction of the heatmap representation of the ensembles we noticed that it was often difficult to understand how common different ensembles were (in particular those that were not large or had odd shapes). It was also difficult for end-users to understand the relationship between all the different ensembles in the heatmap. While algorithms were situated in the metric space so that they could be compared, the contour "blobs" for the ensembles were not so easy to interpret (i.e., how does one easily measure the distance between two irregularly shaped contours in the metric space?). To better support these comparisons we generated a dendrogram (see Figure 7). The view is a depiction of a standard agglomerative hierarchical clustering (using the NMI based distance metric). The size of the bar at the end of each leaf indicates how

prevalent that ensemble is. The heatmap and dendrogram views are linked so that brushing over one highlights the other. Clicking on one changes the ensemble in selection in the other.

As with the heatmap, the dendrogram was intended to support the rapid exploration of alternative ensembles that could act as starting points for further labeling. As a second benefit both allowed the end-user to understand the stability of different ensembles in making decisions. The end-user could act based on their knowledge (e.g., number of clusters) but receive useful information to guide their exploration when their mental model was incomplete.

5.3.3 Node-Link Diagram. The network diagram view (see Figures 1(c) and 11) is a standard implementation using a force layout [48]. To incorporate community groupings, we add additional constraints such that all nodes from a community are grouped together inside a rectangular bounding box. Each bounding box (along with the nodes inside them) is colored differently. We would like to ensure that there is color stability for the communities between ensembles. More explicitly, we would not want to disrupt the end-user by randomly assigning colors every time a new ensemble was chosen (most communities are stable between these ensembles and a large, random color change would make a small change appear big).

We applied the following algorithm to achieve this stability. For each node we determine a stability score: In a graph, G, for each edge, e_{ij}, connecting two vertices v_i and v_j, we weight the edge by the number of community detection algorithms among the initial six that put v_i and v_j in the same cluster. For each node v_i, we assign a stability score k_i, where k_i is the weight of the maximum weight edge connected to v_i. Put another way, we ask: if edges were deleted based on a threshold from low to high, when would the node become disconnected from all others? In this case, the most stable nodes have a stability score of six as we are using six community clustering methods. Each node is assigned a color initially. A community takes on the color of the most stable node in the community (ties are broken by lexicographic ordering). However, even with the consistent colors, it is often difficult for the end-user to track changes when they switch between ensembles. To better support mental model preservation, nodes that change color during a different community assignment are briefly highlighted by changing their stroke width (i.e., outline).

Calculating the stability score has an additional benefit that stability is inversely proportional to ambiguity. By resizing the node based on this measure we can guide the end-user to nodes that would help the AL infrastructure. The end-user can also make corrections in this view by selecting multiple communities for merging or clicking on nodes to reassign them to alternative communities. Communities can also be hidden from view to eliminate distraction.

5.3.4 Co-Community View. While dealing with real-world networks, in most cases the user only has partial knowledge about communities in the network. In majority of cases, this knowledge comes in the terms of which nodes belong together and which nodes do not. Co-community lists present an easy way to incorporate these knowledge. These lists can be viewed by shift clicking any node in the network diagram. The first list is the *Add/Remove* list, which is initially empty. The next list the shows the node clicked on the network diagram, which is hereby referred as the focus node. The next seven lists are generated based on how many community detection algorithms put the other nodes in the same community as the focus node. A list marked as *5*, contains the nodes that are put in the same community as the focus node by five community detection algorithms. Note that, the number of these lists varies according to the number of community detection algorithms chosen by the user. Within each list, the nodes are sorted by the shortest distance from the focus node.

As described before, the Co-community View (Figure 8) allows the end-user to make rapid comparisons and corrections. It is intended to be used during the final phase of labeling when many small corrections must be made. The view supports the design guidelines of respecting the

unknown (i.e., uncertainties) and the mental model. By focusing on one node at a time (this is often a central node to a community or a bridge that connects two communities), the end-user can quickly isolate both mistakes and "trends." For example, if many nodes of the same community appear in the 6 column (the high agreement) the end-user can (a) see this due to color coding, and (b) select all these nodes by double clicking to add them definitively to the focus node's community.

There are clearly scaling concerns with the co-community view (long lists of nodes are hard to maintain). However, we have experimentally found (see below) that true positives often appear in the columns 5 or 6 (high positive agreement), whereas true negatives (the bulk) are in columns 1 or 0 (high negative agreement) and can safely be disregarded.

Co-community view allows processing of each individual node compared to the focus node. We use ranked list to give idea about the "goodness" of the cluster of the focus node. If the nodes in the lists with high positive agreement (column 5 or 6) are of same color as the focus node, then the cluster is stable. On the other hand, if these lists contain nodes with different colors then the focus node has strong connection to multiple communities or there is a split community mistake. We do not use any kind of aggregated charts because that restricts manipulation of individual nodes. In the agreement lists, nodes are sorted according to shortest distance from the focus node as distant nodes are more unlikely to be in the same community as the focus node.

5.3.5 Other Controls. CommunityDiff has a number of other basic features to support the analyst. These include undo operations to move back to previous ensemble states, file upload (loading in a new network) and downloads (retrieving the community labels). CommunityDiff operates on GML graphs, a standard graph descriptor language. Brief descriptions for these controls are provided below.

(1) *Merge:* As discussed before, a common problem of many community partitions is oversegmentation, where a single community is divided into multiple smaller communities. We provide a simple remedy of this problem—a merge functionality. A user can click on the legend boxes in the network diagram to select corresponding community or click on a selected legend to deselect them. Community bounding boxes in the network diagram are highlighted when hovering over corresponding legends and vice versa. Clicking the *Merge* button merges selected communities.
(2) *Hide/show communities:* Often a user wants focus on a particular set of communities in a network. By shift clicking on the legend boxes in the network diagram, a user can hide a community or show a hidden community in both the network diagram and the co-community lists. This helps focus on the communities the user is currently working on.
(3) *Download:* We also provide a way to download community assignments if a desirable partition is achieved using a *Download* button. It downloads a text file containing node ids and their corresponding community assignment. If a partition is generated using an ensemble, we provide a way to download weights for different community detection algorithms using a *Download weights* button.
(4) *Find ensemble:* We provide a fast and easy way to highlight an ensemble region nearest to the community assignment (based on NMI score) shown in the networks diagram panel, using the *Find ensemble* button. If there is an exact match, then the selected region is highlighted in green, otherwise in yellow.
(5) *Undo:* Because CommunityDiff can constantly recompute communities based on interaction, it is possible that an action will lead to a non-desirable partitioning. We provide the end-user with the option to return to the previous state. This includes changing ensemble or a community detection algorithm, AL, labeling a node, merging two or more communities and changing community assignment using the co-community lists.

In addition to the commands above, CommunityDiff also allows the end-user to explicitly rename communities, upload new files, or download the output of the analysis. Visualization specific interactions include zooming and panning, viewing and hiding node labels, and hiding/showing different visualization panels.

5.4 Human-in-the-Loop Machine Learning

Feedback from the end-user–by means of interaction—allows CommunityDiff to make dynamic alterations to the partitions in anticipation of the end-user's actions. This again is designed to eliminate unnecessary labeling steps.

5.4.1 Co-Community View Processing. Within the context of the co-community view, once nodes are selected for either inclusion ("must link" to the target node) or exclusion ("most not link"), CommunityDiff propagates these decisions to help partition the rest of the graph (as per our guidelines, manual decisions–those made by explicit selections—are respected and retained). In the simplest case, if our target is node A and we manually indicate that node B should be in the same community and A and B are adjacent (i.e., connected by an edge) or B is adjacent to at least one node in A's community, B is placed in A's community. If B is not adjacent to A's community, we merge the communities of A and B (there can be multiple ways to tackle this problem, we opt for the simplest solution of merging the two communities as very little research is done in tackling this particular problem). When nodes are explicitly indicated to not be part of the target node's community (say node C), C's label will be assigned to the next most likely community that it is not "banned" from. If no such community is found, we create a new community for each connected component in the subgraph spanned by these nodes.

5.4.2 Active Learning. In the traditional sense, AL asks the user to label specific data points and performs semi-supervised learning iteratively. In CommunityDiff, we do not explicitly ask users to label certain nodes, but we do highlight unstable nodes (larger in size) which could be manually labeled and provide useful information to the learner (a blend of mixed-initiative and AL). Addressing this "question" will trigger the algorithm to re-partition the graph. However, any form of supervision (e.g., merging, splitting) will be taken into account.

A key background task for CommunityDiff is an adaptive classification component that can produce better communities given end-user behavior. CommunityDiff uses both constraints on the number of clusters and community constraints in generating new partitions. For the purpose of speed and efficiency (recall, this is one of our design goals) only nodes that can make a difference to the AL algorithm should be manually labeled. The uncertainty (and conversely, stability) metric we use was described above as a side-effect of community color generation.

When a learning step is engaged, CommunityDiff applies the following procedure: Assume k is the desired number of communities (indicated in the control panel). Clearly, the classifier cannot produce more or less than the number of community labels then have been manually created. If there are k_1 manually labeled community labels, and $k_1 > k$, then the value k is adjusted upwards to k_1. The node labels of the labeled nodes are chosen from the community assignments of the network shown in the network diagram panel. First, it is decided which labels should be selected. If there are k_1 labels among the manually labeled nodes, we choose these k_1 nodes. If $k_1 < k$, we choose the other $k - k_1$ labels using community sizes. We choose the labels of the largest $k - k_1$ communities not including the k_1 labels already selected. Apart from the manually labeled nodes, only the most stable nodes of the communities having the selected labels, are chosen as labeled nodes. All the other node labels are determined by a semi-supervised classifier by Zhu et al. [90]. A brief description of which is provided in the next section.

We choose the graph structure for the semi-supervised classifier based on the output of the community detection algorithms where we retain the original graph structure but each edge is weighted according to how many algorithms put the nodes connecting that edge in the same community. This is similar to choosing the co-community graph for a specific ensemble except the fact that each algorithm is weighted equally and there is no removal of edges based on a threshold.

We can think of shifting a node v from *community A* to *community B* as a form of soft assignment. We are only assuming that v has the same community assignment as the most stable nodes in *community B*, that means after the AL, unstable nodes in *community B* may have a different community assignment than v.

5.4.3 Classifier Description. In CommunityDiff, we have used a modified version of the semi-supervised classifier by Zhu et al. [90]. This classifier employs Gaussian random fields and harmonic functions to predict node labels given only network structure.

Given a graph $G = (V, E)$ with $l + u$ vertices, we assume we have l labeled instances $L = \{(x_1, y_1), \ldots, (x_l, y_l)\}$ and u unlabeled instances $U = \{x_{l+1}, \ldots, x_{l+u}\}$. w_{ij} denotes the weight of the edge between i and j. If there exists no edge between i and j, w_{ij} is zero. We also assume the number of classes is C.

The aim of this classifier is to compute a function $f : V \rightarrow \mathbb{R}$ over G to label the unlabeled nodes using f. Ideally, unlabeled points near each other should have the same labels (due, in part, to homophily [12]). f can be rendered as a Gaussian field with energy function,

$$E(f) = \frac{1}{2} \sum_{i,j} w_{ij}(f(i) - f(j))^2. \tag{1}$$

The minimum energy function is harmonic in nature. That is, for labeled data, the value given by f matches the labels exactly and for unlabeled data, $\Delta f = 0$, where Δ is the combinatorial Laplacian defined as $\Delta = D - W$, where D is the diagonal matrix with the non-zero diagonal entries $d_i = \sum_j w_{ij}$ and $W = [w_{ij}]$ is the weight matrix.

However, the labels for the unlabeled data can be calculated using only matrix operations. The Laplacian matrix can be partitioned as follows:

$$\Delta = \begin{bmatrix} \Delta_{ll} & \Delta_{lu} \\ \Delta_{ul} & \Delta_{uu} \end{bmatrix}, \tag{2}$$

and f can be defined as a $C \times (l + u)$ matrix, such that,

$$f = \begin{bmatrix} f_l \\ f_u \end{bmatrix}, \tag{3}$$

where, $f_l[i][j] = 1$, if x_i is labeled as j. The other entries of f_l are zero.

$f_u[i][j]$ denotes the probability that x_{l+i} has label j. This can be computed as,

$$f_u = -\Delta_{uu}^{-1} \Delta_{ul} f_l. \tag{4}$$

For each unlabeled point the label with highest probability is chosen.

6 EVALUATION

To evaluate CommunityDiff, we would like to demonstrate that individual components (e.g., the ensemble generation and co-community lists) are effective as well as demonstrating the usability of the overall system. Our hypothesis is that an ensemble will perform better compared to any individual algorithm and that the best ensemble is often the "largest" one. Similarly, we need to show that the co-community lists provide an idea about the "goodness" of a cluster along with

batch processing of selected nodes. We can verify both of these components via automated experiments where ground truth knowledge is available or can readily be inferred. To test the end-to-end system we performed a controlled lab study.

It is worth noting that a key challenge of evaluating a new community detection algorithm or software is the lack of network data with known community assignments (ground truth). The handful of "toy" datasets often used for this task are often too small to be useful (e.g., Zachary's Karate Club Network [88] or the Football Network [35]). Synthetic datasets such as those generated by the LFR benchmark [51] often create advantages for community detection algorithms that vanish with real networks [17]. To create datasets for automated evaluation, we apply a common technique which is to find unlabeled networks and create a ground truth label based on some feature of the nodes that is not structural (i.e., does not depend on edge configuration). While this is imperfect, it is nonetheless common practice for large-scale evaluation [66].

6.1 Ensemble Evaluation

In the DBLP co-authorship network [87], 846,082 nodes represent authors and the 2,783,165 edges are co-authorship relations. Authors are placed in communities based on their frequent publication venues. Authors who publish in the same conferences or journals thus form a community. As our focus in this work is on disjoint communities, we utilized the DBLP data to construct a ground-truth label based on the *most* frequent publication venue (i.e., an author who publishes 10 times in venue X and once in Y will be labeled X). Using the DBLP data (http://dblp.uni-trier.de/xml/), we created our own DBLP co-authorship network with ground truth labels created as described above. As journals tended to be the longest running "venues" (and had clear continuity), we only utilized articles published in journals for this analysis (yielding 1,530 "communities").

To simulate a realistic use case for this data (e.g., a user organizing a sub-field for a review article), we generated random sub-graphs from the larger network. A node in the larger graph was chosen at random and a subgraph spanned by its 2-step neighborhood was collected. In this sub-graph, we ensure that each ground truth community was connected as all community detection algorithms make this assumption (i.e., there should be a path between each node in a community that does not require moving through another community). Additionally, we removed communities with only 1 or 2 nodes from each network and networks with a single community. We generated 100 such networks as a test dataset. The sizes of these networks vary from 50 to 150 nodes. We also made sure there is at least two communities in the network and disregard networks with 15 and more communities as in these cases the communities are very small (3–10 nodes).

Each of the 100 networks was analyzed using the CommunityDiff backend. We were specifically interested in the partitions generated by each algorithm and a few key (i.e., "common") ensembles. To determine the accuracy of these partitions, we compared each to the ground truth labeling using the NMI metric. In three of the networks, all algorithms produced the exact same output (which was the ground truth). As our goal is to study those situations where ensembles can be used, we exclude these three networks from further evaluation. Recall our general hypothesis that ensemble methods better than base algorithms. Figure 10 clearly shows that the best ensemble outperforms all the other algorithms as it has the highest average NMI score with ground truth and performs best in 88 cases out of 97. One of the reasons for using the heatmap ensemble projection was the belief that an ensemble that is stable across most of the space (i.e., the largest and most salient) would be the best starting point for labeling. To test whether this was valid, we tested the most common ensemble in our ensemble space. We find that in 25 out of 97 cases, the most common ensemble performs better than any base algorithm. This is followed by spinglass that

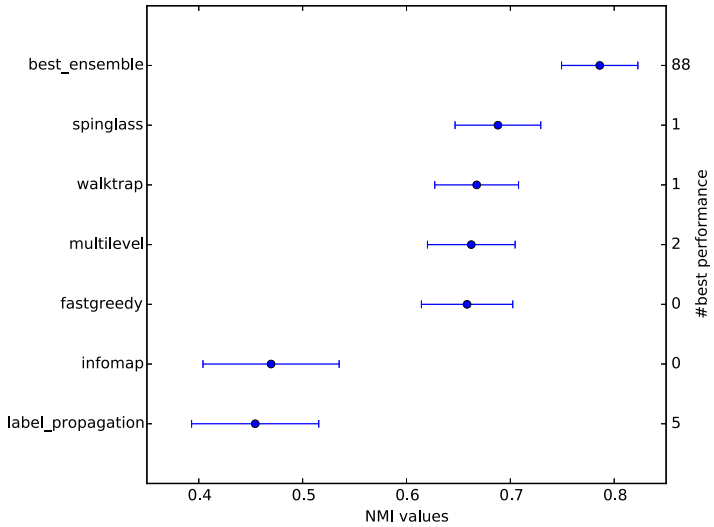

Fig. 10. Comparison between average NMI values with ground truth of different community detection algorithms and number of times an algorithm performs the best. The figure shows that the best ensemble output has higher average NMI score and performs the best among all algorithm in 88 out of 97 cases.

performs the best in 22 cases and InfoMap that is best for 18 cases. However, the mean NMI for InfoMap is much lower and more variable. In those situations when the most common ensemble is not the best partitioning, the best ensemble is found within the top-5 most common ensembles.

6.2 Co-Community Evaluation

To determine whether the Co-community View would correctly emphasize nodes, we performed a second experiment to test whether nodes listed in the top columns of the Co-community View were likely to be good additions to a selected node's community. Specifically, given a "target" node, we would like to know if those nodes displayed in high-agreement columns (i.e., the nodes for which all or all but one algorithm agree that they should belong to the same community as the target node) are likely to be in the same community as the target. To verify this, we chose 10 random networks from the 100 sampled DBLP networks and for each network chose 10 random nodes and studied their co-community lists manually. For 3 of the chosen networks, we find that the different algorithms give widely different results and for most of the 10 randomly clicked nodes (for all 3 networks) the high-agreement columns (those with 5 to 6 algorithms agreeing on a community) are empty. We disregard these networks for this study. For the 7 remaining networks, We observed that in 59 out of 70 instances, all nodes listed under list 6 and 5 (i.e., were agreed upon by most algorithms) were in the ground truth community of the target node. We manually validated that there were multiple nodes in these lists in all 70 cases.

6.3 User Study

In order to reliably test CommunityDiff with end-users we needed to identify a network dataset where ground truth was available but where highly specific domain expertise was not necessary so that we could have a larger participant pool. To ensure high agreement, we would also prefer a network where community labels are not subjective. For this purpose, we generated our own network for evaluating our system. We created a network by utilizing the handwritten digits from

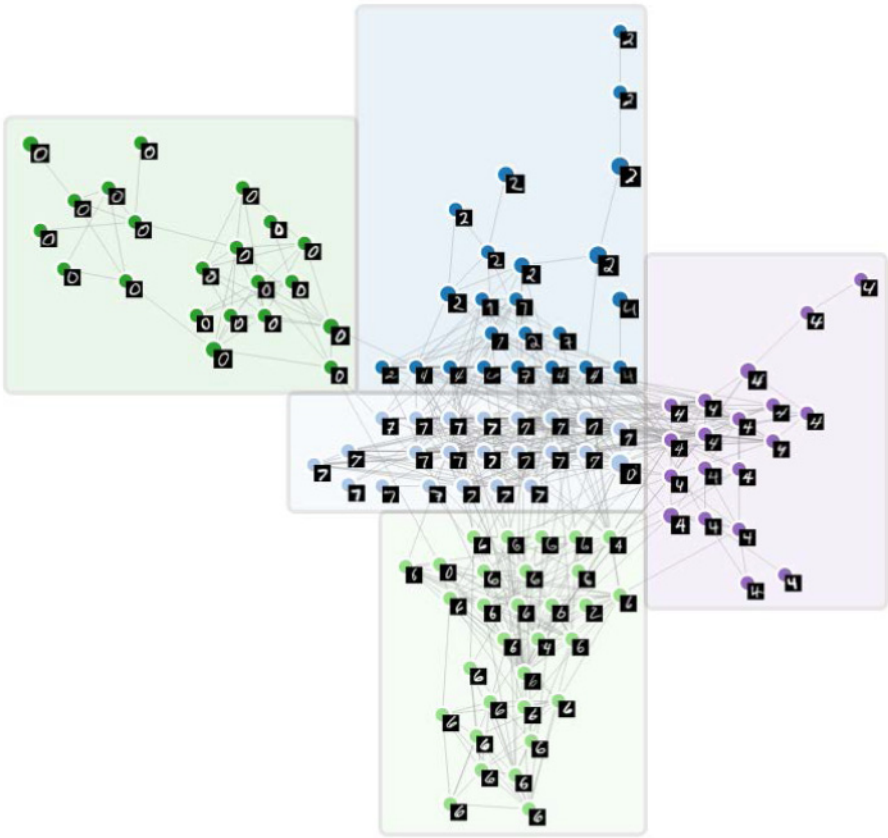

Fig. 11. A screenshot showing MNIST handwritten digits network in CommunityDiff. This screenshot does not reflect the ground truth partition of the network as all communities except communities containing "0" and "4" has multiple different digits as nodes.

MNIST [53] (these are the digits 0–9 written by humans and used in OCR challenges). We cast this as a network problem by creating an artificial network where digits are the nodes and two nodes are linked by similarity between the two digit images. This approach is roughly equivalent to standard clustering where points near each other in the n-dimensional space are considered for inclusion in the same cluster. Though this may appear artificial as the data itself is not structured as a network, this transformation for classification and clustering purposes is common in text clustering (e.g., [30, 61]) or recommender systems (e.g., [44]) among other domains (where edges have the meaning "is-similar-to"). Regardless, it allows us to generate a network that satisfies our goal of reduced domain expertise and high agreement.

To create the network, we randomly sampled 30 images each of the digits 0,2,4,6, and 7 (these have high inter-rater reliability in ground-truth labels). We calculate pairwise similarity between images using a Gaussian kernel and only retain the top 5% values as edges. We took the largest connected component of this graph (117 nodes and 554 edges) as the test dataset. When displaying these nodes in CommunityDiff, we included a small image of the handwriting sample on top of the usual circle. Thus, we do not explicitly provide the ground truth and allow the users to decide on their own which nodes should belong together based on the images. Figure 11 shows a screenshot of the MNIST network in CommunityDiff.

Our subjects included 12 graduate students at a large University (11 with experience in using network datasets including online social networks, linguistic networks like word networks and email communication networks) from three different departments namely, computer science (5 students), school of information (6 students), and mechanical engineering department (1 student). All participants were given basic training with CommunityDiff (to familiarize them with different components and commands of the system) and handed a cheat sheet to remind them of commands. Subjects were then asked to achieve optimal partitioning of the handwritten digit network as quickly as possible. The study was conducted using a laptop computer (this computer is only used for display and browser, the code was running from a Linux server) in an office setting.

Each participant was able to achieve the correct clustering (5 clusters, each cluster contains images of a single digit) within 10 minutes in their first attempt with the fastest one achieving it in 3 minutes. We logged all atomic operations (i.e., "moves") made by the subjects. The number of operations needed to get to the final output varied from 5 to 24.

As we do not have a competing system to compare, we compare performance of different users who used different features of CommunityDiff to reach ground truth to validate the usefulness of different features. Four of the participants got to the final clustering output using only merging and manual labeling from the network diagram but on average (mean) they performed 20.25 operations which is much higher than participants using at least one other feature of CommunityDiff (on average 14.63 moves). Similarly, users who made use of the ensemble map made 13.86 moves on average (7 users), whereas those who did not, made 20.2 moves on average (5 users). We believe this demonstrates that the use of different features results in more efficient solutions. However, additional testing with more subjects and with features explicitly disabled would allow us to better isolate the key features that improve efficiency, but we leave this to future work.

Anecdotally, we also found that users perform much better in their second attempt. Two participants volunteered to perform the task a second time and they produced the final output within 5 and 7 operations (though part of this may be because they were familiar with the data). These attempts were not included in the study but they establish that with experience, users may learn more efficient solutions.

In a post study survey, users indicated that there was a benefit to the different views of the heatmap and liked the AL and flexibility of the tool. Some users felt the initial interface was overly complex and some actions were not intuitive enough. Some of these critical comments are: "Complexity needs to reduce, esp. the initial interface. Maybe put an 'advanced' section." , "Make the selection/moving gesture more intuitive, such as allow drag-and-drop." This is good advice for future iterations of the system. As a response to what they liked most using CommunityDiff, subject comments included: "Different views/metric to decide a partition," "Active learning (it allows the user to adaptively choose number of clusters)," and "Many different ways to manipulate nodes and communities."

7 DISCUSSION

We believe that CommunityDiff effectively supports the analytical process of community detection. The system provides explicit mechanisms to "jump" to a reasonable set of labels/partitions. This jump, however, is done with consideration of alternatives. By allowing for rapid comparison of different algorithmic outputs and ensembles, we believe that end-users can gain more confidence in their choice and are less likely to be biased. A second set of interfaces (the Network and Co-community View) provide a mechanism for quickly making decisions and corrections to complete the task. Underlying the whole process is a classification sub-system that leverages the end-users manipulations to continuously improve the partitioning. By subtly guiding the end-user by varying visual salience, CommunityDiff can guide the end-user to focus their attention

on those decisions that can make those most difference. Taken together these components solve different work "modes" for this task. Though they can function independently, they are designed to work well within this multi-step process.

The workflow described for community-detection is one example of many that require decision making among many alternative algorithms. While ensemble spaces can be applied in any situation where a metric is available for comparing algorithm output, the broader design of CommunityDiff may also be portable to other problems. Our hope is to continue to develop the idea of CommunityDiff and to test it in other contexts.

The current prototype of CommunityDiff does have some limitations. The most evident one is that it is most effective for smaller networks as response time of each interaction goes up with network size. Visual representation of very large networks is a hard problem in general. However, in the case of CommunityDiff, a more critical issue is that the algorithms we implement are computationally intensive (e.g., creating 1,000s of ensembles). We have not experimentally identified the "breaking point" for the tool but have found that with graph sizes of 300 or fewer nodes (an "ego-net" sized network), CommunityDiff can pre-compute all ensembles in under a minute. With networks of 1,500 nodes, the lag is more pronounced but once pre-computation completes the graph can be loaded and used at interactive speeds. It is worth noting that many real world networks sizes fall into this range (social networks with average 150 individuals, protein-protein interaction network with average 1,300 nodes [86]).

Disabling some of the high-cost community detection algorithms like spinglass (something one would do regardless of the visualization tool) allows use of larger graphs. Interestingly, a majority of computationally intensive operations can be parallelized. We currently support parallelization with respect to 10 threads in precomputation of the ensemble space heatmaps, which provides better response time for initial processing of a graph. It is worth noting that the ensemble technique and the classifier can be used on much larger graphs if we do not care about the response time or the visualization aspect.

Similarly, CommunityDiff is limited on how many communities it can handle efficiently. We have found that the interface becomes increasingly cluttered if we have more than 50 communities. However, as the number of communities in a real world network are usually far less compared to number of nodes, we usually see the scalability issues with respect to graph size pose more of a problem compared to scalability with respect to number of communities. For example, a social network with 150 nodes has around 4 communities, whereas protein–protein interaction networks with about 1,300 nodes has about 40 communities [86] (these are handled well by CommunityDiff). To further address the scaling problem, we have begun to design algorithms that adaptively fill in the ensemble space by proceeding from coarse to finer grid sizes dynamically.

Visually, CommunityDiff is constrained by limitations of force-directed node-link layouts, a modification of which we use in our network diagram. For networks with more than 1,500 nodes, the node-link diagram becomes too cluttered to be useful. A similar problem is present with co-community lists as with number of nodes these lists can become very long. However, an user is likely to work with only a few communities at once, so we provide a hide communities" button to hide communities and all corresponding nodes from the network diagram and the co-community lists to help the user focus on the task at hand with an uncluttered interface.

Though we believe the node-link diagram is an effective (and familiar) tool for visualization there are significant developments in group-structure visualization that may yield better results in the future [84]. This may be more critical if CommunityDiff begins to support other community structures. Currently, CommunityDiff works on undirected input graphs and generates only hard partitions (rather than overlapping communities). As discussed above, overlapping community detection algorithms are still inferior compared to the disjoint algorithms and unpopular in many

domains. Though this limits the tasks for which CommunityDiff is currently suitable for, we believe it can be adapted for additional inputs and outputs. For example, if we just use overlapping community detection algorithms as base algorithms of CommunityDiff, ensemble space heatmaps and co-community lists would retain most of their functionality. However, we would need efficient visual representation of overlapping communities in a network and a classifier that could work with overlapping communities (in AL) to make the whole pipeline work. Effective visualization of networks with overlapping communities is a hard problem in general, but solutions do exist for this task (e.g., [3]). Similar modifications may be necessary to the co-community lists with additional commands specifying how many communities a node should belong to. Finding a network classifier resulting in overlapping communities is a harder task. However, the semi-supervised classifier used in CommunityDiff generates probabilistic community assignments which could be used for assigning multiple communities to a node. The co-community lists are an area for possible improvement. The lists function well in our examples, but because the underlying data is networks and sets, alternative encodings may be more effective [4].

In the future, we also want to further evaluate the system by using multiple real-world networks and varied domains for the purpose of ecological validity. We intend to deploy CommunityDiff for others to use and hope to collect information on how it functions in real-world contexts. This type of evaluation would also help determine if certain features are more useful for certain kinds of networks or certain types of users and would give us a sense of performance of different algorithms on specific kinds of networks.

8 CONCLUSION

In this article, we present CommunityDiff, a novel tool supporting end-users in the construction of community-labeled networks. Though there are many network analysis tools and packages, one of the most common features—the efficient production of accurate community labels—is often lacking. The noisiness of networks, variable performance of community finding algorithms, and incompleteness of knowledge all factor into the challenging nature of this problem.

With CommunityDiff, we demonstrate a set of features that addresses many of these concerns. We show how an ensemble space can be created from competing algorithms that often outperforms single algorithms. Visualization of this space allows the end-user to make easy comparisons between different options. Focused visualizations further allow the end-user to view "votes" in a co-community graph, and allows the end-user to rapidly make decisions and correct system mistakes. As the user makes direct manipulation actions, CommunityDiff improves the partitions thus boosting labeling efficiency. By "nudging" the user, through visual cues, CommunityDiff can suggest high value feedback for the end-user to concentrate on. The overall system thus satisfies the needs of end-users by combining algorithms and visualizations. Though CommunityDiff focuses on community detection, we believe that many of the techniques and guidelines described here are portable to other data mining problems where end-user interactivity is valued and/or necessary.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their comments. This work is supported by the Intelligence Advanced Research Projects Activity (IARPA) via contract number D11PC20155. The U.S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

REFERENCES

[1] Lada A. Adamic and Natalie Glance. 2005. The political blogosphere and the 2004 U.S. election: Divided they blog. In *Proceedings of the 3rd International Workshop on Link Discovery (LinkKDD'05)*. ACM, New York, NY, 36–43. DOI: http://dx.doi.org/10.1145/1134271.1134277

[2] Eytan Adar. 2006. GUESS: A language and interface for graph exploration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*. ACM, New York, NY, 791–800. DOI: http://dx.doi.org/10.1145/1124772.1124889

[3] Basak Alper, Nathalie Henry Riche, Gonzalo Ramos, and Mary Czerwinski. 2011. Design study of linesets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2259–2267.

[4] Bilal Alsallakh, Luana Micallef, Wolfgang Aigner, Helwig Hauser, Silvia Miksch, and Peter Rodgers. 2014. Visualizing sets and set-typed data: State-of-the-art and future challenges. In *EuroVis - STARs*. R. Borgo, R. Maciejewski, and I. Viola (Eds.), The Eurographics Association, Geneva, Switzerland, 21. DOI: http://dx.doi.org/10.2312/eurovisstar.20141170

[5] Saleema Amershi, Maya Cakmak, W Bradley Knox, and Todd Kulesza. 2014. Power to the people: The role of humans in interactive machine learning. *AI Magazine* 35, 4 (2014), 105.

[6] Saleema Amershi, James Fogarty, and Daniel Weld. 2012. Regroup: Interactive machine learning for on-demand group creation in social networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'12)*. ACM, New York, NY, 21–30. DOI: http://dx.doi.org/10.1145/2207676.2207680

[7] Mihael Ankerst, Christian Elsen, Martin Ester, and Hans-Peter Kriegel. 1999. Visual classification: An interactive approach to decision tree construction. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99)*. ACM, New York, NY, 392–396. DOI: http://dx.doi.org/10.1145/312129.312298

[8] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, and others. 2009. Gephi: An open source software for exploring and manipulating networks. *ICWSM* 8 (2009), 361–362.

[9] Vladimir Batagelj and Andrej Mrvar. 1998. Pajek-program for large network analysis. *Connections* 21, 2 (1998), 47–57.

[10] Barry Becker, Ron Kohavi, and Dan Sommerfield. 2002. Visualizing the simple Baysian classifier. In *Information Visualization in Data Mining and Knowledge Discovery*. Usama Fayyad, Georges G. Grinstein, and Andreas Wierse (Eds.), Morgan Kaufmann Publishers Inc., San Francisco, CA, 237–249.

[11] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. 2008. *KNIME: The Konstanz Information Miner*. Springer, Berlin, 319–326. DOI: http://dx.doi.org/10.1007/978-3-540-78246-9_38

[12] P. M. Blau. 1977. *Inequality and Heterogeneity: A Primitive Theory of Social Structure*. Free Press, New York, NY. 77070272

[13] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (2008), P10008.

[14] Stephen P. Borgatti, Martin G. Everett, and Linton C. Freeman. 2002. Ucinet for Windows: Software for social network analysis. Analytic Technologies.

[15] Sebastian Bremm, Tatiana von Landesberger, Jürgen Bernard, and Tobias Schreck. 2011. Assisted descriptor selection based on visual comparative data analysis. In *Proceedings of the 13th Eurographics/IEEE - VGTC Conference on Visualization (EuroVis'11)*. The Eurographs Association, John Wiley & Sons, Ltd., Chichester, UK, 891–900. DOI: http://dx.doi.org/10.1111/j.1467-8659.2011.01938.x

[16] Ed Bullmore and Olaf Sporns. 2009. Complex brain networks: Graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience* 10, 3 (2009), 186–198.

[17] Matthew Burgess, Eytan Adar, and Michael Cafarella. 2016. Link-prediction enhanced consensus clustering for complex networks. *PLoS ONE* 11, 5 (2016), 1–23. DOI: http://dx.doi.org/10.1371/journal.pone.0153384

[18] Maya Cakmak, Crystal Chao, and Andrea L. Thomaz. 2010. Designing interactions for robot active learners. *IEEE Transactions on Autonomous Mental Development* 2, 2 (2010), 108–118.

[19] Doina Caragea, Dianne Cook, and Vasant G. Honavar. 2001. Gaining insights into support vector machine pattern classifiers using projection-based tour methods. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*. ACM, New York, NY, 251–256. DOI: http://dx.doi.org/10.1145/502512.502547

[20] Duen Horng Chau, Aniket Kittur, Jason I. Hong, and Christos Faloutsos. 2011. Apolo: Making sense of large network data by combining rich user interaction and machine learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'11)*. ACM, New York, NY, 167–176. DOI: http://dx.doi.org/10.1145/1978942.1978967

[21] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. 2004. Finding community structure in very large networks. *Physical Review E* 70, 6 (Dec 2004), 066111. DOI: http://dx.doi.org/10.1103/PhysRevE.70.066111

[22] Trevor F. Cox and Michael A. A. Cox. 2000. *Multidimensional Scaling*. CRC Press, Boca Raton, FL.

[23] Gabor Csardi and Tamas Nepusz. 2006. The igraph software package for complex network research. *InterJournal, Complex Systems* 1695, 5 (2006), 1–9.

[24] Johan Dahlin and Pontus Svenson. 2013. Ensemble approaches for improving community detection methods.*CoRR* abs/1309.0242 (2013). http://arxiv.org/abs/1309.0242

[25] Jianyong Dai and Jianlin Cheng. 2008. HMMEditor: A visual editing tool for profile hidden Markov model. *BMC Genomics* 9, Suppl 1 (2008), S8.

[26] Janez Demšar, Blaž Zupan, Gregor Leban, and Tomaz Curk. 2004. Orange: From experimental machine learning to interactive data mining. In *Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'04), Pisa, Italy, September 20–24, 2004.* Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi (Eds.), Springer, Berlin, 537–539. DOI: http://dx.doi.org/10.1007/978-3-540-30116-5_58

[27] C. Dunne and Ben Shneiderman. 2009. Improving graph drawing readability by incorporating readability metrics: A software tool for network analysts. *University of Maryland, HCIL Tech Report HCIL-2009-13* (2009/// 2009).

[28] Scott Emmons, Stephen Kobourov, Mike Gallant, and Katy Brner. 2016. Analysis of network clustering algorithms and cluster quality metrics at scale. *PLoS ONE* 11, 7 (07 2016), 1–18. DOI: http://dx.doi.org/10.1371/journal.pone.0159161

[29] Alex Endert, M. Shahriar Hossain, Naren Ramakrishnan, Chris North, Patrick Fiaux, and Christopher Andrews. 2014. The human is the loop: New directions for visual analytics. *Journal of Intelligent Information Systems* 43, 3 (2014), 411–435.

[30] Günes Erkan and Dragomir R. Radev. 2004. LexRank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research* 22, 1 (Dec. 2004), 457–479. http://dl.acm.org/citation.cfm?id=1622487.1622501

[31] Jerry Alan Fails and Dan R. Olsen, Jr. 2003. Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces (IUI'03).* ACM, New York, NY, 39–45. DOI: http://dx.doi.org/10.1145/604045.604056

[32] Rebecca Fiebrink, Perry R. Cook, and Dan Trueman. 2011. Human model evaluation in interactive supervised learning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'11).* ACM, New York, NY, 147–156. DOI: http://dx.doi.org/10.1145/1978942.1978965

[33] James Fogarty, Desney Tan, Ashish Kapoor, and Simon Winder. 2008. CueFlik: Interactive concept learning in image search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'08).* ACM, New York, NY, 29–38. DOI: http://dx.doi.org/10.1145/1357054.1357061

[34] Santo Fortunato and Marc Barthelemy. 2007. Resolution limit in community detection. *Proceedings of the National Academy of Sciences of the United States of America* 104, 1 (2007), 36–41.

[35] M. Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences* 99, 12 (2002), 7821–7826.

[36] Justin Grimmer and Gary King. 2011. General purpose computer-assisted clustering and conceptualization. *Proceedings of the National Academy of Sciences* 108, 7 (2011), 2643–2650. DOI: http://dx.doi.org/10.1073/pnas.1018067108

[37] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST'11).* ACM, New York, NY, 65–74. DOI: http://dx.doi.org/10.1145/2047196.2047205

[38] Mohammad Hamdaqa, Ladan Tahvildari, Neil LaChapelle, and Brian Campbell. 2014. Cultural scene detection using reverse Louvain optimization. *Science of Computer Programming* 95, P1 (Dec. 2014), 44–72. DOI: http://dx.doi.org/10.1016/j.scico.2014.01.006

[39] Derek Hansen, Ben Shneiderman, and Marc A. Smith. 2010. *Analyzing Social Media Networks with NodeXL: Insights from a Connected World.* Morgan Kaufmann, Burlington, Massachusetts.

[40] Jeffrey Heer and Danah Boyd. 2005. Vizster: Visualizing online social networks. In *Proceedings of the 2005 IEEE Symposium on Information Visualization (INFOVIS'05).* IEEE Computer Society, Washington, DC, 5–. DOI: http://dx.doi.org/10.1109/INFOVIS.2005.39

[41] Petter Holme, Mikael Huss, and Hawoong Jeong. 2003. Subnetwork hierarchies of biochemical pathways. *Bioinformatics* 19, 4 (2003), 532–538.

[42] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'99).* ACM, New York, NY, 159–166. DOI: http://dx.doi.org/10.1145/302979.303030

[43] Darko Hric, Richard K. Darst, and Santo Fortunato. 2014. Community detection in networks: Structural communities versus ground truth. *Physical Review E* 90, 6 (Dec. 2014), 062805. DOI: http://dx.doi.org/10.1103/PhysRevE.90.062805

[44] Mohsen Jamali and Martin Ester. 2009. TrustWalker: A random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09).* ACM, New York, NY, 397–406. DOI: http://dx.doi.org/10.1145/1557019.1557067

[45] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. 2011. Research directions in data wrangling: Visuatizations

and transformations for usable and credible data. *Information Visualization* 10, 4 (Oct. 2011), 271–288. DOI: http://dx.doi.org/10.1177/1473871611415994

[46] Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz. 2010. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'10)*. ACM, New York, NY, 1343–1352. DOI: http://dx.doi.org/10.1145/1753326.1753529

[47] Ashish Kapoor, Bongshin Lee, Desney Tan, and Eric Horvitz. 2012. Learning to learn: Algorithmic inspirations from human problem solving. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*. AAAI Press, Palo Alto, California, 1571–1577.

[48] Stephen G. Kobourov. 2012. Spring Embedders and Force Directed Graph Drawing Algorithms. (2012). http://arxiv.org/abs/1201.3011 cite arxiv:1201.3011Comment: 23 pages, 8 figures.

[49] Todd Kulesza, Simone Stumpf, Weng-Keen Wong, Margaret M. Burnett, Stephen Perona, Andrew Ko, and Ian Oberst. 2011. Why-oriented end-user debugging of naive Bayes text classification. *ACM Transactions on Interactive Intelligent Systems* 1, 1 (2011), 2.

[50] Andrea Lancichinetti and Santo Fortunato. 2012. Consensus clustering in complex networks. *CoRR* abs/1203.6093 (2012). http://dblp.uni-trier.de/db/journals/corr/corr1203.html.

[51] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Physical Review E* 78, 4 (Oct. 2008), 046110. DOI: http://dx.doi.org/10.1103/PhysRevE.78.046110

[52] Gregor Leban, Blaž Zupan, Gaj Vidmar, and Ivan Bratko. 2006. Vizrank: Data visualization guided by machine learning. *Data Mining and Knowledge Discovery* 13, 2 (2006), 119–136.

[53] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of IEEE* 86, 11 (Nov. 1998), 2278–2324. DOI: http://dx.doi.org/10.1109/5.726791

[54] Mingwei Leng, Yukai Yao, Jianjun Cheng, Weiming Lv, and Xiaoyun Chen. 2013. Active semi-supervised community detection algorithm with label propagation. In *Database Systems for Advanced Applications*. Lecture Notes in Computer Science, vol. 7826. Springer, Berlin, 324–338. DOI: http://dx.doi.org/10.1007/978-3-642-37450-0_25

[55] Zhicheng Liu and John T. Stasko. 2010. Mental models, visual reasoning and interaction in information visualization: A top-down perspective. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 999–1008. http://dblp.uni-trier.de/db/journals/tvcg/tvcg16.html#LiuS10.

[56] Richard Maclin and David W. Opitz. 1999. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* (JAIR) 11 (1999), 169–198.

[57] Sofus A. Macskassy. 2009. Using graph-based metrics with empirical risk minimization to speed up active learning on networked data. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. ACM, New York, NY, 597–606. DOI: http://dx.doi.org/10.1145/1557019.1557087

[58] T. May, A. Bannach, J. Davey, T. Ruppert, and J. Kohlhammer. 2011. Guiding feature subset selection with an interactive visualization. In *Proceedings of the 2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, New York, NY, 111–120. DOI: http://dx.doi.org/10.1109/VAST.2011.6102448

[59] Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. 2011. Generalized louvain method for community detection in large networks. In *Proceedings of ISDA*. IEEE, New York, NY, 88–93. http://dblp.uni-trier.de/db/conf/isda/isda2011.html.

[60] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. 2006. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*. ACM, New York, NY, 935–940. DOI: http://dx.doi.org/10.1145/1150402.1150531

[61] Rada F. Mihalcea and Dragomir R. Radev. 2011. *Graph-based Natural Language Processing and Information Retrieval* (1st ed.). Cambridge University Press, New York, NY.

[62] David Mimno and Moontae Lee. 2014. Low-dimensional embeddings for interpretable anchor-based topic inference. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1319–1328.

[63] M. E. J. Newman. 2003. Fast algorithm for detecting community structure in networks. *Physical Review E* 69 (September 2003). http://arxiv.org/abs/cond-mat/0309508

[64] Mark E. J. Newman. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America* 103, 23 (2006), 8577–8582.

[65] Kayur Patel, Steven M. Drucker, James Fogarty, Ashish Kapoor, and Desney S. Tan. 2011. Using multiple models to understand data. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI'11)*. AAAI Press, Palo Alto, CA, 1723–1728. DOI: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-289

[66] Leto Peel, Daniel B. Larremore, and Aaron Clauset. 2017. The ground truth about metadata and community detection in networks. Science Advances 3, 5 (2017). DOI: http://dx.doi.org/10.1126/sciadv.1602548

[67] Peter Pirolli and Stuart Card. 2005. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of International Conference on Intelligence Analysis*. McLean, CA, 6.

[68] Pascal Pons and Matthieu Latapy. 2004. Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications* 10 (2004), 284–293.
[69] Reid Porter, James Theiler, and Don Hush. 2013. Interactive machine learning in data exploitation. *Computing in Science & Engineering* 15, 5 (2013), 12–20.
[70] Arnau Prat-Pérez, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. 2014. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd International Conference on World Wide Web (WWW'14)*. ACM, New York, NY, 225–236. DOI: http://dx.doi.org/10.1145/2566486.2568010
[71] Usha N. Raghavan, Rèka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E* 76, 3 (2007), 036106. DOI: http://dx.doi.org/10.1103/PhysRevE.76.036106
[72] Joerg Reichardt and Stefan Bornholdt. 2006. Statistical mechanics of community detection. *Physical Review E* 74 (2006), 016110. http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0603718.
[73] Lior Rokach. 2010. Ensemble-based classifiers. *Artificial Intelligence Review* 33, 1 (2010), 1–39. DOI: http://dx.doi.org/10.1007/s10462-009-9124-7
[74] Martin Rosvall, Daniel Axelsson, and Carl T. Bergstrom. 2009. The map equation. *The European Physical Journal Special Topics* 178, 1 (2009), 13–23.
[75] Martin Rosvall and Carl T. Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences of the United States of America* 105, 4 (2008), 1118–1123. DOI: http://dx.doi.org/10.1073/pnas.0706851105 arXiv:http://www.pnas.org/content/105/4/1118.full.pdf+html.
[76] Burr Settles. 2010. Active learning literature survey. *University of Wisconsin, Madison* 52, 55–66 (2010), 11.
[77] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S. Baliga, Jonathan T. Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. 2003. Cytoscape: A software environment for integrated models of biomolecular interaction networks. *Genome Research* 13, 11 (2003), 2498–2504.
[78] Hossam Sharara, Awalin Sopan, Galileo Namata, Lise Getoor, and Lisa Singh. 2011. G-PARE: A visual analytic tool for comparative analysis of uncertain graphs. In *Proceedings of the 2011 IEEE Conference on VAST*. IEEE, New York, NY, 61–70. http://dblp.uni-trier.de/db/conf/ieeevast/ieeevast2011.html#ShararaSNGS11.
[79] Michael Shilman, Desney S. Tan, and Patrice Simard. 2006. CueTIP: A mixed-initiative interface for correcting handwriting errors. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST'06)*. ACM, New York, NY, 323–332. DOI: http://dx.doi.org/10.1145/1166253.1166304
[80] Simone Stumpf, Vidya Rajaram, Lida Li, Weng-Keen Wong, Margaret Burnett, Thomas Dietterich, Erin Sullivan, and Jonathan Herlocker. 2009. Interacting meaningfully with machine learning systems: Three experiments. *International Journal of Human-Computer Studies* 67, 8 (Aug. 2009), 639–662. DOI: http://dx.doi.org/10.1016/j.ijhcs.2009.03.004
[81] Justin Talbot, Bongshin Lee, Ashish Kapoor, and Desney S. Tan. 2009. EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'09)*. ACM, New York, NY, 1283–1292. DOI: http://dx.doi.org/10.1145/1518701.1518895
[82] Lei Tang and Huan Liu. 2010. Community detection and mining in social media. *Synthesis Lectures on Data Mining and Knowledge Discovery* 2, 1 (2010), 1–137.
[83] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, 2579–2605 (2008), 85.
[84] Corinna Vehlow, Fabian Beck, and Daniel Weiskopf. 2015. The state of the art in visualizing group structures in graphs. In *Proceedings of the Eurographics Conference on Visualization (EuroVis) - STARs*, R. Borgo, F. Ganovelli, and I. Viola (Eds.), The Eurographics Association, Geneva, Switzerland. DOI: http://dx.doi.org/10.2312/eurovisstar.20151110
[85] Malcolm Ware, Eibe Frank, Geoffrey Holmes, Mark Hall, and Ian H. Witten. 2002. Interactive machine learning: Letting users build classifiers. *International Journal of Human-Computer Studies* 56, 3 (March 2002), 281–292.
[86] Jaewon Yang and Jure Leskovec. 2014. Overlapping communities explain core–Periphery organization of networks. *Proceedings of the IEEE* 102, 12 (2014), 1892–1902.
[87] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.
[88] W. W. Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research* 33 (1977), 452–473.
[89] Kaiyu Zhao, Matthew O. Ward, Elke A. Rundensteiner, and Huong N. Higgins. 2014. LoVis: Local pattern visualization for model refinement. *Computer Graphics Forum* 33, 3 (2014), 331–340.
[90] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. 2003. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on International Conference on Machine Learning (ICML'03)*. AAAI Press, Palo Alto, CA, 912–919.

Received December 2015; revised September 2016; accepted January 2017