Efficient Detection of Communities in Biological Bipartite Networks

Paola Pesántez-Cabrera and Ananth Kalyanaraman

Abstract—Methods to efficiently uncover and extract community structures are required in a number of biological applications where networked data and their interactions can be modeled as graphs, and observing tightly-knit groups of vertices ("communities") can offer insights into the structural and functional building blocks of the underlying network. Classical applications of community detection have largely focused on unipartite networks—i.e., graphs built out of a single type of objects. However, due to increased availability of biological data from various sources, there is now an increasing need for handling heterogeneous networks which are built out of multiple types of objects. In this paper, we address the problem of identifying communities from biological bipartite networks—i.e., networks where interactions are observed between two different types of objects (e.g., genes and diseases, drugs and protein complexes, plants and pollinators, hosts and pathogens). Toward detecting communities in such bipartite networks, we make the following contributions: i) (metric) we propose a variant of bipartite modularity; ii) (algorithms) we present an efficient algorithm called biLouvain that implements a set of heuristics toward fast and precise community detection in bipartite networks (https://github.com/paolapesantez/biLouvain); and iii) (experiments) we present a thorough experimental evaluation of our algorithm including comparison to other state-of-the-art methods to identify communities in bipartite networks. Experimental results show that our biLouvain algorithm identifies communities that have a comparable or better quality (as measured by bipartite modularity) than existing methods, while significantly reducing the time-to-solution between one and four orders of magnitude.

Index Terms—Heterogeneous biological data, bipartite networks, graph algorithms, community detection, bipartite modularity.

1 INTRODUCTION

The increasing identification and characterization of genes, protein complexes, diseases, and drugs have highlighted a need to incorporate heterogeneity while analyzing complex biological data. A heterogeneous network is composed of multiple types of objects. Identifying "community" structures that transcend data boundaries in such a network could provide new insights that may not be readily visible by examining only a specific data type in isolation. For instance, identifying a group of genes that have been implicated across a set of diseases could possibly reveal hidden links among seemingly different diseases or disease conditions, and in the process help identify new drugs and therapies [1]. Similarly, identifying active gene clusters across different subsets of brain regions could provide new insights into brain function [2].

Graph-theoretic representations offer a natural way to model networks built out of heterogeneous data. This work focuses on bipartite networks as a way to model the interplay between two different data types. Bipartite networks are those which have two types of vertices such that edges exist only between vertices of the two different types. Bipartite networks are called unweighted when the edges between vertices are either present or not. Alternatively, if edges are given different values representing the strength of the associations among vertices the bipartite network is called weighted. Some examples of biological bipartite networks include (but are not limited to) gene-disease [3], gene-drug [4], plants-pollinators [5], and host-pathogen [6].

While the idea of modeling heterogeneity as a graph problem is not necessarily new per se, algorithm development efforts have been more recent [7], [8]. Once modeled as a bipartite network, we can view the problem of identifying cluster structures between the two different data types as a problem of community detection in bipartite networks. For example, in the case of the gene-drug network we could identify groups of drugs that might inhibit or otherwise modulate groups of genes. We also could detect groups of genes that are suitable for drug repurposing but may not currently have a drug targeting them. Similarly, for plant-pollinator networks we could identify groups of pollinators that limit or promote the establishment and persistence of plant species.

Given a graph, the goal of community detection is to partition the set of vertices into "communities" such that vertices that are assigned to the same community have a higher density of edges among them than to the vertices in the rest of the network. Community detection can be used to reveal hidden substructures within real world networks, without any known prior knowledge on either the number or sizes of the output communities.

Community detection is a well studied problem in literature [9]. However, the treatment of the problem on bipartite networks has been sparse. Because edges connect vertices of two different types, the classical definition of communities [10] does not directly apply. Projection-based approaches typically result in loss of the bipartite structural information [11]. Instead, the notion of communities needs to be redefined so that a community of vertices of one type is formed on the basis of the strength of its shared connections to vertices of the other type (as shown in Fig. 1).

To unveil important new associations, evaluation of the goodness of a community-wise division of a bipartite net-
work also becomes critical. To this end, a measure called bipartite modularity can be defined by extending the classical measure for unipartite networks [10]. Modularity is a statistical measure which calculates the difference between the observed fraction of intra-community edges to an expected fraction in an equivalent random graph — i.e., null model. Modularity measures have a few limitations such as the resolution limit and misidentification [12]. However, in practice a higher value of modularity suggests a clearer community structure within the underlying network. Modularity optimization is an NP-Hard problem [13]; however, a number of efficient heuristics are used in practice [9]. For bipartite networks, multiple formulations of bipartite modularity have been proposed [14], [15], [16], [17]. Of these, Murata’s definition is known to overcome some of the limitations of the other definitions (see Sections 2 and 4).

1.1 Contributions

We present a direct, efficient method (biLouvain) to optimize our modified version of Murata’s modularity (Murata+). Specifically, the main contributions of this paper are:

- **Murata+ modularity**: We initially consider the classical definition of Murata’s modularity [16] and discuss its suitability for community detection in Section 4. In the process of evaluation, we identify an inconsistency in the classical formula and subsequently propose a simple variant, which we call Murata+.

- **biLouvain algorithm**: We present efficient algorithmic heuristics for optimized detection of bipartite communities using the Murata+ modularity (Section 5). Our approach extends the Louvain algorithm [18], which is one of the most efficient and widely used heuristics for unipartite networks. Consequently, we call our algorithm biLouvain. As part of our algorithm, we provide ways to calculate the modularity gain resulting from vertex migrations in a bipartite network — a step that constitutes the core of the Louvain heuristic.

- **Experimental evaluation**: We present a thorough experimental evaluation of our algorithm using both synthetic and real-world networks and in comparison to other state-of-the-art methods (Section 6). Experimental results show that our algorithm identifies communities that have a comparable or better quality (as measured by the Murata+ bipartite modularity) than existing methods, while significantly reducing the time-to-solution between one and four orders of magnitude.

A preliminary version of the work presented on this paper has been published in [19]. The rest of this paper is organized as follows. Section 2 describes the key related work for community detection in the context of bipartite networks. Section 3 presents the basic notation and terminology for use throughout the paper. Section 4 explores the applicability and extension of Murata’s modularity as a way to determine the quality of a particular division of a network into communities. Section 4 describes in detail the algorithm we have developed — i.e., biLouvain algorithm, for detection of communities in bipartite networks. Section 6 presents the experimental evaluation of our algorithm on different real and synthetic networks. Section 7 concludes the paper.
Table 1 summarizes the conceptual differences among these methods, relative to our algorithm bILouwain that is proposed in this paper.

### 3 Basic Notation and Terminology

We will use \( G(V_1 \cup V_2, E, \omega) \) to denote an undirected bipartite graph. Here, \( V_1 \) and \( V_2 \) represent the two sets of vertices, and \( E \) represents the set of edges such that each edge \( e = (i, j) \in E \) is such that \( i \in V_1 \) and \( j \in V_2 \). Each edge \( e \) is also associated with a numerical weight \( \omega(e) \). We will assume that the edge weights are non-negative values that reflect the strength of the relation between any two vertices. The sum of the weights of all edges incident on a vertex \( i \) is said to be its weighted degree (denoted by \( \gamma(i) \)). Additionally, we use the term binary networks to describe networks whose edges are unweighted. In such cases, all edges that exist are assumed to have a unit weight.

Let \( n_1 \) and \( n_2 \) denote the number of vertices in \( V_1 \) and \( V_2 \) respectively, and \( m \) denote the number of edges. Let \( M = \sum_e \omega(e) \). For sake of consistency, we use \( i \)'s to denote vertices in \( V_1 \), and \( j \)'s for vertices in \( V_2 \).

A community represents a subset of either \( V_1 \) or \( V_2 \). For ease of exposition, we use \( C \)'s to denote communities taken from \( V_1 \) and \( D \)'s to denote communities taken from \( V_2 \). Let

\[
P_1 = \{C_1, C_2, \ldots, C_k\} \text{ denote a set of communities in } V_1 \text{ such that it represents a partitioning of } V_1. \text{ Similarly, let } P_2 = \{D_1, D_2, \ldots, D_k\} \text{ represent a set of communities in } V_2 \text{ such that it represents a partitioning of } V_2. \text{ Throughout this paper, we assume } k_1 \text{ need not be equal to } k_2.

In our algorithmic discourse, we denote the present community containing any vertex \( i \in V_1 \) as \( C(i) \), and the present community containing any vertex \( j \in V_2 \) as \( D(j) \).

### 4 Bipartite Modularity

#### 4.1 Murata’s Bipartite Modularity

In what follows, we describe Murata’s modularity [16] although using our own notation for convenience.

Given a pair of communities, \( C \in P_1 \) and \( D \in P_2 \), and let \( e \) denote any edge that connects a vertex in community \( C \) with a vertex in community \( D \). Consequently, we define:

\[
E_{C,D} = \frac{1}{2M} \sum_e \omega(e) \tag{1}
\]

Note that by this definition, \( E_{C,D} = E_{D,C} \). Also note that the term \( M \) corresponds to the sum of the weighted degree of all vertices — i.e., \( M = \sum_i \gamma(i) \). We use the term \( A_C \) to denote the fraction of this term contributed by a given community \( C \).

\[
A_C = \frac{1}{2M} \sum_D E_{C,D} \tag{2}
\]

Furthermore, we define the co-cluster mate of a community \( C \) to be a community \( D \in P_2 \) to which \( C \) has the most concentration of its edges — i.e.,

\[
\psi(C) = \arg\max_D (E_{C,D}) \tag{3}
\]

1. The factor 2 is a result of each edge getting counted twice — once in each direction.
We refer to the ordered tuple \( \langle C, \psi(C) \rangle \) as a co-cluster. Fig. 2 illustrates this concept. Similarly, the co-cluster mate of a community \( D \) is defined as follows:

\[
\psi(D) = \arg\max_C (\mathcal{E}_{D,C})
\]  

(4)

The ordered tuple \( \langle D, \psi(D) \rangle \) is also a co-cluster.

**Definition 1.** Given a bipartite graph \( G(V_1 \cup V_2, E, \omega) \), and two sets of communities \( P_1 \) in \( V_1 \) and \( P_2 \) in \( V_2 \), Murata’s bipartite modularity \( Q_B \) is defined as follows [16]:

\[
Q_B = \sum_C (\mathcal{E}_{C,\psi(C)} - \mathcal{A}_C \times \mathcal{A}_{\psi(C)}) + \sum_D (\mathcal{E}_{D,\psi(D)} - \mathcal{A}_D \times \mathcal{A}_{\psi(D)})
\]  

(5)

Intuitively, Murata’s modularity is calculated by pairing every community from one side with a community on the other side that it has maximum connections to. The first term inside the two summations in Eqn. 5 corresponds to the fraction of such “intra-co-cluster” edges. The second term inside each of the summations is the expected fraction of such edges in a randomly generated bipartite graph with an identical vertex degree sequence. As in Newman’s modularity [10] for unipartite networks, the idea is to encourage a partitioning that maximizes intra-co-cluster edges while discouraging a partitioning that groups unrelated vertices.

### 4.2 Murata+: Proposed Bipartite Modularity

From Eqns. 3, 4, and 5 we make the following two observations:

**Observation 1.** If a community \( C \) picks a community \( D \) as its co-cluster mate (by Eqn. 3), \( D \) need not necessarily pick \( C \) (by Eqn. 4).

**Observation 2.** The statistical terms \( \mathcal{A}_C \times \mathcal{A}_{\psi(C)} \) and \( \mathcal{A}_D \times \mathcal{A}_{\psi(D)} \) are used in the final modularity calculation of Eqn. 5, but they are not used while picking the co-cluster mates (Eqns. 3 and 4).

Observation 1 implies that the co-cluster relationship is nonsymmetric. This relaxation is necessary to avoid a method-enforced one-to-one mapping between communities and their co-cluster mates. However, the relaxation could also lead to an undesirable effect of a potential lack of cohesion between communities and their co-cluster mates. For bipartite networks, we typically attempt to explain the grouping of a community based on its co-cluster. While one-to-one mapping would make it too restrictive for this purpose, it is also important not to make it excessively many-to-many. For most practical inputs, a middle ground is more desirable where the expected mapping remains closer to an one-to-one mapping. For instance, we can expect a strong (if not strict) two-way correlation between a set of genes and the set of diseases they impact.

Observation 2 indicates a matter of inconsistency because a community \( C \) picks a co-cluster mate solely based on the positive term, while the final modularity is calculated taking into account also the negative term. At best, this can lead to an overestimated value for modularity \( Q_D \), when a co-cluster mate is selected. More importantly, we argue that the negative term is in fact essential as otherwise it could potentially lead to a scenario where a community and its co-cluster mate could be of vastly different sizes. This is shown in Fig. 3.

To address the above inconsistency issue within the classical definition, we propose a variant of bipartite modularity by simply redefining the co-cluster mate selection criterion to include the negative term:

\[
\psi(C) = \arg\max_D (\mathcal{E}_{C,D} - \mathcal{A}_C \times \mathcal{A}_D)
\]  

(6)

Similarly, a co-cluster mate of a community \( D \) is defined as follows:

\[
\psi(D) = \arg\max_C (\mathcal{E}_{D,C} - \mathcal{A}_D \times \mathcal{A}_C)
\]  

(7)

The modularity expression is the same as Eqn. 5. It should be clear that this revised definition would make the choice of co-cluster mates consistent with the modularity calculation — i.e., fixing the problem with Observation 2.

In addition, the revised definition preserves the nonsymmetry property (Observation 1), while being better positioned than the classical definition to encourage an one-to-one mapping, wherever possible, without strictly enforcing it. This is because of the reduced degree of freedom that a community is likely to have (with the introduction of the negative term) when selecting its co-cluster mate.

Henceforth, we refer to our revised version of the Murata’s modularity as Murata+ (Eqns. 5, 6, and 7), and use it as our primary objective function.

### 5 biLouvain: An Algorithm for Bipartite Community Detection

In this section, we present our biLouvain algorithm for community detection in bipartite networks. We adapt the widely used Louvain heuristic [18] to work for bipartite networks. While biLouvain follows the same algorithmic template provided by Louvain, it differs in the objective function (by using Murata+) and in the way all the key steps are computed, as will be elaborated below.

Like the original algorithm, biLouvain is a multi-phase, multi-iterative algorithm, where each phase is a series of
To one another, and vertices and the communities they belong to at the end of the phase. The figure shows the biLouvain on a net modularity gain function. The main steps of the algorithm are as follows (see Fig. 4):

1) Given an input bipartite graph \( G(V_1 \cup V_2, E, \omega) \), initialize a set of \( n_1 + n_2 \) communities, where \( n_1 = |V_1| \) and \( n_2 = |V_2| \), by placing each vertex in its own community.
2) At every iteration, all vertices in \( V_1 \) and \( V_2 \) are scanned linearly (in an arbitrary order). For each vertex \( i \):
   a) Acquire a list of its candidate communities to which it can potentially migrate to;
   b) Evaluate the modularity gain that would result from the scenario of migrating \( i \) to each of the candidate communities;
   c) Finally, migrate vertex \( i \) to a candidate community that maximizes the modularity gain, only if such gain is positive (otherwise, no change).
3) A phase ends when the net modularity gain achieved between two consecutive iterations is negligible — i.e., below a certain threshold \( \tau_v \), which we refer to as the iteration cutoff.
4) Once a phase terminates, a new graph \( G'(V_1' \cup V_2', E', \omega') \) is generated through a compaction step, which collapses each community to a vertex, and edges and their weights in the new graph corresponds to the strength of edges connecting any two communities.
5) The new compacted graph is input to the next phase (step 1). The algorithm terminates when any two consecutive phases result in a negligible modularity gain, defined by a threshold \( \tau_p \), which we call phase cutoff.

In what follows, we describe how the key steps that are impacted by the bipartite structure are implemented in biLouvain. More specifically, the classical definition for candidate communities and their computation (step (2a)), the expression for calculating the modularity gain resulting from a vertex migration, and the algorithm to calculate the modularity gain (step (2b))—all of these need to be defined taking into account the bipartite structure.

5.1 Computing Candidate Communities

A candidate community of a vertex is a community to which that vertex can potentially migrate at any given iteration of the biLouvain algorithm, with a realistic chance of accruing a positive modularity gain.

For ease of exposition, we explain the process of computing candidate communities from the point of view of a vertex \( i \) in \( V_1 \). It should be easy to see that the same approach works for any vertex \( j \in V_2 \).

For a given vertex \( i \in V_1 \), let \( \Gamma(i) \) denote the set of neighbors of \( i \) in \( V_2 \) — i.e.,

\[ \Gamma(i) = \{ j \mid (i, j) \in E \} \]

Let \( \Gamma'(i) \) denote the set of vertices in \( V_1 \) that the neighbors of \( i \) are connected to.

\[ \Gamma'(i) = \{ k \mid (k, j) \in E, \text{where } j \in \Gamma(i) \} \]

Consequently, the set of candidate communities for \( i \), \( \text{Cand}(i) \) is given by:

\[ \text{Cand}(i) = \bigcup_{k \in \Gamma'(i)} C(k) \]

Intuitively, a vertex \( i \in V_1 \) can only migrate to communities in \( V_1 \), within which it has at least one 2-hop neighbor (i.e., via its vertex neighbors in \( V_2 \)). Moving to any other community in \( V_1 \) (i.e., not in this candidate set) will result in a decrease in modularity.

5.2 Calculating Modularity Gain

In the case of unipartite networks [18], calculating the expected modularity gain resulting from moving a vertex from one community to another can be executed in constant time if appropriate data structures are maintained. In the case of bipartite networks, this is not the same because of the following lemma and corollary (as shown in Fig. 5):

**Lemma 5.1.** If vertex \( i \) moves from \( C \) to \( C' \), then the choice of co-cluster mates for either community could change.

**Proof:** The migration of vertex \( i \) from \( C \) to \( C' \) affects the values \( \tilde{E}_C, \tilde{E}_{C'} \), \( \tilde{A}_C \), and \( \tilde{A}_{C'} \), for any \( D \), in Eqn. 6.

Note that the lemma applies to migrations of vertices in \( V_2 \) as well.

Define the community set \( N(C) \) as follows:

\[ N(C) = \bigcup_{j \in \Gamma(i)} D(j) \]

The above lemma leads to the following corollary:
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2017.2765319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

Fig. 5. Illustration of vertex \( i \) evaluating its migration from community \( C \) to community \( C' \). If this move were to be executed, the co-cluster mate choices for both \( C \) and \( C' \) (presently, \( \psi(C) = D \) and \( \psi(C') = D' \)) could potentially change (Lemma 5.1). Furthermore, the co-cluster mate choice for an arbitrary community \( D_x \) which has at least one edge incident from either \( C \) or \( C' \) could also potentially change (Corollary 5.2). Consequently, a question mark is used to show scenarios where the corresponding co-cluster mate assignments need to be re-evaluated.

**Corollary 5.2.** If vertex \( i \) moves from \( C \) to \( C' \), then the co-cluster mate choices for any of the communities in \( N(C) \) or in \( N(C') \) could potentially change.

**Proof:** Due to the migration of \( i \) from \( C \) to \( C' \), any community \( D_x \) of \( V_2 \) to which either of these two communities is connected (see Fig. 5) could potentially change its co-cluster mate preference with changes to the right hand side values of its corresponding \( \psi(D_x) \) calculation (i.e., inside Eqn. 7). This implies that \( D_x \in N(C) \cup N(C') \).

It should be intuitively clear why the lemma and the corollary hold. In short, the two equations for co-cluster mate choices (Eqns. 6 and 7) depend on both the positive and the negative terms, and either of those two terms could change for the communities covered in Lemma 5.1 and Corollary 5.2 as a result of \( i \) moving out of \( C \) into \( C' \). It should also be clear that the co-cluster mate choices for no other communities in the rest of the graph are affected by this move.

Based on the above lemma and corollary, we constitute a set containing all affected communities from \( i \)'s migration from \( C \) to \( C' \) — i.e., \( S = \{C, C'\} \cup N(C) \cup N(C') \). Subsequently, we compute the overall net modularity gain for this single vertex move as follows:

We calculate the contribution of a community \( D \) and its co-cluster mate \( \psi(D) \) to the summation in the modularity Eqn. 5 as follows:

\[
 f(D, \psi(D)) = \varepsilon_{D, \psi(D)} - A_D \times A_{\psi(D)}
\]

Consequently, the change in a community \( D \)'s contribution to the overall modularity \( Q_B \) due to the change in its co-cluster mate (from \( \psi(D) \) to \( \psi(D)' \) imposed by vertex \( i \)'s move) is given by:

\[
 \Delta'Q_B(K) = f(D, \psi(D)) - f(D, \psi(D)')
\]

Finally, the overall modularity gain is given by:

\[
 \Delta Q_B = \sum_{K \in S} \Delta'Q_B(K)
\]

This modularity gain is calculated for each possible vertex move into one of its candidate communities; and finally, vertex \( i \) is moved to that candidate community which maximizes the gain (assuming it is positive).

### 5.3 Complexity Analysis

The run-time within an iteration is dominated by the time taken to calculate the modularity gain for all the vertices (Section 5.2). In our implementation we keep efficient data structures to enable us to calculate each community’s contribution to the overall modularity, in constant time. Given this, the worst-case run-time complexity for calculating the maximum modularity gain for a given vertex \( i \) at any given iteration is \( O(n_1 \times n_2) \)—this is for the worst-case scenario of all \( C \) communities connected to all \( D \) communities.

As all vertices are linearly scanned within each iteration, the worst-case run-time complexity is \( O((n_1 + n_2)n_1n_2) \) per iteration—which makes our exact algorithm a cubic algorithm. In practice; however, we can expect inputs to be sparse—that would imply a quadratic behavior in the initial iterations; but as the algorithm progresses, the number of communities can only shrink and along with it, also the run-time per iteration.

### 5.4 Performance heuristics

In what follows, we present a collection of heuristics aimed at improving the performance of the biLouvain algorithm in practice.

#### 5.4.1 Vertex Ordering

In the biLouvain algorithm, the order in which vertices are processed could potentially impact the performance of the algorithm and the quality of community-wise partitioning to varying degrees, depending on the input and on the ordering scheme used.

To understand the impact of vertex ordering on performance, note that the community assignment made for a vertex on one partition (say \( V_1 \)) at any given iteration is dependent on the community states of its neighboring vertices in the other partition (\( V_2 \)), and also on the community states in the same partition (for selecting candidate communities). Also note that initially the number of communities on each partition is equal to its number of vertices. When coupled together, these observations imply that if vertex decisions are all made, say sequentially, within one partition prior to the other partition, then the time taken for processing vertices on the first partition is likely to be significantly higher than for the vertices in the second partition during the same iteration. However, this performance impact is expected to diminish in the later iterations of the algorithm as communities get larger, thereby shrinking the number of communities. Consequently, vertex ordering is likely to have an effect on the algorithm’s performance.

As for quality, the impact of vertex ordering is likely to be relatively less. It can be expected that for real world inputs with well-defined community structures, the state of communities typically converges faster in the first few iterations of the algorithm, while largely remain stable in the later iterations. However, the final output quality could still differ based on the vertex ordering used.

To evaluate this quality-time tradeoff imposed by vertex ordering, we implement these vertex ordering schemes:

1) **Sequential:** Within each iteration, the vertices in one partition (say, \( V_1 \)) are all processed (in some arbitrary order) prior to vertices in the other partition.
2) **Alternate:** Within each iteration, the processing of vertices from the two partitions is interleaved — i.e., alternating between the two partitions, until one of the partitions is exhausted at which point the algorithm defaults to the sequential mode to cover the remaining vertices of the larger partition.

3) **Random:** Within each iteration, the order of processing vertices in \( V_1 \cup V_2 \) is randomized. By fixing the random seed, one can ensure that the output remains deterministic across multiple runs on the same input.

Another contributing factor to dictate an ordering scheme’s impact on performance is the constituent vertex partition sizes. If the two partitions are of skewed sizes (e.g., \( n_1 \ll n_2 \)) then the Random scheme is expected to have an advantage in run-time over the other two schemes, whereas if the two partitions have comparable sizes, then all three schemes can be expected to behave similarly.

### 5.4.2 Structural Properties

The performance of the algorithm can be further improved by observing and taking advantage of certain structural motifs (properties) within bipartite networks and their relation to the modularity expression. Such structural motifs can include simple topological features such as a star (i.e., vertex \( i \) in \( V_1 \) is connected to \( k \) vertices in \( V_2 \)), a chain (i.e., a linked list) inside a bipartite network, and common co-cluster mate (i.e., communities \( C_1, C_2 \in P_1 \) have community \( D \in P_2 \) assigned as their co-cluster mate), or more complex attributes such as a k-core or inexact versions of chains and stars embedded within a larger subgraph.

The idea is that, based on some provable properties, the input graph can be preprocessed so as to detect such motifs and compact them into their corresponding community structures (as dictated by lemmas). This idea would reduce the number of vertices to be processed, which has a direct impact on the overall work of the biLouvain algorithm.

In this paper, we prove three such properties: i) for the simple star case (Fig. 12(a); Appendix A.1), ii) for the simple chain case (Fig. 12(b); Appendix A.2), and iii) for the common co-cluster mate case (Fig. 6).

Although all these properties can be explored conjointly (i.e., they complement one another), we found their effects on performance to be varying. More specifically, based on our tests, we found that identifying simple stars and chains reduces the number of input vertices only by a factor of about 8%. While such an improvement is not necessarily insignificant in itself, relatively, our experimentation with the common co-cluster mate property showed consistently much larger improvements—in some cases, by even more than three orders of magnitude, as will be discussed in Section 6.3.2. Consequently, we delve into the details of the common co-cluster mate property in what follows; and defer the proofs and discussions corresponding to the other two properties (star and chain) to Appendix A.

The **Fuse heuristic:** Let \( C_1 \) and \( C_2 \) be two communities from vertex set \( V_1 \), and \( D \) be a community from vertex set \( V_2 \), such that \( D \) is a co-cluster mate for both \( C_1 \) and \( C_2 \) (i.e., \( \psi(C_1) = D, \psi(C_2) = D \)), and (without loss of generality) let \( C_2 \) be a co-cluster mate for \( D \) (i.e., \( \psi(D) = C_2 \)). Then the following two lemmas hold:

**Lemma 5.3.** Given \( C_1, C_2 \) and \( D \) as defined above, the community formed by “fusing” \( C_1 \) and \( C_2 \) (i.e., \( C_1 \cup C_2 \)) will also choose \( D \) as its co-cluster mate.

**Proof:** Let:

\[
\begin{align*}
\Delta C_1 &= A_{C_1} - E_{C_1,D} \\
\Delta C_2 &= A_{C_2} - E_{C_2,D} \\
\Delta D &= A_D - E_{D,C_1\cup C_2}
\end{align*}
\]

When individual communities \( C_1 \) and \( C_2 \) have both chosen \( D \) as their co-cluster mate, their contribution to the overall modularity \( (Q_B) \) is given by:

\[
Q_B' = \frac{1}{2M} \left[ E_{C_1,D} + E_{C_2,D} - \frac{1}{2M} \left( (A_{C_1} + A_{C_2}) \times A_D \right) \right]
\]

When communities \( C_1 \) and \( C_2 \) are “fused” together \( (C_1 \cup C_2) \), the contribution of the newly fused community to the overall modularity remains the same as in Eqn. 10. \( \square \)

**Lemma 5.4.** Given \( C_1, C_2 \) and \( D \) as defined above, community \( D \)’s contribution to the overall modularity could potentially increase if \( C_1 \) and \( C_2 \) are fused (i.e., \( C_1 \cup C_2 \)), provided that the condition \( \Delta C_2 \leq \Delta D \) is also met.

**Proof:** When \( C_1 \) and \( C_2 \) are individual communities, and \( D \) has chosen \( C_2 \) as its co-cluster mate, then \( D \)’s contribution \( (Q_B') \) to the overall modularity \( (Q_B) \) is given by:

\[
Q_B' = \frac{1}{2M} \left[ E_{D,C_1} + E_{D,C_2} - \frac{1}{2M} \left( A_D \times (A_{C_1} + A_{C_2}) \right) \right]
\]

However, if communities \( C_1 \) and \( C_2 \) are “fused” together, \( D \) could potentially choose \( C_1 \cup C_2 \) as its new co-cluster mate, in which case, \( D \)’s contribution to the overall modularity will become:

\[
Q_B = \frac{1}{2M} \left[ E_{D,C_1} + E_{D,C_2} - \frac{1}{2M} \left( A_D \times (A_{C_1} + A_{C_2}) \right) \right]
\]

Comparing Eqn. 11 and Eqn. 12:

\[
(A_D) \times (A_{C_1}) \leq (M) \times (2E_{D,C_1})
\]

Immediately, from the expression above we notice that:

\[
\Delta D \leq M.
\]

Therefore, community \( D \) will provide a better modularity contribution choosing \( C_1 \cup C_2 \) as its co-cluster mate only if:

\[
E_{D,C_1} + \Delta C_1 \leq 2E_{D,C_1} \\
\Delta C_1 \leq E_{D,C_1}
\]
We take advantage of the common co-cluster mate property, shown by the above two lemmas, as follows (see Algorithm 1). By fusing pairs of communities that share a common co-cluster mate, in a preprocessing step, we can possibly reduce the number of initial communities, and in turn, compact the input graph by collapsing the fused communities into vertices. This optimization is aimed at achieving run-time savings; however modularity could potentially be lost if our implementation does not explicitly perform the condition in Lemma 5.4. In fact, for the purpose of our implementation, we chose not to check for this condition, thereby trading off quality for performance. This makes our implementation of the Fuse-based algorithm a heuristic.

Also note that there are multiple minor variations possible while implementing the Fuse preprocessing heuristic itself. For instance, a conservative approach is to recompute co-cluster affiliations for communities taken from \( V_2 \), after every fuse operation that merges two communities in \( V_1 \). However, such an approach is likely to increase the runtime. In our default implementation, we choose a more aggressive approach of recomputing co-cluster affiliations for communities in \( V_2 \) only after all fuse operations are completed for communities in \( V_1 \). We observed this approach to achieve run-time savings with minimal quality reduction (as shown in Section 6).

### Run-time complexity for the Fuse heuristic:

Given that the initial number of communities in a partition is equal to the number of vertices in its corresponding vertex set, the worst-case run-time complexity for assigning co-cluster mates for all communities is \( O(n_1 \times n_2) \), assuming full connectivity as explained in Section 5.3. Then, communities belonging to the same partition are tested for common co-cluster mates for all communities is \( O(n_1 \times n_2) \) and \( O(n_2^2) \), respectively. The final complexity is then \( O(2(n_1 \times n_2) + n_1^2 + n_2^2) \) which makes our Fuse heuristic a quadratic algorithm. In practice, however, the number of communities drastically reduces (by orders of magnitude as shown in Section 6), thereby making the Fuse heuristic highly effective and fast in practice.

### 5.5 Implementation and Software Availability

We have implemented all steps of our biLouvain algorithm (including preprocessing) in C++. Scripts for data format conversions and wrangling were written in Perl. Software is available as open source at https://github.com/paolapesantez/biLouvain.

### 6 Experimental Results

#### 6.1 Experimental Setup

**Experimental Platform:** As our experimental platform, we used compute nodes of the Edison supercomputer at the National Energy Research Scientific Computing Center (NERSC). Each node has a 12-core Intel “Ivy Bridge” processor at 2.4 GHz, and 64GB RAM. Since our current implementation is serial, it used only one core of a compute node. For graph visualization, we used the Analysis and Visualization of Social Networks (Visone) software [25].

**Test inputs:** For experimentation, we used a combination of real-world and synthetic data sets (see Table 2):

- a) **Southern Women** [26]: a women vs. social events bipartite graph
- b) **Plant-Pollinator** [5]: 4 of the 23 pollinator networks, binary and weighted, where an edge represents the frequency of a pollinator’s visit to a plant
- c) **Malaria** [24]: a mapping between subsequences and genes in the malaria parasite \( P. falciparum \)
- d) **Drug-Complexes** [27]: drug-protein target interactions
- e) **Gene-Drug** [4]: gene-drug interactions
- f) **Genes-Voxels** [2]: mouse brain development during different phases, where edges represent the activation of a gene on a particular point (“voxel”) in three-dimensional space and
- g) **Host-Pathogen** [6]: species-species interactions, where an edge indicates that a pathogen was found in host.

We also used two synthetic networks (Synthetic1 and Synthetic2). More specifically, Synthetic1 corresponds to a bipartite network with a predefined, well-characterized community structure (probability of 0.9 for intra-cluster edges and probability of 0.1 for inter-cluster edges), whereas Synthetic2 represents a random bipartite network with uniform degree distribution with an unknown (possibly, weak) community structure.

**biLouvain configuration:** All modularity results presented use the Murata+ formulation defined in this paper (Section 4). Recall that biLouvain has two parameters—the iteration and phase cutoffs \( \tau_i, \tau_p \) respectively, as described in Section 5. We experimented with multiple values of \( \tau_i \) in the interval \([10^{-6}, 10^{-2}]\) on different inputs. These preliminary experiments consistently showed that: a) the final output modularities hardly changed within the interval tested; whereas b) as \( \tau_i \) is decreased, the number of iterations per phase increased, thereby increasing run-time to completion. Thus, we set the default value of \( \tau_i = 10^{-2} \) throughout our experiments. We set the phase cutoff \( \tau_p \) to 0.0 in all our experiments. Note that this represents a conservative setting

2. This is a bipartite graph with a known community structure and we use this as a benchmark for validation.
3. We experimented on all 23 networks, and select only the top 4 largest networks for presentation in this section.
where the algorithm is allowed to terminate only when two consecutive phases produce no change in the overall modularity. We also evaluated the quality-time tradeoff among different vertex ordering schemes in Section 6.3.1. Based on this evaluation, we set Random ordering as our default ordering scheme.

6.2 Qualitative Assessment

6.2.1 Validation

First, we validate our biLouvain algorithm using the Southern Women benchmark and the two synthetic networks. For the Southern Women, our algorithm was able to reproduce the expected communities [28] identically as shown in Fig. 7.

For both synthetic networks, the results were along expected lines. In Fig. 8, we show the Synthetic2 input, and the bipartite community division output from biLouvain. Recall that this is a random network with uniform degree distribution. Yet, our algorithm was able to achieve a modularity of 0.505. On the Synthetic1 input, which was configured to have a stronger community structure, the output modularity was 0.816 and the expected community structure was successfully recovered.

6.2.2 Biological Assessment of Clusters

We assessed the significance of the bipartite communities output by the biLouvain algorithm, on the Gene-Drug network, which was one of the larger real world networks tested. For assessment, we computed a Gene Ontology (GO)-based significance for each gene cluster detected by our algorithm.

The biLouvain algorithm detected 505 gene clusters from the Gene-Drug network, each consisting of two or more genes. We computed the GO significance for these 505 clusters using gProfileR [29]. The analysis resulted in 428 (84.75%) clusters with valid GO term annotations. The significance of a particular GO term, associated with a group of genes, is given by its p-value. We used a conservative approach of assigning the maximum p-value (i.e., lowest statistical significance) from within each cluster to be the cluster’s p-value. Based on this conservative scheme, we found that all of the 428 clusters have a p-value of 0.05 or less—indicating a minimum confidence level of 95%.

6.3 Performance Evaluation

In this section we evaluate the performance of the biLouvain algorithm, including an evaluation of the effectiveness of the different vertex ordering schemes (Section 6.3.1), and of the different heuristics (Section 6.3.2).

6.3.1 Performance of vertex ordering schemes

In Section 5.2 we described three vertex ordering schemes and their potential impact on biLouvain’s quality and performance. We studied this quality-time tradeoff on two significantly different real world networks: the Drug-Complex network, which represents the case of an even size distribution between the two vertex partitions (i.e., \( n_1 \approx n_2 \)); and the Gene-Drug network, which represents the case of a skewed size distribution between the partitions (here, \( n_1 \ll n_2 \)).

Fig. 9 depicts this quality-time tradeoff for these two cases. For Drug-Complex (chart (a)), we observed that all

---

TABLE 2

Bipartite network input statistics.

<table>
<thead>
<tr>
<th>Input Data Set</th>
<th>Nodes</th>
<th>Edges</th>
<th>Input Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SouthernWomen</td>
<td>18</td>
<td>14</td>
<td>89</td>
</tr>
<tr>
<td>memott1999</td>
<td>25</td>
<td>79</td>
<td>299</td>
</tr>
<tr>
<td>kevan1970</td>
<td>30</td>
<td>114</td>
<td>312</td>
</tr>
<tr>
<td>junker2013</td>
<td>56</td>
<td>257</td>
<td>572</td>
</tr>
<tr>
<td>kato1990</td>
<td>91</td>
<td>679</td>
<td>1,206</td>
</tr>
<tr>
<td>Malaria</td>
<td>297</td>
<td>806</td>
<td>2,965</td>
</tr>
<tr>
<td>Drug-Complex</td>
<td>680</td>
<td>739</td>
<td>3,690</td>
</tr>
<tr>
<td>Mouse E135L3</td>
<td>163</td>
<td>5,162</td>
<td>8,870</td>
</tr>
<tr>
<td>Mouse E151L3</td>
<td>159</td>
<td>4,341</td>
<td>7,970</td>
</tr>
<tr>
<td>Mouse E155L3</td>
<td>156</td>
<td>5,301</td>
<td>11,237</td>
</tr>
<tr>
<td>Mouse P28L3</td>
<td>89</td>
<td>5,924</td>
<td>11,253</td>
</tr>
<tr>
<td>Mouse E185L3</td>
<td>146</td>
<td>5,339</td>
<td>26,735</td>
</tr>
<tr>
<td>Mouse P41L3</td>
<td>128</td>
<td>7,796</td>
<td>33,001</td>
</tr>
<tr>
<td>Mouse P41L3</td>
<td>230</td>
<td>15,302</td>
<td>48,242</td>
</tr>
<tr>
<td>Host-Pathogen</td>
<td>8,905</td>
<td>6,314</td>
<td>22,512</td>
</tr>
<tr>
<td>Gene-Drug</td>
<td>3,092</td>
<td>14,311</td>
<td>29,389</td>
</tr>
<tr>
<td>Synthetic1</td>
<td>21</td>
<td>180</td>
<td>216</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>67</td>
<td>10</td>
<td>424</td>
</tr>
</tbody>
</table>

---

Fig. 7. (a) Southern Women Bipartite Graph. Circles - women and diamonds - events by dates. (b) Detected Communities: Red={Evelyn, Laura, Theresa, Brenda, Charlotte, Frances, Eleanor, Pearl, Ruth}, Blue={Verne, Myra, Katherine, Sylvia, Nora, Helen, Dorothy, Olivia, Fllor}, Green={6/27, 3/2, 4/12, 9/26, 2/25, 5/19, 3/15, 9/16}, and Purple={4/8, 6/10, 2/23, 4/7, 11/21, 8/3}.
three schemes behave similarly (both by quality and performance). For the Gene-Drug (chart (b)), we observed that the Random scheme demonstrated the best tradeoff, showing a reduction of 0.05 in modularity relative to Sequential, while improving the performance by a factor of 3.43. These results confirm the expected efficacies of the ordering schemes. These results provide a guide to the choice of the ordering scheme based on the input.

6.3.2 Evaluation performance of the heuristics
In this section, we provide a comparative evaluation of two versions of our biLouvain algorithm:

- **Baseline** represents the version of our biLouvain algorithm that deploys only the random vertex ordering scheme (i.e., without any preprocessing); and
- **Fuse** represents the version which also deploys the Fuse preprocessing heuristic that was described in Section 5.4.2.

Table 3 shows the results of comparing the Baseline and Fuse versions of biLouvain. Recall that the Fuse heuristic essentially is aimed at reducing the number of vertices that are input to the main biLouvain clustering algorithm. Consequently we report on both the quality and the runtime performance of both versions. The key observations

<table>
<thead>
<tr>
<th>Input Data Set</th>
<th>n₁</th>
<th>n₂</th>
<th>Murata+ Total Time</th>
<th>biLouvain Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n₁</td>
<td>n₂</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SouthernWomen</td>
<td>18</td>
<td>9</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>memmott1999</td>
<td>25</td>
<td>8</td>
<td>79</td>
<td>11</td>
</tr>
<tr>
<td>kevan1970</td>
<td>30</td>
<td>11</td>
<td>114</td>
<td>14</td>
</tr>
<tr>
<td>junker2013</td>
<td>56</td>
<td>33</td>
<td>257</td>
<td>45</td>
</tr>
<tr>
<td>kato1990</td>
<td>91</td>
<td>54</td>
<td>679</td>
<td>51</td>
</tr>
<tr>
<td>Malaria</td>
<td>297</td>
<td>144</td>
<td>806</td>
<td>221</td>
</tr>
<tr>
<td>Drug-Complex</td>
<td>680</td>
<td>190</td>
<td>739</td>
<td>189</td>
</tr>
<tr>
<td>Mouse E135L3</td>
<td>163</td>
<td>159</td>
<td>5162</td>
<td>131</td>
</tr>
<tr>
<td>Mouse E155L3</td>
<td>159</td>
<td>155</td>
<td>4,341</td>
<td>136</td>
</tr>
<tr>
<td>Mouse E155L3</td>
<td>156</td>
<td>145</td>
<td>5,301</td>
<td>124</td>
</tr>
<tr>
<td>Mouse E155L3</td>
<td>156</td>
<td>145</td>
<td>5,301</td>
<td>124</td>
</tr>
<tr>
<td>Mouse P28L3</td>
<td>89</td>
<td>8</td>
<td>592</td>
<td>74</td>
</tr>
<tr>
<td>Mouse E155L3</td>
<td>146</td>
<td>137</td>
<td>5,359</td>
<td>105</td>
</tr>
<tr>
<td>Mouse P4L3</td>
<td>128</td>
<td>123</td>
<td>7,796</td>
<td>108</td>
</tr>
<tr>
<td>Mouse P4L3</td>
<td>230</td>
<td>216</td>
<td>15,302</td>
<td>172</td>
</tr>
<tr>
<td>Host-Pathogen</td>
<td>8,905</td>
<td>3,263</td>
<td>6,314</td>
<td>3,339</td>
</tr>
<tr>
<td>Gene-Drug</td>
<td>3,090</td>
<td>2,170</td>
<td>14,311</td>
<td>2,146</td>
</tr>
<tr>
<td>Synthetic1</td>
<td>21</td>
<td>21</td>
<td>180</td>
<td>21</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>67</td>
<td>3</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4: Comparison of biLouvain quality and performance against other state-of-the-art methods.
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2017.2765319, IEEE/ACM Transactions on Computational Biology and Bioinformatics

Fig. 9. Evaluation of performance of biLouvain vertex ordering schemes. (a) Drug-Complex data set \( (n_1 \approx n_2) \). (b) Gene-Drug data set \( (n_1 \ll n_2) \).

Fig. 10. Time taken for the Fuse preprocessing step relative to the total time of our biLouvain algorithm. Note that the Fuse step's cost is shown as part of the total time, and the time axis is in log scale.

are as follows.

The Baseline version takes anywhere between seconds (for the smaller inputs) to hours (for most other inputs), while taking more than 48 hours to complete for 5 out of the 18 inputs tested.

Table 3 also shows the results of running the Fuse version. For all inputs, the Fuse version offers a reduction in the number of vertices—in some cases, up to two orders of magnitude reduction (e.g., Mouse P14L3). This drastic reduction in the graph size meant that we were able to achieve drastic reductions in run-time compared to the baseline version (four orders of magnitude or more in some inputs). This drastic reduction in run-time also resulted in a reduction in the output modularity for many inputs (compared to the Baseline version’s output). However, as Table 3 shows, the loss in modularity is generally marginal. These results show the fine balance in quality vs. time achieved by the Fuse version.

Note that all run-times reported in the Fuse column of Table 3 include the time for preprocessing. In fact, we analyzed the cost of Fuse preprocessing in Fig. 10. As shown, the cost can be as high as 23% of the total execution time for some input cases; this percentage is a result of the reductions achieved in the subsequent clustering time (compared to the Baseline), thereby demonstrating the high effectiveness of the Fuse heuristic.

6.4 Comparative Evaluation

In this section, we provide a detailed comparative evaluation of biLouvain against other existing tools (Section 6.4.1), and a comparison with a projection-based implementation (Section 6.4.2).

6.4.1 Comparison with Other Tools

We compared biLouvain against five state-of-the-art methods for bipartite community detection—the Label Propagation (DIITLPAwb+) [22], the Stochastic Block Model (biSBM) [24], the Adaptive BRIM [14], the LPBRIM [21], and the Leading Eigenvector [23]. The latter three algorithms are available in the MATLAB library BiMat [30].

We used the Fuse version of our biLouvain method in all our comparisons, as it was shown to provide a reasonable quality-time tradeoff in Section 6.3.1.

In our evaluation, we report both on the raw performance (run-time and memory usage) and the quality (based on the Murata+ modularity) of the different methods.

While running DIRTLPAwb+, AdaptiveBRIM, and LPBRIM, we observed that the outputs varied across multiple executions. However, upon a closer examination we found such variations to be minor and therefore we report on an arbitrarily selected output from each of these methods.

In the case of biSBM, the outputs not only showed minor variations like the above three methods, but more importantly, the method requires the user to input the number of communities, \( k_1 \) and \( k_2 \), for the two sides of the bipartite input. In fact, we found that the outputs varied significantly with changes to these values. Therefore, we followed an approach of running the tool over multiple configurations of \( k_1 \) and \( k_2 \), and selecting the output for which a high modularity was observed while allowing the tool to complete in a reasonable amount of time (i.e., hours). It is to be noted that the run-time of biSBM was also highly sensitive to the values of \( k_1 \) and \( k_2 \).

Table 4 shows the results of our comparative study on individual binary and weighted network inputs. As can be observed, biLouvain delivers the maximum modularity for 7 out of the 18 inputs. For the other inputs, the modularity figures delivered by biLouvain are generally close to the respective best performing method (except for Malaria). Among the other tools, Leading Eigenvector delivers comparable quality to biLouvain while consuming larger runtime.

Note that Leading Eigenvector does not use edge weights; the higher quality is a result of it using a modularity matrix of the network. This is not the case for Adaptive BRIM and LPBRIM, where the loss in quality can be attributed to their...
lack of ability to factor in the edge weights (among other reasons).

With respect to run-time, Table 4 shows that the biLouvain algorithm is the fastest for 10 out of the 18 inputs. More importantly, biLouvain consistently delivers one of the fastest run-times for most of the large inputs. Among the other tools, Adaptive BRIM delivers a comparable performance to biLouvain, although with a significantly lower quality. Furthermore, Leading Eigenvector, which delivered a high quality of results in modularity, takes substantially longer run-times on average than biLouvain.

To make it easier to see the bigger picture in relative performance (i.e., agnostic to the precise inputs), in Fig. 11 we depict the performance profiles of all the methods. These profile charts are constructed as follows: For any input $X$, let the “top” performance (as defined by the desired metric: run-time, quality or memory) was observed for some method $Y$. Then, for each method $Y'$, its relative performance on that input is expressed as a percentage of this best performance (by $Y$). For instance, if a method took twice as long time as to complete than the top performing for an input, then the contributing factor $2.0$ is added to the list of relative performance values to method $Y'$. Subsequently, the list of all performance values is sorted in non-descending order and plotted on the performance profile chart (one curve for each method).

In the performance plot, the X-axis represents the factor by which a given method fares relative to the best performing method on a particular input. The Y-axis represents the fraction of problems (i.e., inputs). In this scheme of representation, the closer a method’s curve is to the Y-axis, the more superior it’s performance is, relative to the other methods over a wider range of inputs; whereas the worst performance of a method is shown at the top-right most placement of the corresponding curve. Thus, the charts illustrate the relative performance of each method over other methods for the collection of 18 inputs tested (as opposed to the individual inputs).

The performance profile results show that biLouvain delivers the highest quality for the most fraction of the inputs (Fig. 11a). Even the worst performance observed by biLouvain is less than $1.4 \times$ away from the top modularity achieved for that output.

With respect to run-time (Fig. 11b), the performance curve biLouvain shows the best results, followed by Adaptive BRIM.

With respect to memory consumption (Fig. 11c), biLouvain is one of the best performing methods alongside biSBM and DIRTLPawb+. We note that, by default, all biLouvain runs were performed using a memory limit of 3GB. However, some methods required more memory to complete the execution. For instance, for the Gene-Drug input, Adaptive BRIM needed 7GB, LPBRIM needed 9GB, and Leading Eigenvector required 13GB.
6.4.2 A Projection-based Hybrid Approach

Projection, when applied to bipartite graphs, is known to lose information. However, we explored projection as a potential technique to initialize the set of communities at the start of the biLouvain procedure. The goal was to assess the impact of such an initialization procedure on both the quality and run-time of execution.

We implemented our projection-based approach as follows: First, we generate two unipartite graphs—one from $V_1$ and another from $V_2$—by simply performing a projection of vertices. Subsequently, for each projected graph, we run the Louvain algorithm [18], which generates a set of communities for each vertex partition. Using these two sets of communities as “seeds”, we run the biLouvain algorithm (Fuse version) on the original bipartite graph inputs. We then compared the output generated by this process against the output generated by running biLouvain directly on the input bipartite graph inputs. Table 5 shows the results obtained from using Projection in our test input data sets. As expected, we observe an improvement in run-time performance with the use of a projection for community initialization. For instance, we observed two orders of magnitude speedup in the case of the Host-Pathogen input. However, with respect to quality, directly executing biLouvain still produces better modularity values for most inputs.

Table 6 confirms the expectation that Projection all by itself, is not good enough. The table compares the modularity values achieved by: (a) if one were to simply derive the communities strictly based on projection (i.e., without running the biLouvain step) vs. (b) use the projection-based communities only for initialization and run the biLouvain algorithm to compute the final communities. As can be observed, the modularity values produced by the hybrid approach (projection followed by biLouvain) is significantly larger than the modularity values produced by projection alone.

Furthermore, the projection step in general, requires more memory to complete execution than biLouvain — e.g., Mouse P4 (5GB), and Mouse P14 (9GB). This is because the collection of edges leaving a vertex $u \in V_1$ (say) in a

<table>
<thead>
<tr>
<th>Input Data Set</th>
<th>Nodes</th>
<th>Murata+ Modularity</th>
<th>biLouvain Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n_1$</td>
<td>$n_2$</td>
<td>Q$_B$</td>
</tr>
<tr>
<td>SouthernWomen</td>
<td>9</td>
<td>2</td>
<td>0.575</td>
</tr>
<tr>
<td>memott1999</td>
<td>13</td>
<td>2</td>
<td>0.518</td>
</tr>
<tr>
<td>kevan1970</td>
<td>11</td>
<td>2</td>
<td>0.679</td>
</tr>
<tr>
<td>junker2013</td>
<td>33</td>
<td>5</td>
<td>0.670</td>
</tr>
<tr>
<td>kato1990</td>
<td>54</td>
<td>5</td>
<td>0.678</td>
</tr>
<tr>
<td>Malaria</td>
<td>144</td>
<td>11</td>
<td>0.581</td>
</tr>
<tr>
<td>Drug-Complex</td>
<td>190</td>
<td>38</td>
<td>0.833</td>
</tr>
<tr>
<td>Mouse E15L3</td>
<td>159</td>
<td>25</td>
<td>0.714</td>
</tr>
<tr>
<td>Mouse E15L5</td>
<td>145</td>
<td>13</td>
<td>0.662</td>
</tr>
<tr>
<td>Mouse E18L3</td>
<td>137</td>
<td>11</td>
<td>0.513</td>
</tr>
<tr>
<td>Mouse P4L3</td>
<td>123</td>
<td>14</td>
<td>0.513</td>
</tr>
<tr>
<td>Mouse P14L3</td>
<td>216</td>
<td>19</td>
<td>0.540</td>
</tr>
<tr>
<td>Host-Pathogen</td>
<td>3,263</td>
<td>1,636</td>
<td>0.679</td>
</tr>
<tr>
<td>Gene-Drug</td>
<td>2,170</td>
<td>397</td>
<td>0.813</td>
</tr>
<tr>
<td>Synthetic1</td>
<td>21</td>
<td>4</td>
<td>0.806</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>3</td>
<td>2</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Table 6

Comparison of Murata+ modularities achieved by using Projection-only vs. using the hybrid version that uses projection for community initialization and biLouvain for the final communities.

<table>
<thead>
<tr>
<th>Input Data Set</th>
<th>Murata+ Modularity</th>
<th>Hybrid (Projection-biLouvain)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SouthernWomen</td>
<td>0.568</td>
<td>0.568</td>
</tr>
<tr>
<td>memott1999</td>
<td>0.378</td>
<td>0.500</td>
</tr>
<tr>
<td>kevan1970</td>
<td>0.441</td>
<td>0.529</td>
</tr>
<tr>
<td>junker2013</td>
<td>0.444</td>
<td>0.526</td>
</tr>
<tr>
<td>kato1990</td>
<td>0.431</td>
<td>0.578</td>
</tr>
<tr>
<td>Malaria</td>
<td>0.565</td>
<td>0.665</td>
</tr>
<tr>
<td>Drug-Complex</td>
<td>0.676</td>
<td>0.777</td>
</tr>
<tr>
<td>Mouse E15L3</td>
<td>0.651</td>
<td>0.740</td>
</tr>
<tr>
<td>Mouse E15L5</td>
<td>0.511</td>
<td>0.586</td>
</tr>
<tr>
<td>Mouse E15L5</td>
<td>0.583</td>
<td>0.652</td>
</tr>
<tr>
<td>Mouse P28L3</td>
<td>0.368</td>
<td>0.509</td>
</tr>
<tr>
<td>Mouse E18L3</td>
<td>0.444</td>
<td>0.478</td>
</tr>
<tr>
<td>Mouse P4L3</td>
<td>0.441</td>
<td>0.524</td>
</tr>
<tr>
<td>Mouse P14L3</td>
<td>0.355</td>
<td>0.501</td>
</tr>
<tr>
<td>Host-Pathogen</td>
<td>0.425</td>
<td>0.572</td>
</tr>
<tr>
<td>Gene-Drug</td>
<td>0.688</td>
<td>0.794</td>
</tr>
<tr>
<td>Synthetic1</td>
<td>0.612</td>
<td>0.776</td>
</tr>
<tr>
<td>Synthetic2</td>
<td>0.456</td>
<td>0.456</td>
</tr>
</tbody>
</table>
bipartite graph would create a corresponding clique in the projected subgraph for those vertices in \( V_2 \) that it connects to. This may become prohibitive in memory cost for inputs with large vertex degrees.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we introduced an efficient algorithm, bilouvain, for the problem of community detection in bipartite networks. Our approach is designed to address the dual objectives of minimizing execution time, and maximizing the quality (as measured by bipartite modularity). More specifically, we make the following main contributions: i) \( \text{(metric)} \) We propose a modified variant of the Murata’s bipartite modularity; ii) \( \text{(algorithm)} \) We present a set of efficient heuristics to compute bipartite communities; and iii) \( \text{(results)} \) Our experiments demonstrate the overall runtime effectiveness and qualitative efficacy of the proposed algorithm. The experiments also showed that our algorithm substantially outperforms all the five other existing tools compared in our study, both in execution time (by orders of magnitude) and quality.

Given the paucity in tools for carrying out community detection on bipartite networks, we expect that our method and related software will be of a high utility to the research community. Thus, future extensions have been planned. These include (but are not limited to): i) Parallelization to further reduce the time to solution and enhance problem size reach; ii) Incorporation of intra-type edge information, where available, in addition to inter-type edges as part of modularity computation; and iii) Extending applications on more real world data sets.

ACKNOWLEDGMENTS

This research was supported by US Department of Energy grant DE-SC-0006516. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES


**Paola Pesáñez-Cabrera** received the BS in computational systems engineering from University of Cuenca, Cuenca, Ecuador in 2003. She is a 2011 Fulbright scholar. She received her MS degree in computer science from Washington State University, Richland, USA, in 2013. Currently, she is working toward obtaining her PhD degree in computer science at Washington State University, Pullman, USA. Her current research interests include mathematical modeling, algorithm development, computational biology, bioinformatics, graph mining, and data science.

**Ananth Kalyanaraman** received the bachelors degree from Visvesvaraya National Institute of Technology, Nagpur, India, in 1998, and the MS and PhD degrees from Iowa State University, Ames, USA in 2002 and 2006, respectively. Currently, he is an associate professor and Boeing Centennial Chair in Computer Science, at the School of Electrical Engineering and Computer Science in Washington State University, Pullman, USA. His research focuses on developing parallel algorithms and software for data-intensive problems originating in the areas of computational biology and graph-theoretic applications. He is a recipient of a DOE Early Career Award, an Early Career Impact Award and two best paper awards. He serves on editorial boards of IEEE Transactions on Parallel and Distributed Systems and Journal of Parallel and Distributed Computing. Ananth is a member of AAAS, ACM, IEEE, ISCB and SIAM.