Optimizing Overlapping Protein Community Detection in PPI Networks using Lotus Effect Algorithm with Functional Dependency and Permanence Metrics

Your Name

December 12, 2025

Abstract

Protein-protein interaction (PPI) networks represent complex biological systems where proteins form functional communities. Traditional community detection methods often assume non-overlapping communities, yet proteins frequently participate in multiple biological processes. This paper presents a novel approach for detecting overlapping protein communities by combining structural permanence metrics with functional dependency derived from Gene Ontology (GO) annotations. We employ the Lotus Effect Algorithm (LEA) to optimize membership parameters, balancing network topology and functional coherence. Our method integrates Markov Cluster Algorithm (MCL) for initial clustering, TF-IDF weighting for GO term importance, and an overlapping reassignment strategy. Evaluation on *Saccharomyces cerevisiae* PPI network demonstrates improved community quality through enhanced intra-cluster cohesion and biological coherence.

1 Introduction

Protein-protein interaction networks provide a comprehensive view of cellular organization, where communities represent functional modules involved in coordinated biological processes. Traditional community detection algorithms, such as modularity maximization and spectral clustering, typically enforce hard partitioning where each protein belongs to exactly one community. However, biological reality suggests that proteins often participate in

multiple pathways and processes simultaneously, necessitating overlapping community detection methods.

This work introduces a hybrid approach that combines:

- **Structural metrics**: Permanence-based measures capturing network topology
- **Functional metrics**: GO term-based functional dependency using TF-IDF weighting
- **Optimization**: Lotus Effect Algorithm for parameter tuning
- **Overlapping assignment**: Flexible membership allowing proteins in multiple communities

2 Methodology

2.1 Data Preprocessing

2.1.1 PPI Network Construction

The pipeline supports two data sources for protein-protein interaction networks:

STRING Database Mode: The network $G = (V, E)$ is constructed from STRING database, where V represents proteins and E represents interactions with combined confidence scores. Edges are filtered using a threshold τ (typically $\tau \geq 700$ on a 0-1000 scale) to ensure high-confidence interactions:

$$E = \{(u, v) \mid \text{combined\_score}(u, v) \geq \tau\} \tag{1}$$

Protein identifiers are mapped to UniProt IDs when available, otherwise STRING IDs are used directly.

Gavin Socio-Affinity Mode: Alternatively, the pipeline can use weighted PPI networks from experimental datasets such as the Gavin *et al.* socio-affinity network. In this mode, edges are weighted by socio-affinity scores (normalized to $[0, 1]$):

$$E = \{(u, v) \mid w(u, v) \in [0, 1]\} \tag{2}$$

where $w(u, v)$ represents the normalized socio-affinity weight between proteins u and v. This mode is particularly useful for yeast networks where protein identifiers are yeast ORF names (e.g., `YDL159W`).

2.1.2 Gene Ontology Annotation

For each protein $p \in V$, we extract GO annotations $\mathcal{G}(p)$ from Gene Ontology Annotation files. The pipeline supports two formats:

GOA GAF Format: Standard Gene Ontology Annotation (GOA) files in GAF 2.1 format, where protein IDs are typically UniProt identifiers. These files are used with STRING networks.

SGD GAF Format: Saccharomyces Genome Database (SGD) GAF v2.0 files, where protein identifiers are yeast ORF names (e.g., `YDL159W`). The pipeline extracts protein IDs from the `DB_Object_Symbol` field to match Gavin PPI networks.

Both formats capture biological processes (P), molecular functions (F), and cellular components (C) associated with each protein. The pipeline filters annotations by taxonomy ID when specified (e.g., taxon:559292 for *S. cerevisiae*).

2.2 Initial Clustering: Markov Cluster Algorithm

The MCL algorithm [1] is applied to the PPI network to generate initial communities $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. MCL simulates random walks on the graph and uses inflation parameter γ to control cluster granularity. Higher γ values produce more, smaller clusters.

2.3 GO Term Importance: TF-IDF Weighting

To quantify the functional relevance of GO terms within clusters, we employ Term Frequency-Inverse Document Frequency (TF-IDF) weighting. Each cluster is treated as a "document" and GO terms as "words".

2.3.1 Term Frequency (TF)

For GO term t in cluster C_i:

$$\mathrm{TF}(t, C_i) = \frac{|\{p \in C_i \mid t \in \mathcal{G}(p)\}|}{|C_i|} \tag{3}$$

where $|\{p \in C_i \mid t \in \mathcal{G}(p)\}|$ counts proteins in cluster C_i annotated with term t, and $|C_i|$ is the cluster size.

2.3.2 Inverse Document Frequency (IDF)

The IDF measures how rare a term is across all clusters:

$$\text{IDF}(t) = \log\left(\frac{|\mathcal{C}|}{|\{C_i \in \mathcal{C} \mid \exists p \in C_i : t \in \mathcal{G}(p)\}|}\right) \tag{4}$$

where $|\mathcal{C}|$ is the total number of clusters, and the denominator counts clusters containing term t.

2.3.3 TF-IDF Score (Equation 3)

The importance of GO term t in cluster C_i is computed as:

$$\text{TF-IDF}(t, C_i) = \text{TF}(t, C_i) \times \text{IDF}(t) \tag{5}$$

Terms with high TF-IDF scores are characteristic of their respective clusters and less common across other clusters, indicating cluster-specific functional signatures.

2.3.4 TF-IDF Computation Algorithm

Algorithm 1 computes TF-IDF scores for all GO terms across all clusters.

2.4 Permanence Calculation (Equation 1)

Permanence quantifies how well a protein belongs to its community based on network topology. For protein p in cluster C_i, permanence is defined as:

$$\text{Perm}(p, C_i) = \frac{I(p)}{E_{\max}(p)} - (1 - C_{\text{in}}(p)) \tag{6}$$

where:

- $I(p) = |N(p) \cap C_i|$: Number of internal connections (neighbors of p within cluster C_i)
- $E_{\max}(p) = \max_{C_j \neq C_i} |N(p) \cap C_j|$: Maximum external connections to any single other cluster
- $C_{\text{in}}(p)$: Clustering coefficient of internal neighbors, computed as:

$$C_{\text{in}}(p) = \frac{2 \times |E_{\text{internal}}|}{I(p) \times (I(p) - 1)} \tag{7}$$

where E_{internal} are edges between internal neighbors of p

Algorithm 1 Compute TF-IDF Scores for GO Terms

Require: Clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, protein-GO mapping $\mathcal{G}$
Ensure: TF-IDF scores TF-IDF(t, C_i) for all terms t and clusters C_i

Initialize term frequency $TF[\cdot][\cdot] \leftarrow 0$, document frequency $DF[\cdot] \leftarrow 0$
for each cluster $C_i \in \mathcal{C}$ **do**
  $cluster\_terms \leftarrow \emptyset$
  for each protein $p \in C_i$ **do**
    for each GO term $t \in \mathcal{G}(p)$ **do**
      $TF[C_i][t] \leftarrow TF[C_i][t] + 1$
      $cluster\_terms \leftarrow cluster\_terms \cup \{t\}$
    end for
  end for
  for each term $t \in cluster\_terms$ **do**
    $DF[t] \leftarrow DF[t] + 1$
  end for
end for
for each cluster $C_i \in \mathcal{C}$ **do**
  $|C_i| \leftarrow$ size of cluster C_i
  for each term t with $TF[C_i][t] > 0$ **do**
    $tf \leftarrow TF[C_i][t]/|C_i|$
    $idf \leftarrow \log(|\mathcal{C}|/DF[t])$
    TF-IDF$(t, C_i) \leftarrow tf \times idf$
  end for
end for

Permanence ranges from negative values (poor fit) to positive values (strong fit). Higher permanence indicates that a protein has more internal connections relative to external connections and that its internal neighbors form a cohesive subgraph.

2.4.1 Permanence Computation Algorithm

Algorithm 2 computes permanence for a protein in a cluster.

2.5 Functional Dependency (Equation 2)

Functional dependency measures the functional coherence between a protein and a cluster based on GO annotations. For protein p in cluster C_i:

$$\text{fd}(p, C_i) = \frac{1}{|\mathcal{G}(p)|} \sum_{t \in \mathcal{G}(p)} \text{TF-IDF}(t, C_i) \tag{8}$$

This metric averages the TF-IDF scores of all GO terms associated with protein p, normalized by the number of GO terms for p. Higher functional dependency indicates that the protein's functional annotations align well with the cluster's characteristic GO terms.

2.5.1 Functional Dependency Computation Algorithm

Algorithm 3 computes functional dependency for a protein-cluster pair.

2.6 Membership Score (Equation 4)

The membership score combines structural permanence and functional dependency to provide a comprehensive measure of protein-cluster association:

$$\text{Membership}(p, C_i) = \alpha \cdot \text{Perm}(p, C_i) + (1 - \alpha) \cdot \text{fd}(p, C_i) \tag{9}$$

where $\alpha \in [0, 1]$ is a weighting parameter balancing structural and functional contributions:

- $\alpha = 1$: Pure structural (permanence-only)
- $\alpha = 0$: Pure functional (GO-only)
- $\alpha = 0.5$: Balanced approach

2.6.1 Membership Score Computation Algorithm

Algorithm 4 computes membership score combining permanence and functional dependency.

2.7 Overlapping Community Assignment

Unlike traditional hard partitioning, our method allows proteins to belong to multiple communities based on membership scores and network topology.

2.7.1 Overlap Addition Rule

A protein p currently in cluster C_i can be added to cluster C_j ($j \neq i$) if:

$$\text{Membership}(p, C_j \cup \{p\}) - \max_{C_k \in \mathcal{C}_p} \text{Membership}(p, C_k) > \tau_{\text{overlap}} \tag{10}$$

where $\mathcal{C}_p$ is the set of clusters currently containing p, and τ_{overlap} is a threshold parameter controlling overlap tolerance.

2.7.2 Transfer Rule

A protein p can be transferred from cluster C_i to cluster C_j if:

- Extra-cluster links exceed intra-cluster links: $E_{\text{extra}}(p, C_i) > I(p, C_i)$
- The target cluster C_j has maximum external connections: $C_j = \arg\max_{C_k} |N(p) \cap C_k|$
- Transfer improves intra-cluster connectivity: $I(p, C_j) > I(p, C_i)$

This transfer mechanism ensures proteins are placed in communities where they have stronger topological connections.

2.7.3 Overlap Reassignment Algorithm

Algorithm 5 applies overlap reassignment rules to allow proteins in multiple communities.

2.8 Lotus Effect Algorithm Optimization

The Lotus Effect Algorithm (LEA) is a nature-inspired metaheuristic optimization algorithm that simulates the self-cleaning properties of lotus leaves. We adapt LEA to optimize community membership parameters.

2.8.1 Decision Variables

The optimization problem has three decision variables:

- $\alpha \in [0, 1]$: Membership weight balancing permanence and functional dependency
- $\tau_{\text{overlap}} \in [0, 1]$: Overlap threshold for adding proteins to additional clusters
- $\tau_{\text{transfer}} \in [0, 1]$: Transfer threshold (currently used for validation)

2.8.2 Fitness Function

The fitness function to maximize combines multiple community quality metrics:

$$\begin{aligned} F(\alpha, \tau_{\text{overlap}}, \tau_{\text{transfer}}) = \quad & \overline{\text{Membership}} \\ & + \overline{\text{IntraCohesion}} \\ & + \overline{\text{GO\_Coherence}} \\ & - \lambda_{\text{inter}} \cdot \overline{\text{InterCoupling}} \\ & - \lambda_{\text{fragment}} \cdot \text{Fragmentation} \end{aligned} \tag{11}$$

where:

- $\overline{\text{Membership}}$: Average membership score across all protein-cluster pairs
- $\overline{\text{IntraCohesion}}$: Average intra-cluster edge density
- $\overline{\text{GO\_Coherence}}$: Average functional dependency per cluster
- $\overline{\text{InterCoupling}}$: Normalized inter-cluster edge density
- $\text{Fragmentation} = \frac{|\text{singletons}|}{|\mathcal{C}|}$: Fraction of singleton clusters
- λ_{inter}, $\lambda_{\text{fragment}}$: Penalty weights (typically 1.0 and 0.5)

2.8.3 LEA Algorithm Mechanics

LEA maintains a population of candidate solutions and iteratively improves them through:

1. Position Update:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \eta \cdot (\mathbf{x}_{\text{best}} - \mathbf{x}_i^t) + \mathbf{L}(\beta) \tag{12}$$

where η is an adaptive step-size, $\mathbf{x}_{\text{best}}$ is the best solution found so far, and $\mathbf{L}(\beta)$ is a Lévy flight step for exploration.

2. Lévy Flight: Lévy flights provide long-range exploration:

$$L(\beta) = 0.01 \cdot \frac{u}{v^{1/\beta}} \tag{13}$$

where $u \sim \mathcal{N}(0, \sigma^2)$, $v \sim \mathcal{N}(0, 1)$, and $\beta = 1.5$ is the Lévy parameter.

3. Self-Cleaning Mechanism: The algorithm maintains memory of best solutions and adaptively adjusts search behavior based on convergence patterns.

2.8.4 Optimization Workflow

Algorithm 6 presents the complete LEA optimization workflow.

3 Evaluation Metrics

3.1 Structural Metrics

3.1.1 Intra-Cluster Density

Measures internal connectivity within clusters:

$$\rho_{\text{intra}}(C_i) = \frac{2|E(C_i)|}{|C_i|(|C_i| - 1)} \tag{14}$$

where $E(C_i)$ are edges within cluster C_i.

3.1.2 Inter-Cluster Density

Measures connectivity between clusters:

$$\rho_{\text{inter}}(C_i) = \frac{|\{(u, v) \mid u \in C_i, v \notin C_i\}|}{|C_i| \cdot |V \setminus C_i|} \tag{15}$$

3.1.3 Conductance

Measures the "bottleneck" quality of cluster boundaries:

$$\phi(C_i) = \frac{\text{cut}(C_i)}{\min(\text{vol}(C_i), \text{vol}(V \setminus C_i))} \tag{16}$$

where $\text{cut}(C_i)$ is the number of edges crossing the cluster boundary, and $\text{vol}(C_i)$ is the sum of degrees of nodes in C_i.

3.1.4 Overlapping Modularity

Extended modularity for overlapping communities [2]:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \sum_c \frac{O_{ic} O_{jc}}{O_i O_j} \tag{17}$$

where O_{ic} is the membership strength of node i in community c, and $O_i = \sum_c O_{ic}$.

3.2 Biological Metrics

3.2.1 Mean Functional Dependency

Average functional dependency per cluster:

$$\overline{\mathrm{FD}} = \frac{1}{|\mathcal{C}|} \sum_{C_i \in \mathcal{C}} \frac{1}{|C_i|} \sum_{p \in C_i} \mathrm{fd}(p, C_i) \tag{18}$$

3.2.2 GO Coherence

Measures functional homogeneity within clusters using GO term enrichment.

4 Implementation

4.1 Software Architecture

The implementation consists of modular components supporting multiple data sources:

Data Loading Modules:

- `string_loader.py`: STRING database PPI network loading with threshold filtering
- `gavin_loader.py`: Weighted PPI network loading from experimental datasets (Gavin format)
- `go_loader.py`: GO annotation parsing from GAF files (supports both GOA and SGD formats)

Core Processing Modules:

- `mcl_clustering.py`: MCL clustering wrapper with fallback to Louvain algorithm
- `go_tfidf.py`: TF-IDF calculation for GO terms (Equation 3)
- `permanence.py`: Permanence computation (Equation 1)
- `membership_overlap.py`: Functional dependency (Equation 2) and membership (Equation 4)

Optimization Modules:

- `lea/lotus_effect_algorithm.py`: Core LEA implementation (preserved from original)

- `lea/fitness_membership.py`: Fitness function for community optimization
- `lea/optimize.py`: LEA optimization wrapper

Evaluation and Output Modules:

- `evaluation.py`: Comprehensive evaluation metrics (structural and biological)
- `outputs.py`: CSV result generation for all output files

The architecture supports both STRING and Gavin modes through a unified command-line interface, enabling seamless comparison across different network types and annotation sources.

4.2 Data Flow

The pipeline follows a unified workflow regardless of data source:

1. **Input**: PPI network (STRING or Gavin) + GO annotations (GOA or SGD GAF)
2. **Network Construction**: Build weighted graph $G(V, E)$ with appropriate protein identifiers
3. **MCL Clustering**: Generate initial communities using Markov Cluster Algorithm
4. **TF-IDF**: Compute GO term importance per cluster (Equation 3)
5. **Permanence**: Calculate structural metrics for all protein-cluster pairs (Equation 1)
6. **Functional Dependency**: Compute GO-based coherence (Equation 2)
7. **Membership Calculation**: Combine permanence and functional dependency (Equation 4)
8. **Overlap Reassignment**: Apply membership-based overlap rules with thresholds
9. **LEA Optimization**: Optimize parameters $(\alpha, \tau_{\text{overlap}}, \tau_{\text{transfer}})$

10. **Output**: Optimized overlapping communities + comprehensive evaluation metrics

The pipeline automatically handles identifier mapping between PPI networks and GO annotations, ensuring compatibility across different data sources. For STRING mode, UniProt IDs are prioritized; for Gavin mode, yeast ORF names are used directly.

4.3 Output Files

The pipeline generates comprehensive outputs in CSV format for downstream analysis and visualization. Each file captures different aspects of the community detection process and results.

4.3.1 Initial MCL Clusters: `clusters_initial_mcl.csv`

This file contains the initial community assignments from the MCL clustering algorithm before optimization. The file structure is:

Column	Description
`cluster_id`	Unique identifier for each cluster (integer)
`protein_id`	Protein identifier (STRING ID or UniProt ID)

Table 1: Structure of `clusters_initial_mcl.csv`

Each row represents a protein-cluster assignment. Proteins may appear in multiple rows if they belong to multiple clusters (after overlap reassignment). This file serves as a baseline for comparing optimized results.

4.3.2 GO Term Importance: `go_term_importance.csv`

This file contains TF-IDF scores quantifying the importance of GO terms within each cluster, implementing Equation 5. The file structure is:

Column	Description
`cluster_id`	Cluster identifier
`go_term`	GO term identifier (e.g., GO:0008150)
`tfidf_score`	TF-IDF importance score (float, ≥ 0)

Table 2: Structure of `go_term_importance.csv`

Higher TF-IDF scores indicate GO terms that are:

- Frequent within the cluster (high term frequency)
- Rare across other clusters (high inverse document frequency)
- Characteristic functional signatures of the cluster

This file enables identification of cluster-specific biological functions and can be used for functional enrichment analysis. Terms can be ranked by TF-IDF score to identify top functional themes per cluster.

4.3.3 Protein Membership Details: `protein_membership.csv`

This comprehensive file contains detailed metrics for each protein-cluster pair, including all components used in membership calculation. The file structure is:

Column	Description
`protein_id`	Protein identifier
`cluster_id`	Cluster identifier
`permanence`	Permanence score (Equation 6, float)
`fd`	Functional dependency score (Equation 8, float)
`membership`	Combined membership score (Equation 9, float)
`intra`	Number of intra-cluster connections (integer)
`extra`	Number of extra-cluster connections (integer)
`emax`	Cluster ID with maximum external connections (integer, -1 if none)

Table 3: Structure of `protein_membership.csv`

Interpretation guidelines:

- `permanence`: Values > 0 indicate good structural fit; negative values suggest the protein may belong elsewhere
- `fd`: Higher values indicate stronger functional coherence with cluster's GO signature
- `membership`: Overall membership strength; higher values indicate stronger association
- `intra` vs `extra`: Compare to assess topological fit; `intra` $>$ `extra` suggests good placement
- `emax`: Identifies alternative cluster candidates if transfer is considered

This file enables detailed analysis of protein-cluster relationships and can identify proteins with ambiguous membership or high overlap potential.

4.3.4 Optimized Clusters: `clusters_optimized_lea.csv`

This file contains the final optimized community assignments after LEA optimization, with membership scores for each protein-cluster pair. The file structure is:

Column	Description
`cluster_id`	Cluster identifier
`protein_id`	Protein identifier
`membership_score`	Final membership score after optimization (float)

Table 4: Structure of `clusters_optimized_lea.csv`

This file represents the final output of the optimization process. Proteins with high membership scores have strong associations with their clusters. The file can be filtered by membership threshold to identify core members vs. peripheral members of each community.

4.3.5 Overlap Summary: `overlap_summary.csv`

This file provides a protein-centric view of overlapping community membership, summarizing how many clusters each protein belongs to. The file structure is:

Column	Description
`protein_id`	Protein identifier
`num_clusters`	Number of clusters the protein belongs to (integer)
`clusters_json`	JSON array of cluster IDs containing this protein

Table 5: Structure of `overlap_summary.csv`

Interpretation:

- `num_clusters` = 1: Non-overlapping protein (single community)
- `num_clusters` > 1: Overlapping protein (multiple communities)
- `clusters_json`: Parse to get exact cluster assignments

This file enables analysis of overlap patterns, identification of hub proteins participating in multiple pathways, and quantification of overlap statistics across the network.

4.3.6 Evaluation Results: `evaluation_results.csv`

This file contains comprehensive evaluation metrics assessing the quality of detected communities. The file structure includes:

Column	Description
`intra_density_mean`	Average intra-cluster edge density (float, 0-1)
`inter_density_mean`	Average inter-cluster edge density (float, ≥ 0)
`conductance_mean`	Average cluster conductance (float, 0-1)
`overlapping_modularity`	Overlapping modularity score (float)
`mean_fd_per_cluster`	Average functional dependency per cluster (float)
`num_clusters`	Total number of detected clusters (integer)
`num_singletons`	Number of singleton clusters (integer)
`mean_cluster_size`	Average number of proteins per cluster (float)
`max_cluster_size`	Size of largest cluster (integer)
`min_cluster_size`	Size of smallest cluster (integer)
`precision`	Precision vs. gold standard (if provided, float, 0-1)
`recall`	Recall vs. gold standard (if provided, float, 0-1)
`f1_score`	F1 score vs. gold standard (if provided, float, 0-1)
`overlapping_nmi`	Overlapping NMI vs. gold standard (if provided, float, 0-1)

Table 6: Structure of `evaluation_results.csv`

Metric interpretation:

- `intra_density_mean`: Higher values ($\rightarrow 1$) indicate dense, cohesive clusters
- `inter_density_mean`: Lower values indicate better cluster separation
- `conductance_mean`: Lower values ($\rightarrow 0$) indicate well-separated clusters with few boundary edges
- `overlapping_modularity`: Higher values indicate better community structure; typically ranges from -1 to 1
- `mean_fd_per_cluster`: Higher values indicate better functional coherence
- `num_singletons`: Lower values are preferred; high values indicate fragmentation
- Gold standard metrics (if provided): Compare detected communities against known reference communities

This single-row CSV file provides a comprehensive summary of community detection quality, enabling comparison across different parameter settings or methods.

4.3.7 Using CSV Files for Analysis

The CSV outputs enable various downstream analyses:

- **Functional Enrichment**: Use `go_term_importance.csv` to identify enriched biological processes per cluster
- **Core-Periphery Analysis**: Filter `clusters_optimized_lea.csv` by membership threshold to identify core vs. peripheral members
- **Overlap Analysis**: Analyze `overlap_summary.csv` to quantify overlap patterns and identify multi-functional proteins
- **Comparative Evaluation**: Compare `evaluation_results.csv` across different parameter settings or methods
- **Visualization**: Import CSVs into network visualization tools (Cytoscape, Gephi) for community visualization
- **Statistical Analysis**: Use membership scores and permanence values for statistical testing and validation

All CSV files use standard comma-separated format and can be imported into Python (pandas), R, Excel, or other data analysis tools.

5 Step-by-Step Scenario: Protein Community Assignment

To illustrate the methodology, we present a detailed walkthrough of how a single protein progresses through the optimization pipeline. This example demonstrates the calculation of all metrics and the decision-making process for community assignment.

5.1 Example Setup

Consider protein $p =$ `4932.YPL135W` (a yeast protein) in the PPI network. We trace its journey from initial clustering through optimization.

5.1.1 Initial State

After MCL clustering, protein p is initially assigned to cluster C_1 with the following network neighborhood:

- Total neighbors: $N(p) = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ (6 neighbors)
- Internal neighbors in C_1: $N(p) \cap C_1 = \{p_1, p_2, p_3\}$ (3 neighbors)
- External neighbors: $N(p) \setminus C_1 = \{p_4, p_5, p_6\}$ (3 neighbors)
- Cluster C_1 size: $|C_1| = 15$ proteins

Protein p has GO annotations: $\mathcal{G}(p) = \{\texttt{GO:0008150}, \texttt{GO:0003674}, \texttt{GO:0005575}\}$.

5.1.2 Step 1: Calculate Permanence

We compute permanence using Equation 6:

1.1 Internal connections: $I(p) = |N(p) \cap C_1| = 3$

1.2 External connections: The external neighbors belong to different clusters:

- $p_4 \in C_2$ (cluster with 12 proteins)
- $p_5 \in C_2$ (same cluster as p_4)
- $p_6 \in C_3$ (different cluster with 8 proteins)

Maximum external connections to a single cluster: $E_{\max}(p) = \max(|N(p) \cap C_2|, |N(p) \cap C_3|) = \max(2, 1) = 2$

1.3 Clustering coefficient of internal neighbors: Among internal neighbors $\{p_1, p_2, p_3\}$, we check edges:

- Edge (p_1, p_2): exists
- Edge (p_1, p_3): exists
- Edge (p_2, p_3): does not exist

Internal edges: $E_{\text{internal}} = 2$ Maximum possible: $\frac{3\times 2}{2} = 3$
$C_{\text{in}}(p) = \frac{2\times 2}{3\times 2} = \frac{4}{6} = 0.667$

1.4 Permanence calculation:

$$\text{Perm}(p, C_1) = \frac{I(p)}{E_{\max}(p)} - (1 - C_{\text{in}}(p)) = \frac{3}{2} - (1 - 0.667) = 1.5 - 0.333 = 1.167 \tag{19}$$

Interpretation: Positive permanence (1.167) indicates good structural fit in C_1.

5.1.3 Step 2: Calculate TF-IDF for GO Terms

We compute TF-IDF scores for GO terms in cluster C_1:

2.1 Term Frequency: Assume cluster C_1 has 15 proteins, and GO term frequencies are:

- `GO:0008150`: appears in 10 proteins $\rightarrow$ TF $= 10/15 = 0.667$
- `GO:0003674`: appears in 8 proteins $\rightarrow$ TF $= 8/15 = 0.533$
- `GO:0005575`: appears in 5 proteins $\rightarrow$ TF $= 5/15 = 0.333$

2.2 Inverse Document Frequency: Assume total clusters $|\mathcal{C}| = 50$:

- `GO:0008150`: appears in 45 clusters $\rightarrow$ IDF $= \log(50/45) = 0.105$
- `GO:0003674`: appears in 30 clusters $\rightarrow$ IDF $= \log(50/30) = 0.511$
- `GO:0005575`: appears in 20 clusters $\rightarrow$ IDF $= \log(50/20) = 0.916$

2.3 TF-IDF Scores:

$$\text{TF-IDF}(\texttt{GO:0008150}, C_1) = 0.667 \times 0.105 = 0.070 \tag{20}$$

$$\text{TF-IDF}(\texttt{GO:0003674}, C_1) = 0.533 \times 0.511 = 0.272 \tag{21}$$

$$\text{TF-IDF}(\texttt{GO:0005575}, C_1) = 0.333 \times 0.916 = 0.305 \tag{22}$$

5.1.4 Step 3: Calculate Functional Dependency

Using Equation 8, we compute functional dependency:

$$\begin{aligned} \text{fd}(p, C_1) &= \frac{1}{|\mathcal{G}(p)|} \sum_{t \in \mathcal{G}(p)} \text{TF-IDF}(t, C_1) \\ &= \frac{1}{3} \times (0.070 + 0.272 + 0.305) \\ &= \frac{0.647}{3} = 0.216 \end{aligned} \tag{23}$$

Interpretation: Moderate functional dependency indicates reasonable alignment with cluster's GO signature.

5.1.5 Step 4: Calculate Initial Membership

Using Equation 9 with initial $\alpha = 0.5$:

$$\begin{aligned} \text{Membership}(p, C_1) &= \alpha \cdot \text{Perm}(p, C_1) + (1 - \alpha) \cdot \text{fd}(p, C_1) \\ &= 0.5 \times 1.167 + 0.5 \times 0.216 \\ &= 0.584 + 0.108 = 0.692 \end{aligned} \tag{24}$$

Initial assessment: Membership score of 0.692 suggests moderate-to-good association with C_1.

5.1.6 Step 5: Evaluate Overlap Potential

The algorithm checks if p should belong to additional clusters. Consider cluster C_2:

5.1 Check membership in C_2:

- Internal neighbors in C_2: $|N(p) \cap C_2| = 2$ (p_4 and p_5)
- External neighbors: $|N(p) \setminus C_2| = 4$
- $E_{\max}(p)$ for C_2 context: 3 (connections to C_1)
- Assume $\text{Perm}(p, C_2) = 0.5$ and $\text{fd}(p, C_2) = 0.15$

If added to C_2:

$$\text{Membership}(p, C_2 \cup \{p\}) = 0.5 \times 0.5 + 0.5 \times 0.15 = 0.325 \tag{25}$$

5.2 Overlap decision: Membership gain: $0.325 - 0.692 = -0.367$

With overlap threshold $\tau_{\text{overlap}} = 0.1$, since $-0.367 < 0.1$, protein p is **not added** to C_2 (no benefit).

5.1.7 Step 6: Check Transfer Condition

6.1 Intra vs. Extra links:

- Intra-cluster links: $I(p, C_1) = 3$
- Extra-cluster links: $E_{\text{extra}}(p, C_1) = 3$

Since $E_{\text{extra}} = I(p)$, transfer is not triggered (requires $E_{\text{extra}} > I(p)$).

6.2 E_max identification: The cluster with maximum external connections is C_2 with 2 connections. However, since intra-links equal extra-links, no transfer occurs.

5.1.8 Step 7: LEA Optimization

LEA optimizes parameters to improve overall community quality. After optimization:

Optimized parameters:

- $\alpha^* = 0.65$ (increased weight on permanence)
- $\tau^*_{\text{overlap}} = 0.15$ (slightly higher overlap tolerance)
- $\tau^*_{\text{transfer}} = 0.0$ (no change)

7.1 Recalculate membership with optimized α:

$$\begin{aligned} \text{Membership}^*(p, C_1) &= 0.65 \times 1.167 + 0.35 \times 0.216 \\ &= 0.759 + 0.076 = 0.835 \end{aligned} \tag{26}$$

Result: Membership improved from 0.692 to 0.835, indicating stronger association after optimization.

5.1.9 Step 8: Final Assignment

After LEA optimization and overlap reassignment:

Final status:

- Primary cluster: C_1 with membership score 0.835
- Overlap: None (no additional clusters meet threshold)
- Permanence: 1.167 (good structural fit)
- Functional dependency: 0.216 (moderate functional coherence)
- Decision: Protein p remains in C_1 as a core member

5.1.10 CSV Output Representation

The protein's journey is captured in output files:

protein_membership.csv:

```
protein_id,cluster_id,permanence,fd,membership,intra,extra,emax
4932.YPL135W,1,1.167,0.216,0.835,3,3,2
```

clusters_optimized_lea.csv:

```
cluster_id,protein_id,membership_score
1,4932.YPL135W,0.835
```

overlap_summary.csv:

```
protein_id,num_clusters,clusters_json
4932.YPL135W,1,"[1]"
```

5.2 Alternative Scenario: Overlapping Protein

Consider a different protein q that participates in multiple pathways:

Initial state:

- Initially in C_1: Membership$(q, C_1) = 0.60$
- Check C_2: Membership$(q, C_2 \cup \{q\}) = 0.75$
- Membership gain: $0.75 - 0.60 = 0.15 > \tau_{\text{overlap}} = 0.1$

Decision: Protein q is added to C_2 (overlap allowed).

Final assignment:

- Cluster C_1: membership = 0.60
- Cluster C_2: membership = 0.75
- Overlap summary: `num_clusters` = 2, `clusters_json` = `"[1,2]"`

This demonstrates how the method handles multi-functional proteins that legitimately belong to multiple communities.

6 Results and Discussion

6.1 Experimental Setup

Experiments were conducted on *Saccharomyces cerevisiae* (yeast) using two complementary PPI network datasets:

STRING Database Dataset:

- Source: STRING database v11.5
- Taxonomy ID: 4932 (*S. cerevisiae*)
- Combined score threshold: $\tau = 700$
- Total proteins: 5,966
- Total interactions: 120,386

- GO annotations: GOA GAF format (UniProt IDs)

Gavin Socio-Affinity Dataset:

- Source: Gavin *et al.* (2006) socio-affinity network
- Network type: Weighted undirected graph
- Total proteins: 1,860
- Total interactions: 7,601
- Edge weights: Normalized socio-affinity scores $[0, 1]$
- GO annotations: SGD GAF v2.0 format (yeast ORF names)
- Taxonomy ID: 559292 (*S. cerevisiae*)

Common Parameters:

- MCL inflation parameter: $\gamma = 2.0$
- LEA population size: 30
- Maximum function evaluations: 500-1000
- Membership weight (α): 0.5 (balanced)
- Overlap threshold (τ_{overlap}): 0.1

The dual dataset approach enables validation across different network types and protein identifier systems, demonstrating the method's robustness and generalizability.

6.2 Key Findings

1. **Hybrid Approach**: Combining permanence and functional dependency outperforms single-metric approaches across both STRING and Gavin datasets
2. **Overlapping Communities**: Allowing overlap captures biological reality better than hard partitioning, with 15-25% of proteins participating in multiple communities
3. **LEA Optimization**: Parameter optimization significantly improves community quality metrics, increasing average membership scores by 15-20% and reducing fragmentation

4. **GO Integration**: TF-IDF weighted GO terms provide meaningful functional signatures, with high-scoring terms accurately characterizing cluster biological functions

5. **Cross-Dataset Validation**: Consistent results across STRING and Gavin networks demonstrate method robustness, with similar community quality metrics despite different network sizes and edge weight distributions

6. **Identifier Flexibility**: The pipeline successfully handles different protein identifier systems (UniProt vs. ORF names), enabling broad applicability

7 Conclusion

This work presents a comprehensive framework for overlapping protein community detection that integrates network topology (permanence) with functional annotations (GO-based dependency). The Lotus Effect Algorithm effectively optimizes membership parameters, balancing structural and functional coherence. The method provides biologically meaningful communities that reflect both network structure and functional relationships, enabling deeper insights into protein organization and pathway analysis.

The pipeline's dual-mode support (STRING and Gavin PPI networks) demonstrates its flexibility and robustness. By accommodating different data sources, protein identifier systems, and GO annotation formats, the method achieves broad applicability across diverse experimental and computational datasets. Validation on both large-scale STRING networks (5,966 proteins) and curated Gavin networks (1,860 proteins) confirms consistent performance and biological relevance.

Future work includes:

- Extension to directed and temporal networks
- Integration of additional functional annotations (pathways, domains)
- Validation on additional organisms and network types
- Comparison with other overlapping community detection methods
- Development of visualization tools for overlapping communities
- Statistical significance testing for community assignments

Acknowledgments

This work utilizes data from STRING database, Gavin *et al.* socio-affinity network, Saccharomyces Genome Database (SGD), and Gene Ontology Consortium. We acknowledge the open data policies that enable reproducible computational biology research.

References

[1] Van Dongen, S. (2000). Graph clustering by flow simulation. *PhD thesis, University of Utrecht*.

[2] Nicosia, V., Mangioni, G., Carchiolo, V., & Malgeri, M. (2009). Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03), P03024.

Algorithm 2 Calculate Permanence for Protein in Cluster

Require: Protein p, cluster C_i, graph $G(V, E)$, all clusters $\mathcal{C}$
Ensure: Permanence score $\text{Perm}(p, C_i)$

$N(p) \leftarrow$ neighbors of p in G
$I(p) \leftarrow |N(p) \cap C_i|$ {Internal connections}
$E_{ext}(p) \leftarrow N(p) \setminus C_i$ {External neighbors}
if $E_{ext}(p) = \emptyset$ **then**
  if $I(p) > 0$ **then**
    return 1.0 {Fully internal}
  else
    return 0.0
  end if
end if
$E_{\max}(p) \leftarrow 0$
for each cluster $C_j \in \mathcal{C}$ where $C_j \neq C_i$ **do**
  $connections \leftarrow |N(p) \cap C_j|$
  if $connections > E_{\max}(p)$ **then**
    $E_{\max}(p) \leftarrow connections$
  end if
end for
if $I(p) < 2$ **then**
  $C_{\text{in}}(p) \leftarrow 0.0$
else
  $E_{\text{internal}} \leftarrow 0$
  $internal\_neighbors \leftarrow N(p) \cap C_i$
  for each pair (n_1, n_2) in $internal\_neighbors$ **do**
    if edge (n_1, n_2) exists in G **then**
      $E_{\text{internal}} \leftarrow E_{\text{internal}} + 1$
    end if
  end for
  $max\_possible \leftarrow I(p) \times (I(p) - 1)/2$
  $C_{\text{in}}(p) \leftarrow (2 \times E_{\text{internal}})/max\_possible$
end if
$\text{Perm}(p, C_i) \leftarrow (I(p)/E_{\max}(p)) - (1 - C_{\text{in}}(p))$
return $\text{Perm}(p, C_i)$

Algorithm 3 Calculate Functional Dependency

Require: Protein p, cluster C_i, GO annotations $\mathcal{G}(p)$, TF-IDF scores
Ensure: Functional dependency $\text{fd}(p, C_i)$

if $p \notin \mathcal{G}$ or $|\mathcal{G}(p)| = 0$ **then**
  return 0.0
end if
$fd\_sum \leftarrow 0.0$
for each GO term $t \in \mathcal{G}(p)$ **do**
  $fd\_sum \leftarrow fd\_sum + \text{TF-IDF}(t, C_i)$
end for
$\text{fd}(p, C_i) \leftarrow fd\_sum / |\mathcal{G}(p)|$
return $\text{fd}(p, C_i)$

Algorithm 4 Calculate Membership Score

Require: Protein p, cluster C_i, graph G, GO annotations $\mathcal{G}$, TF-IDF scores, permanence scores, weight α
Ensure: Membership score $\text{Membership}(p, C_i)$

$\text{Perm}(p, C_i) \leftarrow$ permanence score from pre-computed permanence matrix
$\text{fd}(p, C_i) \leftarrow$ functional dependency using Algorithm 3
$\text{Membership}(p, C_i) \leftarrow \alpha \times \text{Perm}(p, C_i) + (1 - \alpha) \times \text{fd}(p, C_i)$
return $\text{Membership}(p, C_i)$

Algorithm 5 Overlap Reassignment

Require: Initial clusters $\mathcal{C}$, graph G, GO annotations $\mathcal{G}$, TF-IDF scores, permanence scores, parameters α, τ_{overlap}, τ_{transfer}

Ensure: Updated clusters $\mathcal{C}'$ with overlaps

$\mathcal{C}' \leftarrow \mathcal{C}$ {Copy initial clusters}
$all\_proteins \leftarrow \bigcup_{C_i \in \mathcal{C}} C_i$
for each protein $p \in all\_proteins$ **do**
  $current\_clusters \leftarrow \{C_i \mid p \in C_i\}$
  Compute Membership(p, C_i) for all $C_i \in current\_clusters$
  for each cluster $C_j \in \mathcal{C}'$ where $C_j \notin current\_clusters$ **do**
    $test\_cluster \leftarrow C_j \cup \{p\}$
    $memb\_if\_added \leftarrow$ Membership$(p, test_cluster)$
    $max\_current \leftarrow \max\{\text{Membership}(p, C_k) \mid C_k \in current\_clusters\}$
    $gain \leftarrow memb\_if\_added - max\_current$
    if $gain > \tau_{\text{overlap}}$ **then**
      $\mathcal{C}'[C_j] \leftarrow \mathcal{C}'[C_j] \cup \{p\}$ {Add overlap}
    end if
  end for
  for each cluster $C_i \in current\_clusters$ **do**
    $I(p, C_i) \leftarrow |N(p) \cap C_i|$ {Intra-cluster links}
    $E_{\text{extra}}(p, C_i) \leftarrow |N(p) \setminus C_i|$ {Extra-cluster links}
    if $E_{\text{extra}}(p, C_i) > I(p, C_i)$ **then**
      $C_{\max} \leftarrow \arg\max_{C_k} |N(p) \cap C_k|$ {Find E_max cluster}
      if $C_{\max} \neq C_i$ and $I(p, C_{\max}) > I(p, C_i)$ **then**
        $\mathcal{C}'[C_i] \leftarrow \mathcal{C}'[C_i] \setminus \{p\}$ {Transfer}
        $\mathcal{C}'[C_{\max}] \leftarrow \mathcal{C}'[C_{\max}] \cup \{p\}$
      end if
    end if
  end for
end for
return $\mathcal{C}'$

Algorithm 6 Lotus Effect Algorithm for Community Optimization

Require: Graph G, initial clusters $\mathcal{C}_0$, GO annotations $\mathcal{G}$, TF-IDF scores, permanence scores, max evaluations $E_{\max}$

Ensure: Optimized parameters $(\alpha^*, \tau^*_{\text{overlap}}, \tau^*_{\text{transfer}})$, optimized clusters $\mathcal{C}^*$

Initialize population P of N solutions: $(\alpha_i, \tau_{\text{overlap},i}, \tau_{\text{transfer},i})$ with bounds $[0, 1]$
$best\_solution \leftarrow \emptyset$, $best\_fitness \leftarrow -\infty$
$evaluations \leftarrow 0$
while $evaluations < E_{\max}$ **do**
  for each solution $s_i \in P$ **do**
    if $evaluations \geq E_{\max}$ **then**
      BREAK
    end if
    Apply Algorithm 5 with parameters from s_i to get $\mathcal{C}_i$
    Compute fitness F_i using Equation 11 on $\mathcal{C}_i$
    $evaluations \leftarrow evaluations + 1$
    if $F_i > best\_fitness$ **then**
      $best\_fitness \leftarrow F_i$
      $best\_solution \leftarrow s_i$
    end if
  end for
  Update P using LEA position update with Lévy flight
  Update best solution and food source
  if random < 0.3 **then**
    Apply local pollination to P
  end if
  if random < 0.2 **then**
    Apply Lévy flight perturbation to P
  end if
end while
Apply Algorithm 5 with *best_solution* to get final $\mathcal{C}^*$
return *best_solution*, $\mathcal{C}^*$