Super.Complex: A supervised machine learning pipeline for molecular complex detection in protein-interaction networks

Short title- Supervised ML pipeline for molecular complex detection in PPI networks

Meghana V. Palukuri[1*] and Edward M. Marcotte[2*]

[1]Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, Texas, USA
[2]Department of Molecular Biosciences, Center for Systems and Synthetic Biology, The University of Texas at Austin, Austin, Texas, USA

ORCIDs: M.V.P., orcid.org/0000-0002-6529-6127; E.M.M., orcid.org/0000-0001-8808-180X

*Correspondence to meghana.palukuri@utexas.edu (M.V.P.), marcotte@utexas.edu (E.M.M.)

Abstract

Protein complexes can be computationally identified from protein-interaction networks with community detection methods, suggesting new multi-protein assemblies. Most community detection algorithms tend to be un- or semi-supervised and assume that communities are dense network subgraphs, which is not always true, as protein complexes can exhibit diverse network topologies. The few existing supervised machine learning methods are serial and can potentially be improved in terms of accuracy and scalability by using better-suited machine learning models and by using parallel algorithms, respectively. Here, we present Super.Complex, a distributed supervised machine learning pipeline for community detection in networks. Super.Complex learns a community fitness function from known communities using an AutoML method and applies this fitness function to detect new communities. A heuristic local search algorithm finds maximally scoring communities with epsilon-greedy and pseudo-metropolis criteria, and an embarrassingly parallel implementation can be run on a computer cluster for scaling to large networks. In order to evaluate Super.Complex, we propose three new measures for the still outstanding issue of comparing sets of learned and known communities. On a yeast protein-interaction network, Super.Complex outperforms 6 other supervised and 4 unsupervised methods. Application of Super.Complex to a human protein-interaction network with ~8k nodes and ~60k edges yields 1,028 protein complexes, with 234 complexes linked to SARS-CoV-2, with 111 uncharacterized proteins present in 103 learned complexes. Super.Complex is generalizable and can be used in different applications of community detection, with the ability to improve results by incorporating domain-specific features. Learned community characteristics can also be transferred from existing applications to detect communities in a new application with no known communities. Code and interactive visualizations of learned human protein complexes are freely available at: https://sites.google.com/view/supercomplex/super-complex-v3-0.

Author summary

Characterization of protein complexes, *i.e.* sets of proteins assembling into a single larger physical entity, is important, as such assemblies play many essential roles in cells, ranging from gene regulation (as for the elaborate multiprotein complexes of mRNA transcription and elongation) to establishing the major structural elements of cells (as for key cytoskeletal protein complexes, such as microtubules and their trafficking proteins). Disruption of protein-protein interactions often leads to disease, therefore identifying a complete list of protein complexes allows us to better understand the association of protein and disease. From networks of protein-protein interactions, potential protein complexes can be identified computationally through the application of community detection methods, which flag groups of entities interacting with each other in certain patterns. In this work, we present Super.Complex, a generalizable and scalable supervised machine learning-based community detection algorithm that outperforms existing methods by accurately learning and using patterns from known communities. We propose 3 novel evaluation measures to compare learned and known communities, an outstanding issue. We use Super.Complex to identify 1028 human protein complexes, including 234 complexes linked to SARS-CoV-2, the virus causing COVID-19, and 103 complexes containing 111 uncharacterized proteins.

Keywords

protein complex, overlapping community detection, supervised machine learning, protein-interaction network, graph mining

Introduction

A protein complex is a group of proteins that interact with each other to perform a particular function in a cell, the basic biological unit of all living organisms. Extensive biological experiments have investigated the physical interactions between proteins, and these have been modeled via weighted protein-protein interaction (PPI) networks, where a protein-protein edge weight corresponds to the strength of evidence for the protein-protein interaction. All experimental protocols for detecting complexes (such as AP/MS, affinity purification with mass spectrometry, and CF/MS, co-fractionation with mass spectrometry) have a tendency to miss interactions (false negatives) and may also predict extra interactions (false positives). Proteins may also participate in more than one complex, potentially blurring the boundaries of otherwise unrelated protein communities. Computational analysis of protein-protein interaction networks can therefore be very useful in identifying accurate protein complexes and will help augment and direct experimental methods.

The weighted PPI network or graph G can be represented as pairs of nodes and edges *(V, E)*, where the set of nodes or vertices *V* represents the proteins, and the set of weighted edges *E* represents the strengths of evidence for interactions between proteins. Any group of nodes and edges that can be characterized as a protein complex can be referred to as a community; community detection methods can be used in turn to identify protein complexes.

A standard guideline for defining communities [1] is that a community should have more interactions or connectivity among the community than with the rest of the network. This can be modeled for example by the community fitness function in Equation 1, mapping a subgraph, *C*, *i.e.* a group of nodes and edges from the full graph, to a scalar value representing a score, where a higher score indicates more community resemblance.

$$f(C) = \delta\text{int}\ (C) - \delta\text{ext}\ (C) \qquad (1)$$

The intra-cluster density $\delta int\ (C)$ and inter-cluster density $\delta ext\ (C)$ are given by

$$\delta int\ (C) = \frac{\#\ intra-cluster\ edges}{\#\ of\ all\ possible\ edges\ in\ the\ cluster} = \frac{m_c}{n_c(n_c-1)/2} \quad (2)$$

$$\delta ext\ (C) = \frac{\#\ inter-cluster\ edges}{\#\ of\ all\ possible\ inter-cluster\ edges} = \frac{\#\ inter-cluster\ edges}{n_c(n-n_c)} \quad (3)$$

Here, n_c and m_c are the numbers of nodes and edges in subgraph *C*, respectively, and *n* is the number of nodes in graph *G*.

However, there exist many communities that do not follow this criterion but can be identified by different properties they exhibit. One such example is a star-like topology, where one central node interacts with several nodes in yeast protein-interaction networks [2], as, for example, in the case of a molecular chaperone that acts on a number of separate protein clients. In the case of human protein complexes, we also observe different topologies such as clique, linear, and hybrid between linear and clique, as shown **Fig 1**. These human protein complexes represent proteins known to belong to experimentally characterized gold-standard protein complexes from CORUM 3.0 (the comprehensive resource of mammalian protein complexes) [3] with edge weights taken from hu.MAP [4], a human protein interaction network with interactions derived from over 9,000 published mass spectrometry experiments.

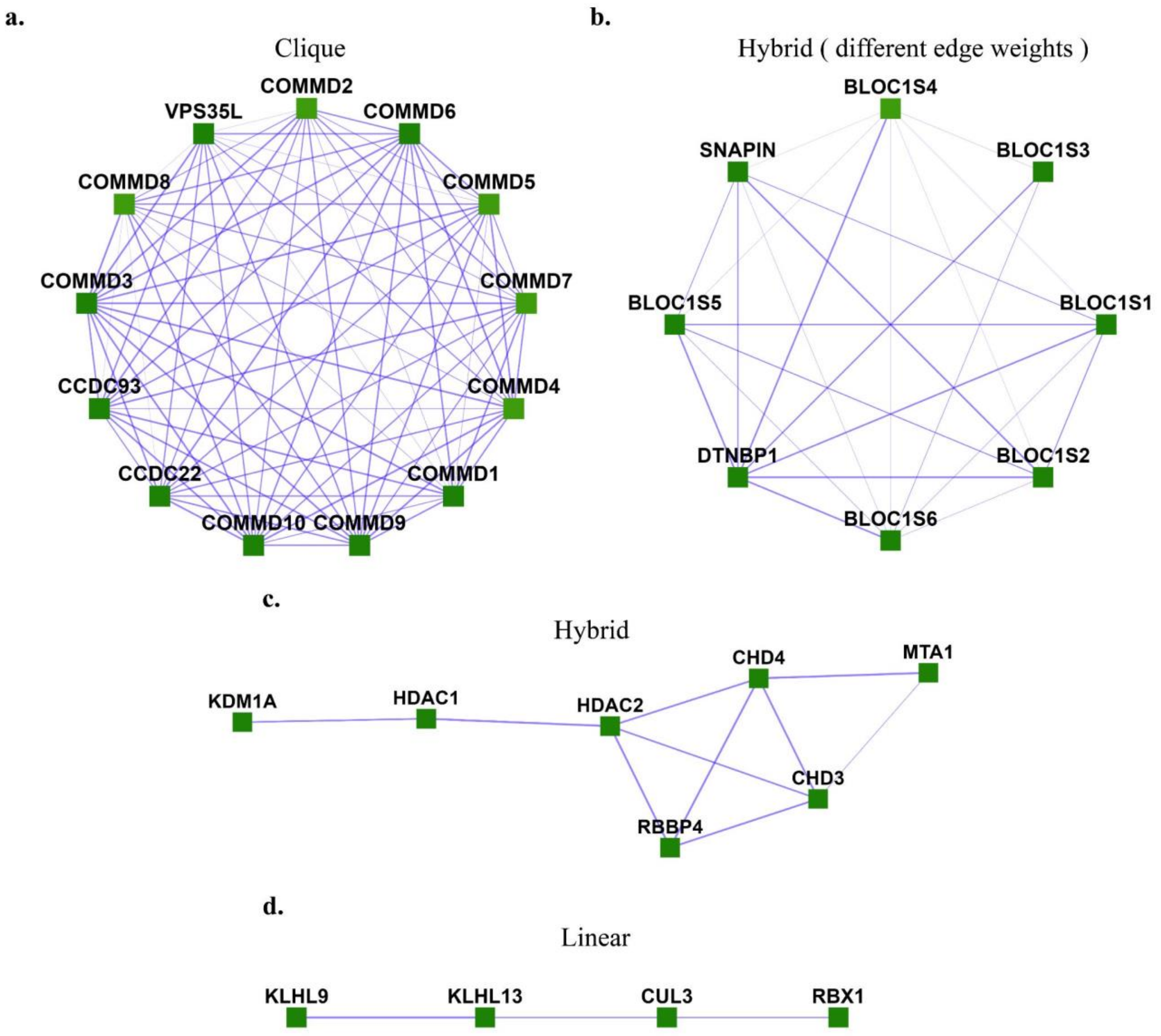


Fig 1. Different topologies are exhibited by human protein complexes. a. Clique (Commander/CCC complex), **b.** Hybrid with different edge-weights (BLOC-1 (biogenesis of lysosome-related organelles complex 1)), c. Hybrid (NRD complex (Nucleosome remodeling and deacetylation complex), **d.** Linear (Ubiquitin E3 ligase (CUL3, KLHL9, KLHL13, RBX1)). These are experimentally characterized complexes from CORUM [3] with protein interaction evidence obtained from hu.MAP [4].

Existing community detection methods have primarily tried to optimize for high scores of community fitness functions, such as that of equation 1 [5]. These include unsupervised methods, such as implemented by MCL- Markov Clustering [6], MCODE - Molecular COmplex DEtection [7], CFinder [8], SCAN- Structural Clustering Algorithm for Networks [9], CMC - Clustering based on Maximal Cliques [10], COACH - COre-AttaCHment based method [11], GCE - Greedy Clique Expansion [5], and ClusterONE - clustering with overlapping neighborhood expansion [12], as well as semi-supervised machine learning algorithms such as COCDM - Constrained Overlapping Complex Detection Model [13].

When there are sufficient data available on known communities, rather than applying a generic community fitness function to the problem, it can be more accurate to learn a community fitness function directly from known communities. Then, new communities detected with the learned community fitness function can be expected to better resemble known communities in the field. Supervised machine learning methods are well suited for this purpose, and a few methods have been used to learn a community fitness function from constructed community embeddings, *i.e.*, community representations in vector space, obtained by extracting topological and domain-specific features from communities. The community fitness function learned can then be used to select candidate communities from the network and evaluate them. Since finding maximally scoring communities in a network is an NP-hard (non-deterministic polynomial-time hard) problem [2], heuristic algorithms have been used to find candidate communities. A common strategy is to select a seed (such as a node or a clique) and grow it into a candidate community by iteratively selecting neighbors to add to the current subgraph using heuristics such as iterative simulated annealing until a defined stopping criterion is met for the growth process. This process is repeated with different seeds to generate a set of candidate communities.

Existing supervised methods use different machine learning methods to learn the community fitness function after extracting different features and use different heuristic algorithms to select candidate communities. The first supervised method [2] used a support vector machine (SCI-SVM) and a Bayesian network (SCI-BN) with 33 features with a greedy heuristic, followed by iterative simulated annealing. Stopping criteria for the growth of a seed include limiting the rounds of growth, checking for score improvement over multiple iterations, and checking for overlap with learned candidate communities so far. A second approach [14] recursively trained a two-layer feed-forward neural network model, NN for the classifier using 43 features. This greedy heuristic sequentially grows seeds of the highest degree with similar stopping criteria as [2]. Supervised learning protein complex detection SLPC [15] uses a regression model (RM) with 10 topological features, solved by gradient descent. A modified cliques algorithm finds and grows maximal cliques using a random but exhaustive neighbor selection followed by a greedy growth heuristic. The algorithm stops when no node addition can yield a higher score, after which they merge some pairs of overlapping complexes with an overlap greater than a threshold. ClusterEPs, short for cluster emerging patterns [10] uses a score function based on noise-tolerant emerging patterns (NEPs) which are minimal discriminatory feature sets using 22 features, along with an average node degree term. Like [14], the heuristic for this method also grows the highest degree seed nodes sequentially. The neighboring node that shares the maximum number of edges with the current subgraph is selected as a candidate for growth in each iteration and a greedy growth heuristic is used, stopping when the score is greater than 0.5. ClusterSS, short for clustering with supervised and structural information [16] uses a neural network with one hidden layer and 17 features, along with a traditional structural score function from [12]. A greedy heuristic grows seed nodes, also considering deletion of any existing subgraph nodes, with an optimization step of considering only the top k nodes by degree. The stopping criterion is when the new score is less than a factor times the old score. Both ClusterEPs and ClusterSS merge pairs of communities with overlap greater than a threshold at the end.

Regarding scalability, the above methods have generally only been implemented on small yeast protein complex datasets, except for ClusterEPs, which trains on yeast data and tests on human PPIs. [17] implement the regression model of [15] on a human PPI network re-weighted by breast-cancer specific PPIs extracted from biomedical literature to detect disease-specific complexes. However, these methods employ serial candidate community sampling, negatively impacting their scalability to large networks such as hu.MAP [4], a human protein-interaction network with ~8k nodes and ~60k edges.

In this work, we present Super.Complex (short for Supervised Complex detection algorithm), an end-to-end highly scalable, distributed, and efficient community detection pipeline that explores multiple supervised learning methods with AutoML (Automated Machine Learning) to learn the most accurate community fitness function from known communities. Super.Complex then samples candidate subgraphs in parallel by seeding nodes or starting with maximal cliques and growing them with an epsilon-greedy heuristic, followed by an additional heuristic such as iterative simulated annealing or pseudo-metropolis using the learned community fitness function. On a yeast PPI network, Super.Complex outperforms all 6 existing supervised methods, as well as 4 unsupervised methods. Three novel evaluation measures are proposed to overcome certain shortcomings of existing metrics. We apply Super.Complex to hu.MAP to yield 1028 protein complexes, including high-scoring previously unknown protein complexes, potentially contributing to new biology, and make all data, code, and visualizations openly available.

Results and Discussion

Super.Complex - a scalable, distributed supervised ML-based community detection method

The pipeline Super.Complex comprises two main tasks, first, learning a community fitness function with AutoML methods, and second, using the community fitness function to intelligently sample overlapping communities from a network in parallel, as shown in **Fig 2**. For the first task, we first perform (i) Data Preparation, where known communities are cleaned and split into non-overlapping training and testing sets, followed by construction of training and testing negative community data. Next, in (ii) Feature Extraction, topological characteristics for all communities are computed to construct training and testing feature matrices. AutoML (iii) then compares different ML (Machine Learning) pipelines to select the best one, followed by training and testing the best ML pipeline, thus learning the community fitness function as the binary classifier distinguishing positive communities from negatives. Having learned the community fitness function, Super.Complex then uses it in its heuristic algorithm for the second task of (iv) Candidate Community Search. The algorithm can start with either single nodes or maximal cliques as seeds. These seeds are grown, *i.e.* neighbors are added iteratively to form the final set of candidate communities. Four options for heuristics are provided to decide which neighboring node to add in each iteration of growth, including greedy edge weight, ϵ-greedy, ϵ-greedy + pseudo metropolis, and ϵ-greedy + iterative simulated annealing. Finally, in the last step (v) evaluation, the learned communities are compared with known communities. The steps of the pipeline are fairly independent and can be improved on their own with methods to test the accuracy/performance of each of the steps.

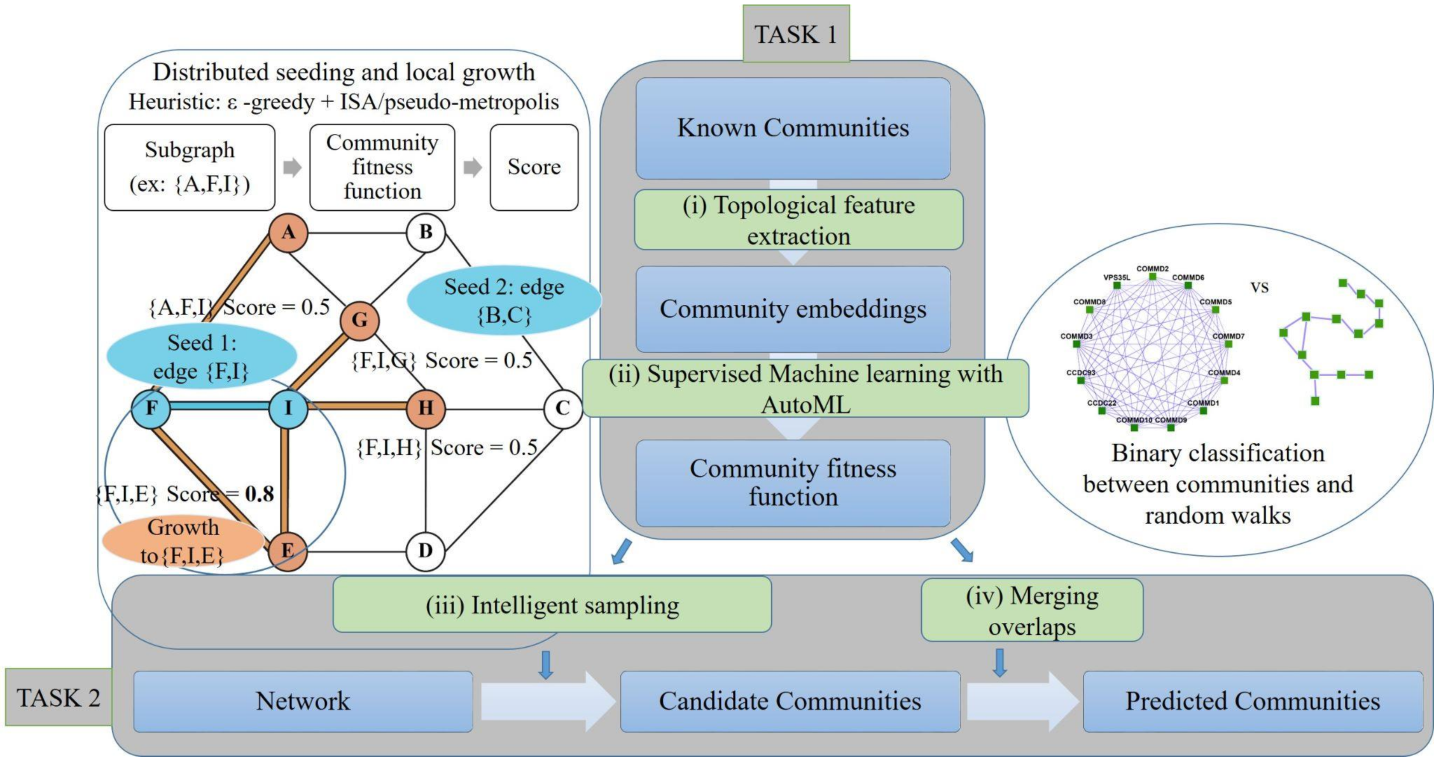

TASK 1
Distributed seeding and local growth
Heuristic: ε -greedy + ISA/pseudo-metropolis
Subgraph (ex: {A,F,I})
Community fitness function
Score
A
B
G
C
F
I
H
E
D
{A,F,I} Score = 0.5
Seed 2: edge {B,C}
Seed 1: edge {F,I}
{F,I,G} Score = 0.5
{F,I,H} Score = 0.5
{F,I,E} Score = 0.8
Growth to {F,I,E}
Known Communities
(i) Topological feature extraction
Community embeddings
(ii) Supervised Machine learning with AutoML
Community fitness function
VS
Binary classification between communities and random walks
(iii) Intelligent sampling
(iv) Merging overlaps
TASK 2
Network
Candidate Communities
Predicted Communities

Fig 2. Super.Complex identifies likely protein complexes within a PPI network using a distributed supervised AutoML method.
Task 1: Learning a community fitness function:
(i) Topological features are extracted from known communities to build community embeddings (feature vectors, which are representations of communities in vector space)

(ii) A score function for communities, the community fitness function, is learned from the community embeddings as the decision function to classify a network subgraph as a community or a random walk. The best score function is selected after training multiple machine learning models with tpot [18], an AutoML pipeline.
Task 2: Selecting candidate communities from the network:
(iii) Multiple communities are sampled in parallel from the network. To build each candidate community, a seed edge is selected and grown using a 2-stage heuristic. First, we use an epsilon-greedy heuristic to select a candidate neighbor, and then we use a pseudo-metropolis (constant probability) or iterative simulated annealing heuristic to accept or reject the candidate neighbor for growing the current community. The figure shows an iteration of neighbor selection using a greedy heuristic, starting from a seed edge.
(iv) The candidate communities are merged such that the maximum overlap between any 2 communities is not greater than a specified threshold.

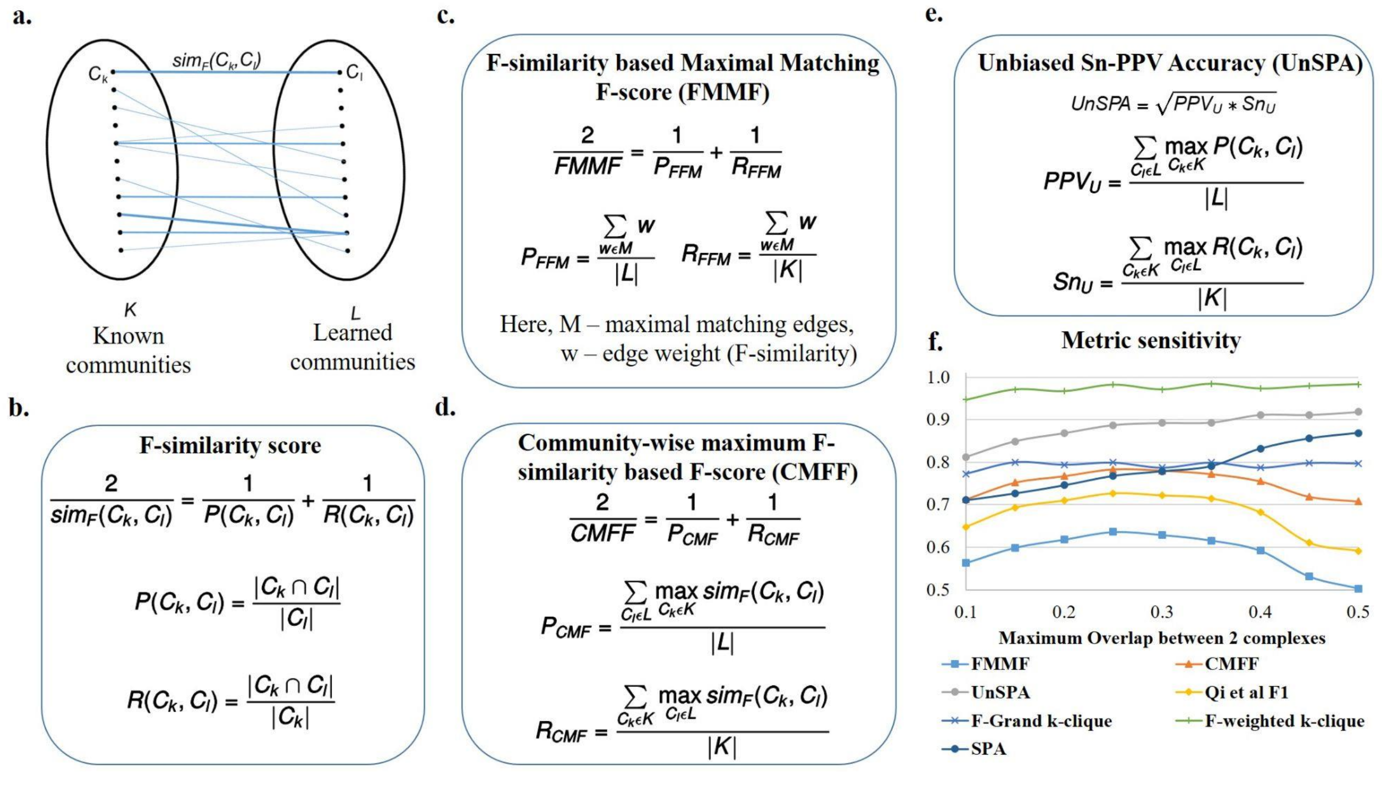
Although the pipeline broadly follows the same major steps as other supervised techniques, each step has been further optimized as follows: Unlike some existing methods that remove nodes in the process of growth (e.g. such as Louvain [19] and ClusterSS), we show that our method guarantees properties such as internal connectivity of communities. Further, the merging algorithm we employ guarantees that no two communities overlap more than a specified threshold, and provides an additional guarantee that no two communities can be merged to yield a higher scoring community when the specified threshold is 0 overlap, *i.e.* in the case of obtaining non-overlapping communities. Super.Complex implements an original distributed architecture and software design, scaling to large networks such as hu.MAP with ~8k nodes and ~60k edges. To our knowledge, epsilon-greedy heuristics in conjunction with other heuristics such as iterative simulated annealing have not been applied in the past for community detection. Super.Complex has a cross-validation pipeline to select the heuristic and the parameters that work best for the application at hand. Minimal hyper-parameter selection is required in our algorithm with default parameters provided when smart hyperparameters cannot be inferred.

Since the number of known communities can be limited, we emphasize the preservation of known communities when splitting them into train-test sets while ensuring independence - *i.e.*, no edge overlap between a train and test community on the network, similar size distributions for both sets, and 70-30 ratios in train-test sets. Similarly, a minimal number of merges is attempted in the merging algorithms devised to maintain a high number of learned protein complexes. We perform manual feature selection to demonstrate the merit of this step and include automated feature selection in the pipeline. Further, unlike existing supervised methods, which evaluate the performance of their algorithms on a reduced network with only nodes present in known communities, we evaluate our algorithm on the full network for more accurate evaluation. Finally, we note that our methods are also applicable in domains with limited or no knowledge by transferring community fitness functions from other domains, such as the defaults we provide for human protein complex detection.

Three novel evaluation measures to compare learned communities with known communities

Comparing sets of learned and known communities accurately is an outstanding issue. Poor evaluation measures do not satisfactorily identify the quality of learned communities and make it difficult to evaluate a community detection algorithm. Sets of learned communities achieving high scores with existing evaluation measures have been observed to have a lot of redundancies, e.g. multiple learned communities are very similar with high overlaps [20]. Known big communities were also observed to be split into several learned communities while still achieving good scores on evaluation measures. While it is undesirable to have many false negatives, having many false positives is more hurtful, as wet-lab experiments for biological validation tend to be quite expensive and time-consuming to perform. Therefore, we concentrate on including precision-like measures that compute false positives. Further, evaluation measures that are not sensitive to changes in the sets of learned communities limit our abilities to iterate successfully over algorithm modifications to improve algorithms. We examine the specific shortcomings of different evaluation measures and propose new measures to help overcome the issues discussed and construct robust yet sensitive measures.

F-similarity-based Maximal Matching F-score (FMMF). An issue with many measures such as Qi et al F1 score (equation 13) and SPA (equation 14) is that they don't penalize redundancy, *i.e.* if we learn multiple same or very similar communities which are each individually high scoring, we will get a high value of precision-like measures. This is because in many cases, many to one matches are being made between learned communities and known communities. To deal with such issues, it is best to make one-to-one matches. The MMR (Maximal Matching Ratio) is one such good measure, however, it only calculates a recall-like measure by dividing the sum of the weights of edges (in a maximal sum of one-to-one edge weights) by the total number of known communities. Taken alone this cannot account for precision, for instance, if we learn a series of random subgraphs, these have low weights and will be ignored, while a high MMR score can be obtained from only a small number of high quality learned communities. Therefore, we define the precision equivalent for MMR, P_{FFM} in **Fig 3c**.


a.
$sim_F(C_k, C_l)$
C_k
C_l
K
Known communities
L
Learned communities
b.
F-similarity score
c.
F-similarity based Maximal Matching F-score (FMMF)
Here, M – maximal matching edges, w – edge weight (F-similarity)
d.
Community-wise maximum F-similarity based F-score (CMFF)
e.
Unbiased Sn-PPV Accuracy (UnSPA)
f.
Metric sensitivity
1.0
0.9
0.8
0.7
0.6
0.5
0.1
0.2
0.3
0.4
0.5
Maximum Overlap between 2 complexes
FMMF
CMFF
UnSPA
Qi et al F1
F-Grand k-clique
F-weighted k-clique
SPA

$$\frac{2}{sim_F(C_k, C_l)} = \frac{1}{P(C_k, C_l)} + \frac{1}{R(C_k, C_l)}$$

$$P(C_k, C_l) = \frac{|C_k \cap C_l|}{|C_l|}$$

$$R(C_k, C_l) = \frac{|C_k \cap C_l|}{|C_k|}$$

$$\frac{2}{FMMF} = \frac{1}{P_{FFM}} + \frac{1}{R_{FFM}}$$

$$P_{FFM} = \frac{\sum_{w \epsilon M} w}{|L|} \quad R_{FFM} = \frac{\sum_{w \epsilon M} w}{|K|}$$

$$\frac{2}{CMFF} = \frac{1}{P_{CMF}} + \frac{1}{R_{CMF}}$$

$$P_{CMF} = \frac{\sum_{C_l \epsilon L} \max_{C_k \epsilon K} sim_F(C_k, C_l)}{|L|}$$

$$R_{CMF} = \frac{\sum_{C_k \epsilon K} \max_{C_l \epsilon L} sim_F(C_k, C_l)}{|K|}$$

$$UnSPA = \sqrt{PPV_U * Sn_U}$$

$$PPV_U = \frac{\sum_{C_l \epsilon L} \max_{C_k \epsilon K} P(C_k, C_l)}{|L|}$$

$$Sn_U = \frac{\sum_{C_k \epsilon K} \max_{C_l \epsilon L} R(C_k, C_l)}{|K|}$$

Fig 3. Proposed evaluation measures - FMMF, CMFF, and UnSPA are sensitive metrics. a. Bipartite graph, where each edge weight corresponds to the F-similarity ($sim_F(C_k, C_l)$) between C_k, a known community from K , the set of known communities and C_l, a learned community from L , the set of learned communities. **b**. The F-similarity score combines precision ($P(C_k, C_l)$) and recall ($R(C_k, C_l)$) measures, computed as fractions of the number of common nodes w.r.t the number of nodes in a community. $|C|$is the number of nodes in community C and $|C_1 \cap C_2|$ is the number of nodes common to both communities. **c.** F-similarity-based Maximal Matching F-score (FMMF) combines precision (P_{FFM}) and recall (R_{FFM}) measures computed for a maximal matching, M of the bipartite graph in **Fig 3a d.** Community-wise Maximum F-similarity based F-score (CMFF) combines precision (P_{CMF}) and recall (R_{CMF}) measures, averaging over the maximum F-similarity score for a community in a particular set (e.g. known communities) w.r.t to a community of the other set (e.g. learned communities) **e.** UnSPA is an unbiased version of Sn-PPV accuracy (SPA), computed as the geometric

mean of unbiased PPV (PPV_u) and unbiased Sensitivity (Sn_u), computed similar to precision and recall measures in CMFF, only, instead of the F-similarity score, precision and recall similarity scores are used respectively **f.** Sensitivity of different evaluation measures w.r.t. (maximum pairwise Jaccard coefficient) overlap between communities shows that FMMF, CMFF, UnSPA, and existing measures Qi et al F1 score (equation 13), and SPA (equation 14) are sensitive metrics, with FMMF, CMFF, and Qi et al F1 score following the desired trend. Here, each data point on the plot corresponds to a measure evaluating an individual run of Super.Complex's merging algorithm with a maximum Jaccard overlap threshold set to the x-axis value.

In **Fig 3c**, M is a set of weights of a set of maximal one-to-one matches, found using Karp's algorithm [21]. The weight w that we use is the F-similarity score (**Fig 3b**), also described in the next section, Community-wise Maximum F-similarity based F-score (CMFF), unlike the neighborhood affinity used in the original MMR. Correspondingly we can define an F-score, FMMF, as the harmonic mean of the precision P_{FFM} and recall R_{FFM}, also shown in **Fig 3c**.

By doing a one-to-one match, we are also indirectly penalizing cases where the benchmark community is split into multiple smaller communities in the learned set of communities, since the measure considers the weight of only one of the smaller learned communities that comprise the known community, ignoring the rest. Thus only the small weight of the matched community is considered, penalizing this case, unlike one-to-many measures that aggregate the contributions from each of the smaller communities to finally achieve a high score.

Community-wise Maximum F-similarity-based F-score (CMFF). The authors of [5] compute F1 scores at the individual known community-learned community match level and look at the histograms of these scores for all known communities. While their work does not state the exact formulation of their F1 score, we are inspired by them to define an F1 score at the match level, *i.e.* an F-similarity score, by comparing the nodes of a learned and a known community. Our F-similarity score is a combination of the recall (of the nodes of the known community) and the precision (of the nodes of the learned community), as shown in **Fig 3b**.

Our F-similarity score can be used in a thresholding scenario to determine a match, and then the overall precision, recall, and F1 scores for the set of predictions can be computed as in equation 12. Alternatively, our F-similarity score can be used to determine the best matches for communities and overall measures can be defined that can be investigated to reveal the contributions at the individual match level as well. For interpretability at the match level, similar to the unbiased sensitivity and PPV metrics (as discussed in the next section, Unbiased Sn-PPV Accuracy (UnSPA)), we can define precision and recall measures that evaluate, for each community, the closest matching community in the other set using a similarity metric. Using the F-similarity score as the similarity metric here, we define precision and recall-like measures, and combine them into the F1-like measure, CMFF - Community-wise Maximum F-similarity based F-score, as shown in **Fig 3d**. We detail a general framework to construct similar measures in the next paragraph, drawing inspiration from modifications to the Qi et al F1 score. This framework also gives another method of constructing the CMFF.

In the Qi et al measures from [2], (equations 11, 12 and 13), a binary indication of a possible match is used, *i.e.* as long as there exists a possible match, it is used as a 1 or 0 count towards the aggregate precision or recall measures. Having a measure that provides matches between learned and known communities allows easy identification of previously unknown communities. One to many matches such as Qi et al precision-recall (PR) measures that do not use an explicit matching between learned and known communities can be modified to obtain a matching. In the modified measure, for each community, we choose the most similar community in the other set in order to give the matching. While thresholding measures such as Qi et al F1 score (equation 13) have the advantage of being robust measures, until a match crosses a threshold, the measure will not change, making it unchanging to small variations in predictions. Measures with low sensitivity make it difficult to compare algorithms and select parameters. Weighted measures are more sensitive, giving different values based on the quality of matches, and are more precise when compared to summing binary values of match existence. Accordingly, a more sensitive and precise version of the Qi et al F1 score can be obtained by summing up weights indicating the similarity scores. For instance, instead of the thresholding measure in equation 10, the neighborhood affinity similarity measure (also obtained by multiplying the L.H.S of equation 10) can be used to construct a more precise and sensitive measure, i.e,

$$sim(C_1, C_2) = Neighborhood\ affinity = \frac{|C_1 \cap C_2|^2}{|C_1| * |C_2|} \quad (4)$$

$$Recall = \frac{\sum_{Ck \in K} \max_{C_l \in L} sim(C_l, C_k)}{|K|}, \ Precision = \frac{\sum_{Cl \in L} \max_{C_k \in K} sim(C_l, C_k)}{|L|} \text{ and } F1 = \frac{2^* Recall * Precision}{Recall + Precision} \quad (5)$$

Here, $sim(C_1, C_2)$is a similarity measure between communities $C_1$and C_2, with $|C_1|$is the number of nodes in $C_1$and $|C_1 \cap C_2|$ is the number of nodes common to both communities $C_1$and C_2. C_k is a known community from K, the set of known communities and $C_l$is a learned community from L, the set of learned communities.

Different similarity measures, such as the Jaccard coefficient from equation 6 can be used to construct different F1 measures. We recommend the F-similarity measure in **Fig 3b**, as it can be broken down into a precision-based and recall-based measure at the level of comparing a known and learned community, and use it to construct the CMFF score.

Unbiased Sn-PPV Accuracy (UnSPA). Consider the precision-like positive-predictive value (PPV), recall-like Sensitivity (Sn), and their combined Sn-PPV accuracy (which we term as SPA) [22], also given in equation 14. In Sensitivity, the numerator is a sum of the maximal number of recalled nodes for each community and the denominator is a sum of the number of nodes in each community. Measures like these do not give equal importance to each of the known communities and assign higher values for recalling larger communities when compared to recalling smaller communities. For instance, an algorithm that perfectly recalls numerous smaller communities and does not recall much of a few bigger communities can get a worse sensitivity score when compared to an algorithm that does the opposite, *i.e.* recalls most of the big community and does not recall much of any of the smaller communities. Rather than inducing bias into a measure that decides which communities should be weighted higher, it may be a better idea to have a measure that gives equal weights to all communities. We define an unbiased sensitivity Sn_u in **Fig 3e**, by dividing by the total number of known communities.

In PPV, the denominator sums, for each learned community, the sum of the subset of nodes in the learned community shared by all known communities. This does not contribute accurately to a precision-like measure, as nodes that are absent in known communities are ignored. For instance, a learned community that has all the nodes in a known community, but also includes a lot of possibly spurious nodes will be scored in the same way as a learned community which is an exact match to the known community. Further, in PPV, nodes in a learned community shared by multiple known communities get counted an extra number of times in the denominator. So if we share a set of nodes with multiple known communities we get penalized more than if we share the set with only a few known communities, or if nodes of our community are shared disjointly with different known communities. The reasoning for allowing such behavior is again biased and does not support the detection of overlapping known communities. For example, a learned community that has a high overlap with 2 known communities (ex: a learned community with 10 nodes that shares all of its nodes with each of the known communities) will contribute lesser (0.5) to a PPV score than a learned community which overlaps lesser with one known community (ex: 6 nodes in a learned community with 10 nodes overlapping with only one known community, giving a 0.6 contribution to the PPV). To overcome these issues, we propose an unbiased PPV, PPV_u in **Fig 3e**, where we divide by the total number of learned communities. The corresponding unbiased accuracy is obtained by taking the geometric mean of the PPV_u and Sn_u as shown in **Fig 3e**.

From the sensitivity of measures plot in **Fig 3f**, we find that the FMMF score, the Qi et al F1 score (equation 13), and CMFF score are most sensitive to the pairwise overlap between communities, giving high values at the overlap coefficient yielding the best results, determined via visual inspection of the learned results, as follows. We observed highly overlapping, repetitive, and large numbers of similar learned protein complexes in our experiment on hu.MAP, such as several resembling the ribosome complexes at the high overlap threshold of 0.5 Jaccard coefficient, whereas, at low overlaps, we obtain a total small number of learned complexes, 84 learned complexes after removing proteins absent from known complexes. As we would like a high number of good quality complexes, we find that intermediate values of overlap Jaccard coefficient yield satisfactory results, for instance, at 0.25 Jaccard coefficient, we obtain 121 complexes after removing proteins absent from known complexes, with a high recall of known complexes and good observed quality, *i.e.* low numbers of very similar overlapping learned complexes. The clique-based measures from [4] - F-grand K-clique and F-weighted K-clique do not vary much with overlap, and the UnSPA, like the SPA, increases with increasing overlap threshold. However, the rate of increase of SPA w.r.t increasing overlap values is greater than UnSPA, yielding comparatively higher scores at undesirable high overlaps. In other words, instead of the desired decreasing trend from 0.25 to 0.5 Jaccard coefficient overlap, we have a highly increasing trend for SPA, compared to the almost constant trend for UnSPA - an improvement over SPA that can possibly be attributed to the unbiasing modification we have introduced. Therefore, for accurate evaluation in which redundancy (high overlap) is penalized, we recommend unSPA over SPA, and primarily recommend the FMMF score, CMFF score, and the existing Qi et al F1 score (equation 13).

Super.Complex applied to a human protein interaction network to detect protein complexes

Experiment details. We first test and ensure that the pipeline achieves perfect results on a toy dataset we construct comprising disconnected cliques of varying sizes, each corresponding to a known community, where we use all nodes as seeds for growth during the prediction step.

To learn potentially new human protein complexes, we apply Super.Complex on the human PPI network hu.MAP [4] using a community fitness function that is learned from known complexes in CORUM [3]. The network available on the website (http://hu1.proteincomplexes.org/static/downloads/pairsWprob.txt) has 7778 nodes and 56,712 edges, after an edge weight cutoff of 0.0025 was applied to the original 64,048 edges. There are 188 complexes after data cleaning, a set we term as 'refined CORUM', out of the original 2916 human CORUM complexes, which underscores the importance of minimizing any losses in the merging and splitting steps of the pipeline. Note that of the complexes from CORUM that were removed, there were over 1000 complexes that had fewer than 3 members, and the remaining removed complexes consisted of duplicates and disconnected complexes with edges from hu.MAP. Note, however, that hu.MAP was the highest confidence human protein interaction network integrating 3 large previous human protein-interaction networks, all built using high confidence data from large-scale (~9000) laboratory experiments. The edge weights of hu.MAP were trained using an SVM based on features obtained from experiments.

Experiment results. The best results, following different parameter sweeps from the experiment on hu.MAP are given in **Fig 4**. The best-case corresponds to using 10 times the number of positives while sampling negatives with a uniform size (number of nodes) distribution, ϵ as 0.01 in the ϵ- greedy + iterative simulated annealing with T_0 as 1.75 and αas 0.005, and using the merging algorithm with a Qi overlap measure (equation 10) of 0.375. From **Fig 4e**, we verify that the size distributions of the train and test sets are similar. In **Fig 4a**, we can see that we get a good precision-recall curve on the test set for the subgraph classification task as a positive or negative community, achieving an average precision score of 0.88 with a logistic regression model (which is the final ML model stacked on a set of other ML models and processors, output as the best model trained on the training set with 5-fold cross-validation and achieving a cross-validation score of 0.978).

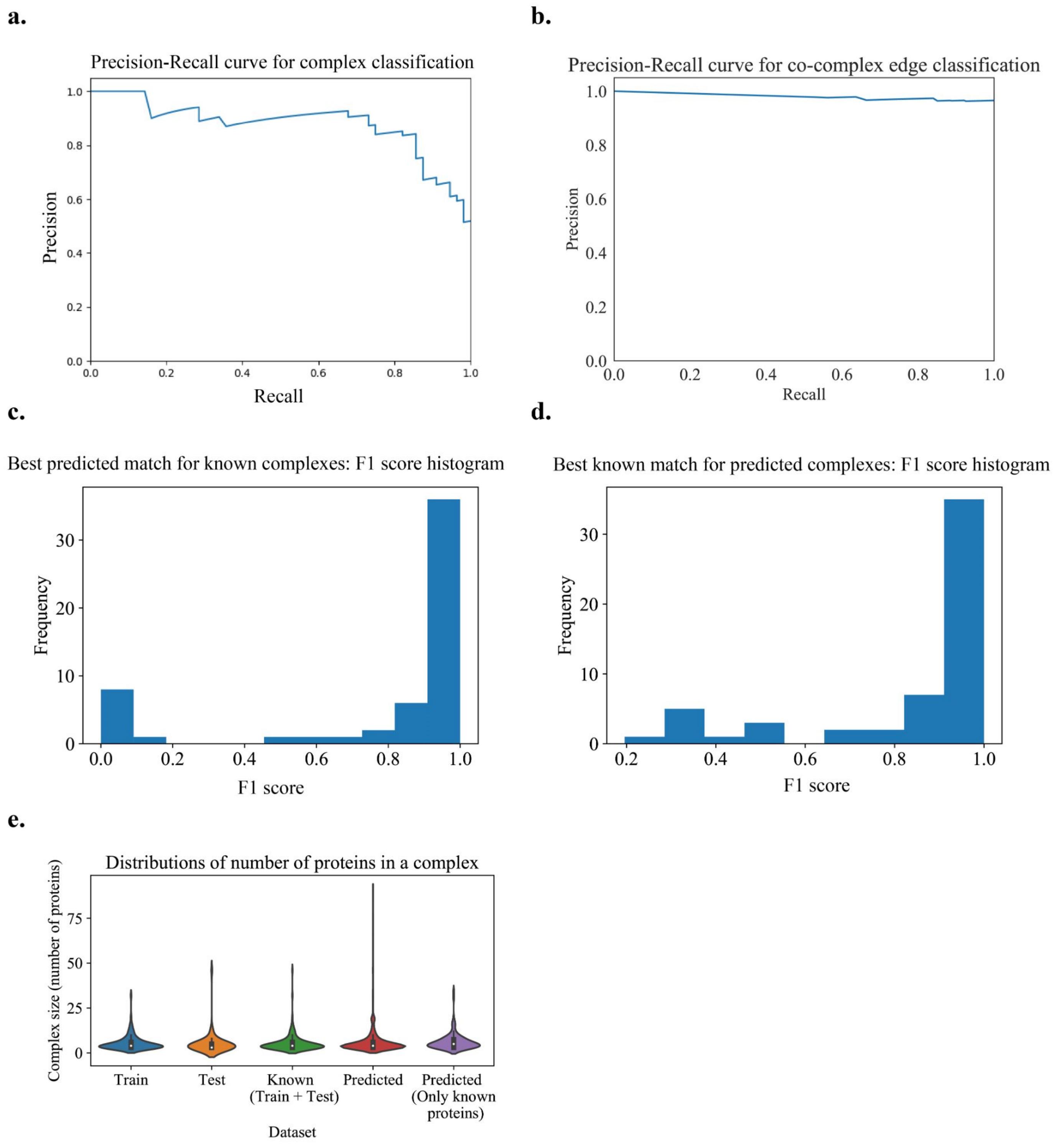


Fig 4. Learned human protein complexes with Super.Complex achieve good PR curves and follow similar size distributions as known complexes. **a.** PR curve for the best model (community fitness function) from the AutoML pipeline on the test dataset, for the task of classifying a subgraph as a community or not. **b.** Co-complex edge classification PR curve for final learned complexes. **c & d.** Best F-similarity score distributions per known complex and per learned complex. **e.** The size distributions of train, test, and all known complexes, learned complexes, and learned complexes after removing known complex proteins.

We use **Fig 4e** to set the maximum number of steps taken in the candidate complex growth stage as 20 and learn a total of 1028 complexes. On removal of non-gold standard proteins from these complexes for evaluation purposes, we obtain 131 complexes We get a good PR curve for the prediction of co-complex edges in comparison with known complex edges, as shown in **Fig 4b**. From **Fig 4c**, we can see that the best learned complex matches for known complexes have high F-similarity scores. Also, from **Fig 4d**, we can see that the best known complex matches for learned complexes have high F-similarity scores. Note that there may be unknown but true complexes that are learned by the algorithm that contribute to false positives.

In **Fig 4e**, we can see that learned complexes have a similar size distribution as known complexes. The small peak at size 20 is an artifact of our threshold on the maximum number of steps that can be taken in growing the complex. This means that either of our stopping criteria was not reached for these complexes, *i.e.* the criteria of a score less than 0.5 or no observed score improvement over a specified number of steps (here, 10).

Evaluation measures comparing learned complexes on hu.MAP by Super.Complex w.r.t known complexes from CORUM are given in **Table 1**, along with the measures computed on the protein complexes comprising hu.MAP obtained from a 2 stage clustering method with the unsupervised ClusterOne algorithm applied first, followed by the unsupervised MCL algorithm. We observe that Super.Complex does better in terms of precision, as can be seen with the higher FMM precision value, while ClusterOne+MCL does better in terms of recall. This can be attributed to more number of complexes learned by ClusterOne+MCL (~4000 compared to ~1000 by Super.Complex) including a few highly overlapping complexes (the maximum pairwise overlap observed was 0.97 Jaccard coefficient), compared to the strict low overlap among complexes learned by Super.Complex (the maximum pairwise overlap observed was 0.36 Jaccard coefficient). We observe 4152 pairs of complexes learned by ClusterOne + MCL having an overlap greater than 0.36 Jaccard coefficient, the maximum pairwise overlap observed in learned complexes from Super.Complex. Note that while the values of F1 evaluation measures are similar, the results from ClusterOne+MCL were achieved by the authors after significant cross-validation, while Super.Complex was faster as detailed in the performance section.

Table 1. Evaluating learned complexes on hu.MAP w.r.t 'refined CORUM' complexes. Refined CORUM comprises 188 complexes after cleaning original CORUM complexes.

Method	FMM			CMF F1 score	Unbiased Sn-PPV accuracy	Qi et al F1 Score (t=0.5)	F-Grand k-Clique	F-weighted k-Clique
	Precision	Recall	F1 score					
Super.Complex	**0.767**	0.534	**0.63**	0.783	0.888	0.739	**0.785**	**0.972**
hu.MAP (ClusterOne + MCL)	0.471	**0.686**	0.559	**0.797**	**0.911**	**0.764**	0.77	0.967

Performance. The current problem of learning the best community fitness function and growing multiple seeds on a network into candidate communities poses an interesting design challenge. Multiple architectures are valid and can be the best based on the constraints of the computing systems. Our design performs well and is capable of giving results in an order of minutes for a network with a similar scale (~10k nodes, ~100k edges) as hu.MAP. With default parameters, a single run of the pipeline on hu.MAP gave reasonably good results in around 30 minutes, on 4 skylake nodes (each with 48 cores @2GHz clock rate). In more detail, it averages ~10min per parameter set of the candidate community sampling part of the pipeline using 4 skylake nodes. For the best results, we ran multiple parameter sweeps for a few hours (a total of around 100 sets), achieving only a minor improvement in results. The AutoML model can yield good models in an order of minutes as well, however, we ran it for a few hours on a single node to obtain the best model. For accessibility to run on non-

HPC (high-performance computing) systems as well, using the optimizations provided and setting low thresholds, it is possible to run the pipeline serially in a matter of hours, even on a personal laptop.

State of the art comparison: Super.Complex achieves good evaluation measures and performance

To compare our method with published results from existing methods, we perform experiments on the data used by these methods in their experiments - a yeast PPI network, DIP -Database of Interacting Proteins [23] with known protein complexes from MIPS - Munich Information Center for Protein Sequence [24] and TAP- Tandem Affinity Purification [25]. Specifically, for an accurate comparison, we use the same PPI network (projection of DIP yeast PPI network on MIPS + TAP proteins) and known protein complexes, available from the ClusterEPs software website. The results from **Table 2** show that our method outperforms all 6 supervised as well as 4 unsupervised methods (by achieving the highest F1 score) in these experiments. We note that like in the experiments with hu.MAP, we obtain high precision values, suggesting that many of the learned protein complexes are of high quality. Further, the whole pipeline (including the AutoML part run on a single skylake node, along with parameter-sweeps for the candidate community sampling part of the pipeline run on 4 skylake nodes - each with 48 cores @2GHz clock rate) was completed in an order of minutes with Super.Complex. We attempted to run other algorithms on hu.MAP as well, but were unsuccessful due to unavailability of code or limited scalability, as detailed below.

Table 2. Comparing our method with 6 supervised and 4 unsupervised methods on a yeast PPI network. Precision, recall, and F-measures are from Qi et al. In experiment 1, for training on TAP and testing on MIPS, in Super.Complex, the best results were obtained when all nodes were used as seeds, with iterative simulated annealing for growth with parameters T_0 as 0.88, α as 1.8 and number of steps as 4, using all neighbors for growth, and overlap Qi measure threshold as 0.1. In experiment 2, for training on MIPS and testing on TAP, in Super.Complex, the best results were obtained when all nodes were used as seeds, with pseudo-metropolis for growth with a probability of 0.1 and number of steps as 9, using all neighbors for growth, and overlap Qi measure threshold as 0.3. In experiment 3, for training on MIPS and testing on MIPS itself, in Super.Complex, the best results were obtained when all nodes were used as seeds, with iterative simulated annealing for growth with parameters T_0as 0.88, α as 1.8 and number of steps as 10, using all neighbors for growth, and overlap Qi measure threshold as 0.9. Evaluation precision, recall, and F-measure are from Qi et al.

	Train	**Test**	**Precision**	**Recall**	**F-measure**
Super.Complex	TAP	MIPS	**0.841**	0.629	**0.72**
ClusterSS	TAP	MIPS	0.526	0.807	0.636
ClusterEPs	TAP	MIPS	0.606	0.664	0.633
RM	TAP	MIPS	0.489	0.525	0.506
SCI-BN	TAP	MIPS	0.219	0.537	0.312
SCI-SVM	TAP	MIPS	0.176	0.379	0.240
ClusterOne		MIPS	0.428	0.435	0.431
COACH		MIPS	0.364	0.495	0.419
CMC		MIPS	0.46	0.38	0.416

MCODE		MIPS	0.4	0.1	0.16
Super.Complex	MIPS	TAP	**0.718**	0.581	**0.642**
ClusterSS	MIPS	TAP	0.477	0.864	0.614
ClusterEPs	MIPS	TAP	0.424	0.782	0.548
RM	MIPS	TAP	0.424	0.433	0.429
SCI-BN	MIPS	TAP	0.312	0.489	0.381
SCI-SVM	MIPS	TAP	0.247	0.377	0.298
ClusterOne		TAP	0.480	0.46	0.47
COACH		TAP	0.387	0.533	0.449
CMC		TAP	0.447	0.353	0.395
MCODE		TAP	0.422	0.127	0.195
Super.Complex	MIPS	MIPS	**0.552**	**0.733**	**0.63**
NN	MIPS	MIPS	0.333	0.491	0.397

We performed experiments with the implementation of the state-of-the-art method SCI-SVM on hu.MAP whose code was available and which we were able to run on a single core of a skylake node (2GHz clock rate). The implementation with the default parameters on a toy dataset and a human dataset resulted in the identification of a large number of small complexes and a very few big complexes, despite sufficient training with big complexes. As big complexes were only a few, the results have low precision and recall. Computed precision, recall, and F1 scores as per equation 10 for even a low threshold *t* of 0.2 are 0.016, 0.0075, and 0.0103 respectively for hu.MAP. A single run of SCI-SVM took around 3 hours to complete and yield these results, unlike better results yielded by Super.Complex in around 30min. Additional runs tuning different parameters in SCI-SVM did not improve results.

In our experiments, we were unable to get the UI (User Interface) for the ClusterEPs software to complete execution on a personal laptop for hu.MAP. Note that Super.Complex on the other hand finishes running on the same personal laptop and provides results after a few hours. Code was not available for other supervised methods.

Learned human protein complexes from Super.Complex, and applications to COVID-19 and characterizing unknown proteins

We provide interactive lists and visualizations of the 1028 learned human protein complexes by Super.Complex, along with refined and original CORUM complexes as a resource on https://sites.google.com/view/supercomplex/super-complex-v3-0. The high precision values obtained by Super.Complex in **Table 1** suggest that many of the learned complexes are of high quality, since the ones with proteins from known complexes match individual known complexes closely. We provide individual community fitness function scores for each of the learned complexes, and rank the list of learned complexes by this score to help identify good candidates for investigation for various applications. In one application, we construct a map of SARS-CoV-2 interactions with learned complexes. In another, we highlight complexes with uncharacterized proteins to provide experimental candidates for functional characterization. These applications are discussed in detail in the next two sections.

Application 1: SARS-CoV-2 affected protein complexes. SARS-CoV-2 is the novel coronavirus that resulted in the pandemic which started at the end of 2019. The virus infects human cells through a spike protein and enters the cell, and subsequently, the proteins of the virus have the potential to interact with multiple human proteins. We preliminarily investigated the human protein complexes that may be affected by the virus, and thus contribute, directly or indirectly to COVID-19, the disease caused by SARS-CoV-2. This study was made possible by the timely work of [26] who identified 332 human proteins that physically interact with SARS-CoV-2 proteins, discovered using AP-MS experiments.

We consider a protein complex to be linked to COVID-19 if any of the proteins in the human complex interact with a SARS-CoV-2 protein, *i.e.*, if the human protein complex contains one or more of the above mentioned 332 human proteins. Super.Complex learns 234 complexes linked to COVID-19, all except one new, *i.e.* not perfectly matching a known CORUM complex. While CORUM contains 430 complexes linked to COVID-19 ('original CORUM'), we obtain 214 complexes after cleaning ('refined CORUM'), *i.e.* removing complexes with sizes lesser than or equal to 2 and using the merging algorithm described in the Materials and Methods section to ensure no more than 50% Jaccard overlap between any two SARS-CoV-2 complexes. All SARS-COV2 linked protein complexes from Super.Complex, scored based on the number of proteins in a complex interacting with a SARS-CoV-2 protein, along with lists of refined CORUM and original CORUM complexes linked to SARS-COV2 are available on the website (https://meghanapalukuri.github.io/Complexes/* where * is Complex2proteins_covid.html, CORUM_Complex2proteins.html, and originalCORUM_Complex2proteins_covid.html respectively). **Fig 5** shows examples of human protein complexes likely especially relevant to the SARS-CoV-2 life cycle, as they involve multiple interactions from SARS-CoV-2 proteins and members of the same (or a functionally related) complex. Further biological analysis on the top-scoring protein complexes learned can perhaps contribute to fighting COVID-19.

Application 2: Uncharacterized proteins and their complexes. 111 uncharacterized proteins (Uniprot [27] annotation score unknown or less than 3) and their corresponding learned 103 complexes are presented on the website (https://meghanapalukuri.github.io/Complexes/* where * is Protein2complex_annotated.html and Complex2proteins_annotated.html). Three examples of uncharacterized proteins (C11orf42, C18orf21, and C16orf91) along with their corresponding complexes are highlighted in **Fig 6**. C11orf42 could potentially be related to trafficking, as it is a part of a complex with 30% similarity to the retromer complex, (*i.e.* with 0.3 Jaccard similarity to the known CORUM retromer complex), with additional evidence available from the Human Protein Atlas (HPA) [28] (available from http://www.proteinatlas.org) showing subcellular localization to vesicles, similar to other proteins of the complex. C18orf21 also has evidence from HPA, localized to the nucleoli and interacting with other proteins of a complex with 50% similarity to the Rnase/Mrp complex with most members in the nucleoli/nucleoplasm. Further evidence from [29] also independently supports C18orf21 as a cellular component of the ribonuclease MRP complex and a participant in ribonuclease P RNA binding as it exhibits significant co-essentiality across cancer cell lines with the POP4, POP5, POP7, RPP30, RPP38, and RPP40 proteins. C16orf91 could potentially be localized to mitochondria like other proteins of the COX 20-C16orf91-UQCC1 complex, with independent experimental evidence from [30].

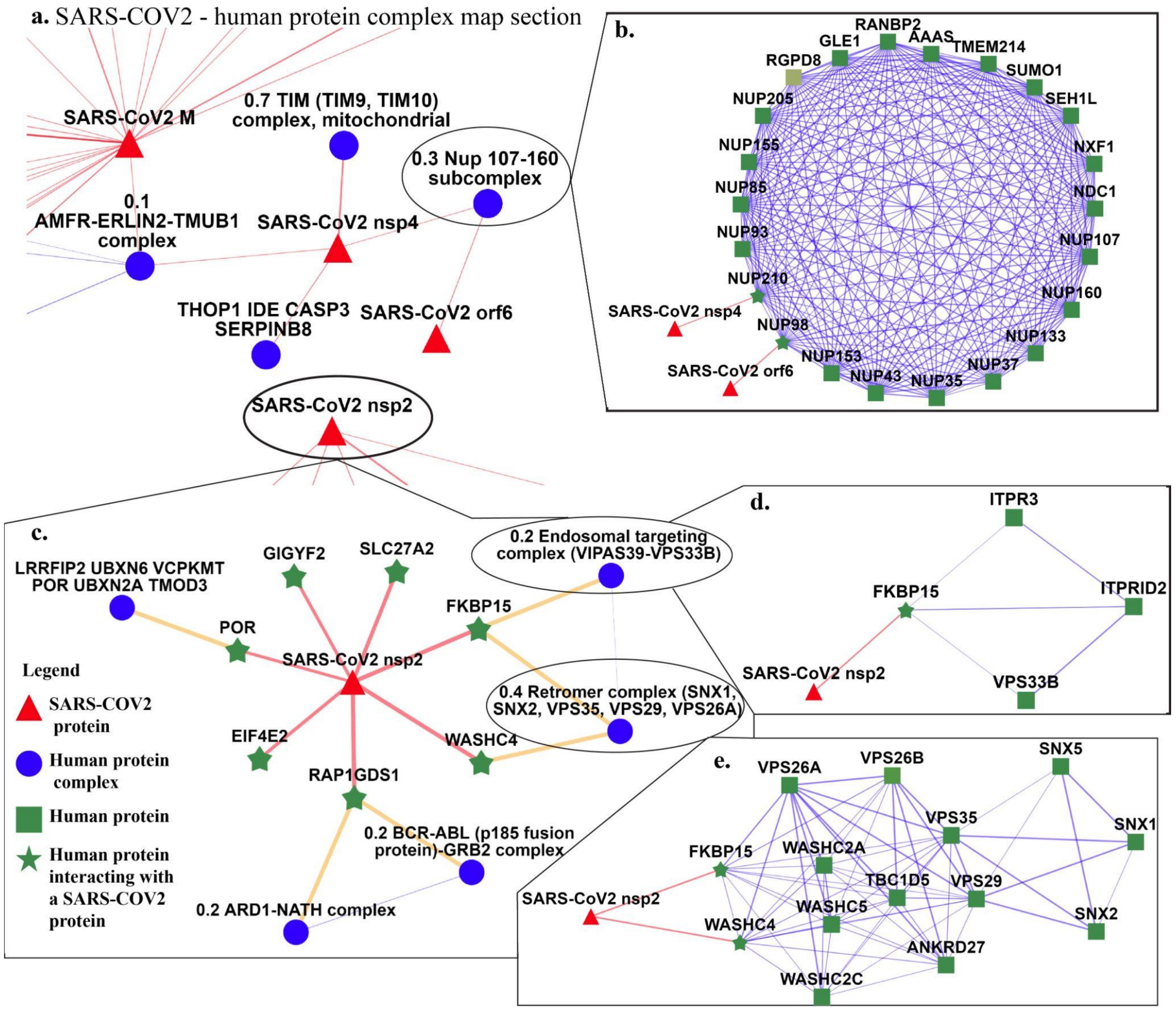
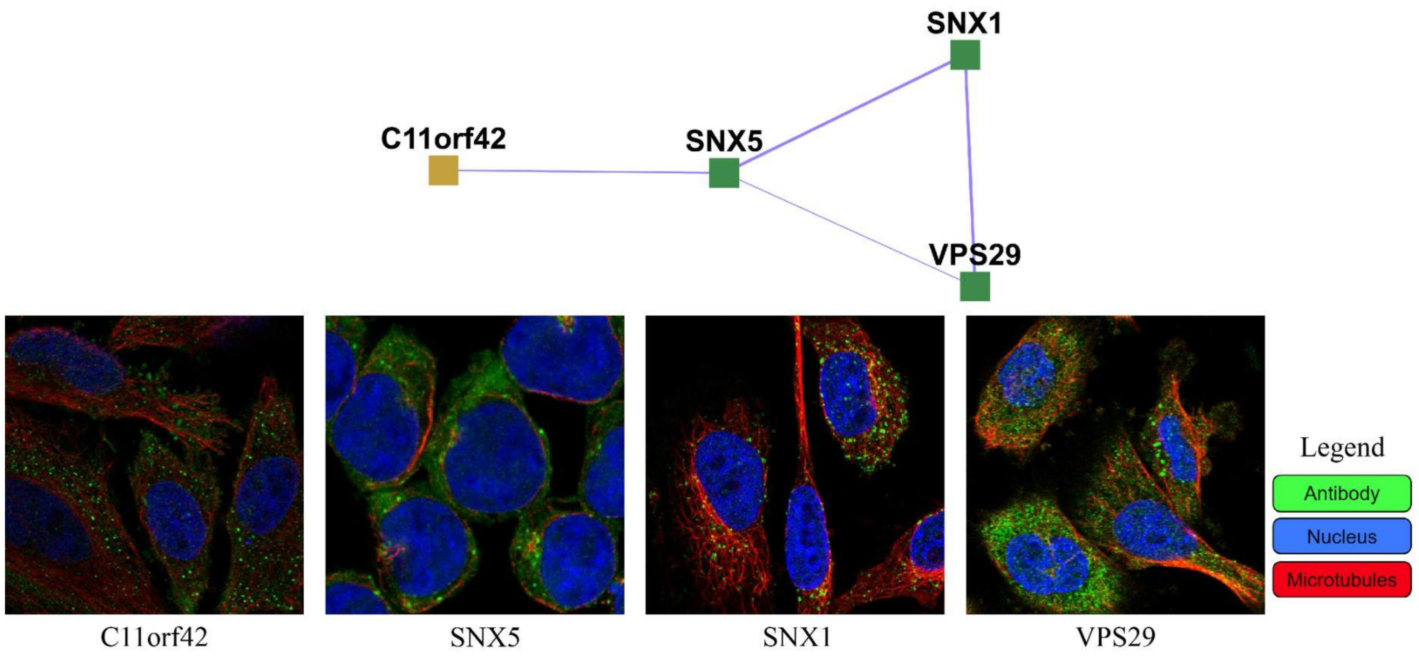
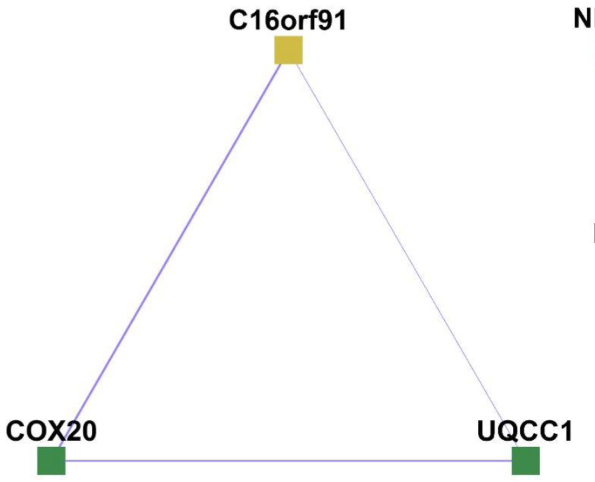


Fig 5. SARS-CoV-2 - human protein complex map showing complexes identified by Super.Complex. a. A section of the full map, featuring SARS-CoV-2 nsp4 and orf6 and their interacting human protein complexes **b.** A protein complex with a 30% match to the Nup 107-160 subcomplex interacts with both SARS-CoV-2 nsp4 and orf6 **c.** Map of SARS-CoV-2 nsp2 interactions with human proteins and their corresponding complexes **d.** A complex with a 20% match to the Endosomal targeting complex, and **e.** A complex with a 40% match to the retromer complex, both of which interact with SARS-CoV-2 nsp2. An interactive map is available at https://meghanapalukuri.github.io/Complexes/SARS_COV2_Map_only_mapped_complexes_names.html.

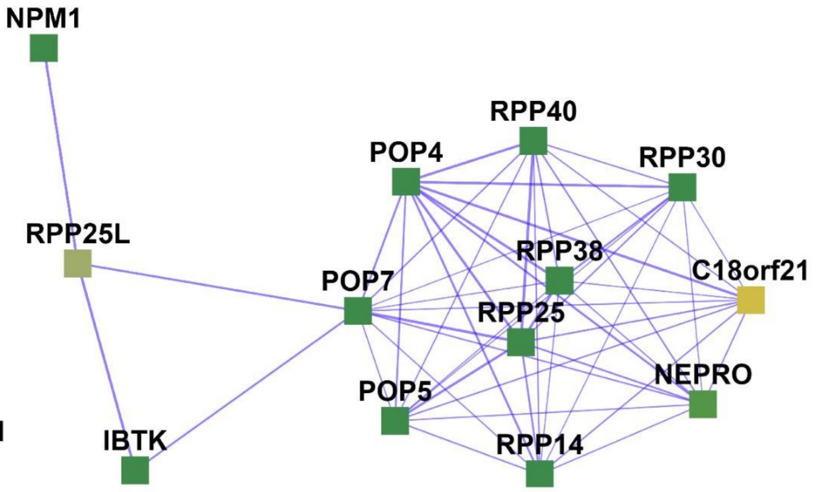
a.



b.



c.



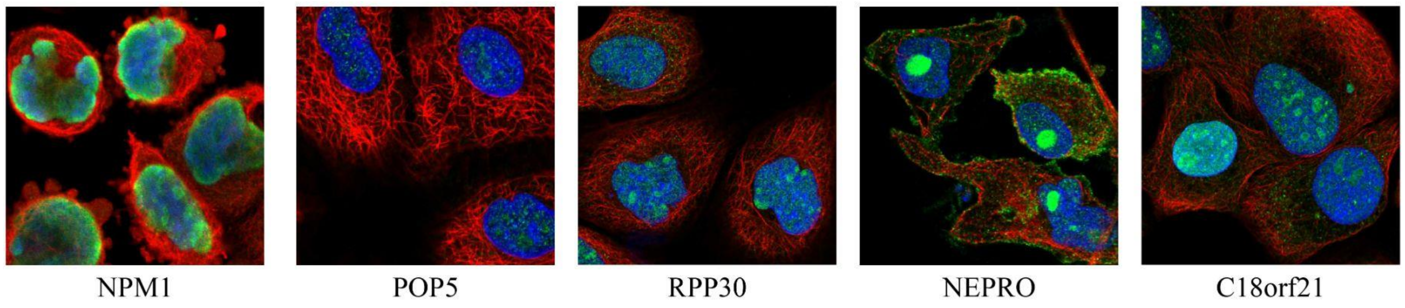


Fig 6. **Examples of complexes with proteins having low annotation scores. a.** C11orf42 constitutes the Retromer complex (SNX1, SNX2, VPS35, VPS29, VPS26A), potentially related to trafficking, with C11orf42 localized in cells to vesicles, similar to the other proteins of the complex (SNX1, SNX5, and VPS29) **b.** C16orf91 constitutes the COX 20-C16orf91-UQCC1 complex, potentially localized to mitochondria like COX20. **c.** C18orf21 constitutes the Rnase/Mrp complex, with C18orf21, localized to nucleoli, closely interacting with nucleoplasm proteins of the complex such as RPP25, POP5, RPP14, NEPRO, RPP30, IBTK, RPP25L, and NPM1. The images of subcellular localization are available from v20.1 of proteinatlas.org, as

https://v20.proteinatlas.org/ENSG00000*/cell, where * is 180878-C11orf42, 028528-SNX1, 089006-SNX5, 111237-VPS29, 167272-POP5, 163608-NEPRO, 148688-RPP30, and 181163-NPM1. Note that localizations were measured in varying cell types, including HeLa, HEL, U2OS, and U-251 MG cells, across the highlighted proteins.

In conclusion, we present Super.Complex, a distributed supervised learning-based community detection algorithm, capable of scaling to large networks that fit on a disk. The pipeline accurately learns communities having different topological structures found in true communities, unlike traditional unsupervised methods that primarily assume and find dense communities. We propose 3 novel, sensitive evaluation measures to compare learned and known communities, addressing different issues with existing evaluation measures. Further analysis of the merits and shortcomings of different evaluation measures, considering various cases of known and learned communities, along with empirical evaluations on different data can help assess and improve evaluation measure formulations. We perform experiments on hu.MAP, a human protein-protein interaction network with ~8k nodes and ~60k edges, identifying 1028 multi-protein complexes, with 234 potentially linked to COVID-19 and 103 complexes with 111 uncharacterized proteins. We hope some of these candidates will serve as candidates for wet-lab experiments for further biological investigation. Our method also outperforms all 6 existing supervised methods, as well as 4 unsupervised methods. The pipeline of Super.Complex is generalizable and can be applied to community detection on networks from various fields, with the possibility of including domain-specific features to learn more accurate domain-specific community fitness functions. Further, learned community fitness functions can be transferred from one field to another where sufficient data on known communities is unavailable. Application of Super.Complex on more datasets, especially in other fields can help demonstrate its generalizability. Open source code along with interactive visualizations of learned human protein complexes are freely available at: https://sites.google.com/view/supercomplex/super-complex-v3-0.

Materials and Methods

Data Preparation

First, the weighted network under consideration is cleaned by removing self-loops, as we do not consider interactions with oneself as a feature of communities. The graph is stored on disk as a set of files, each corresponding to a node and containing a list of the node's neighbors via weighted edges.

Positive communities. Super.Complex takes sets of nodes comprising known communities and obtains their edge information from the induced subgraph of these nodes on the weighted network. Nodes in communities that are absent from the network are removed. Communities with fewer than 3 nodes, communities that are internally disconnected, and duplicate communities are also removed. Constructing the final set of positive communities involves 2 main steps: (i) merging similar communities, and (ii) splitting them into non-overlapping train and test sets.

In the first step, using a merging algorithm we devised, we merge highly similar communities to yield a final list of communities, where no pair of communities have a Jaccard score (equation 6) greater than or equal to j, which we set to 0.6.

$$\text{Jaccard similarity}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \quad (6)$$

Here, $|C|$ is the number of nodes in community C, $|C_1 \cap C_2|$ is the number of nodes common to both communities, and $|C_1 \cup C_2|$ is their union, *i.e.* the total number of nodes in the two communities with each node counted once.

Multiple solutions exist that achieve this goal, however, we want a solution with a large number of communities, *i.e.*, with only a small number of merges performed on the original set of known communities. This is especially important in applications with limited data, such as the human and yeast protein complex experiments in this work. Our algorithm was designed with this objective in mind, and works as follows. The iterative algorithm makes multiple passes through the list of communities performing the merging operation until the specified criterion is achieved. In a single pass of the list of communities, each community is considered in order and merged with the community with which it has the highest overlap (if greater than or equal to 0.6) and the list is updated immediately by removing the original 2 communities and adding the merged community to the end of the list, so that the updated list is available for the next community in consideration. This merging algorithm achieves a lesser number of merges than a trivial merging solution which would merge random pairs of communities that do not satisfy the required criteria until convergence. In practice, the proposed algorithm quickly converges to a solution (*i.e.* the final set has no communities that overlap more than the specified value *j*).

In the second step, the communities are split into non-overlapping training and testing datasets, to emphasize their independence. We obtain sets with equal size distributions and a 70-30 train-test split, as recommended for machine learning algorithms with a small amount of data. Previous algorithms such as [4] and Super.Complex v2.0 [31] discard test communities with sizes greater than a threshold, thus losing out on information from some known communities and which also, in practice, do not yield train-test splits that are close to the recommended 70-30 split. Therefore, we propose the following algorithm. Here, we first make the recommended 70-30 random split into train and test communities. Then we perform iterations of transfers between the two sets until they become independent. In each iteration, we perform two directions of transfers, from train to test and vice-versa, and if the 70-30 split is disturbed, we remove the communities at the end of the list which have extra communities and add them to the other list. In each direction of transfer, for instance, from train to test, we go through the training communities in one pass and if a training community has an overlap (at least one edge) with any of the test communities, it is immediately transferred to the test set, making the updated test set available for comparison with subsequent training set communities. In practice, for many random splits, the algorithm converges fast enough to a solution that is non-overlapping. If for an initial random split, convergence is not achieved after a few iterations, we recommend restarting the algorithm with a different random split.

Negative communities. Negative communities, or non-communities are represented by random walks sampled from the network by growing random seeds, adding a random neighbor at each step. The number of steps ranges from the minimum size to the maximum size of positive communities, with a total number of random walks equal to the number of positive communities multiplied by a scale factor > 1. The random walks are split almost equally across all the sizes, by splitting equally across the different number of steps to be taken for a random walk, to yield an almost uniform size distribution for negative communities. We say almost uniform size distribution, as random walks with the same number of steps need not yield the same sizes, given that the random walk as defined here can revisit edges it has already visited. To achieve random walks of the same size, the algorithm attempts an extra number of random walks and an extra number of steps to achieve the desired random walk size.

The size distribution of positive communities is taken into consideration while training the machine learning model when using a uniform distribution for negatives. We also explore using almost the same size distribution as the positive communities to construct the negative communities. For this, for each size of the positive communities, we construct the negatives by sampling a number of random walks equal to the scale factor times the number of positive communities of this size. However, in this case, we find that there are quite a few missing sizes due to limited positives which may affect the scoring of subgraphs of the missing sizes. Using a uniform distribution would provide more information to learn a more accurate community fitness function that can recognize negatives at sizes missing for positives. In the following feature extraction step, random walks resembling communities are removed. The final number of negative communities is close to the number of positive communities, as we have sampled a slightly higher number of random walks via the scale factor.

Feature Extraction

As communities exhibit different topological structures on the graph, these can be learned by considering useful topological features of communities. Based on graph theory, we extract 18 topological features for each of the positive/negative communities to construct the final train and test data feature matrices, *i.e.* the positive and negative community embeddings. These features include 5 subgraph specific features - the no. of nodes in the subgraph, the 1st 3 singular values of the adjacency matrix of the subgraph, and the weighted intra-cluster density, obtained by using the weighted sum of edges instead of the number of edges in the numerator of the intra-cluster density defined in equation 2. The remaining 13 features are statistical measures of the combinations of features defined for individual nodes of the subgraph. We use the median, mean, variance, and maximum value of the degree d_v of a vertex *v*, which is the sum of the weights of the edges connecting the node to its immediate neighbors. Next, we use the mean, variance, and maximum value of the clustering coefficient of a vertex, c_v, which gives the ratio of the number of triangles it is a part of considering only its first nearest neighbors and the number of all such possible triangles, *i.e.*,

$$c_v = \frac{No. of\ edges\ connecting\ pairs\ of\ the\ neighbors\ of\ v}{No. of\ all\ possible\ edges\ connecting\ pairs\ of\ neighbors\ of\ v} = \frac{|\{(v',v'') \mid v',v'' \in N(v), v' \in N(v'')\}|/2}{|N(v)|^* (|N(v)|-1)/2} \quad (7)$$

Where, *N(v)* is the set of neighbors of vertex v. We use the mean, variance, and maximum value of the degree correlation of a node *v*, which is the average degree of all neighbors of node *v*, *i.e.*,

$$DC_v = \frac{\sum_{v' \in N(v)} d_{v'}}{|N(v)|} \quad (8)$$

The last 3 features are the mean, variance, and maximum value of the weights of edges of the subgraph.

Learning the community fitness function with AutoML

A community fitness function is learned as the decision function of a binary machine learning classifier trained to distinguish the community and non-community embeddings constructed in the previous feature extraction step. For this, we use an AutoML algorithm, *tpot* [18], a genetic algorithm that yields the best model and parameters. It evaluates several preprocessors along with ML models and yields cross-validation scores on the training dataset for each pipeline, which itself is usually a combination of several preprocessors followed by the machine learning model. We configure the algorithm to run in a distributed setting, exploring several combinations of several preprocessors and ML models.

We specify 6 pre-processors that scale the feature matrix. These are - (i) Binarizer, which sets a feature to 0 or 1 based on a threshold, (ii) MaxAbsScaler, which divides the feature by the maximum absolute value of the feature, (iii) MinMaxScaler, which subtracts the minimum of the feature from the feature vector and divides by the range of the feature, (iv) Normalizer, which divides the feature vector by its norm to get a unit norm, (v) RobustScaler, which makes a feature robust to outliers by scaling using the interquartile range and (vi) StandardScaler, which standardizes to the Z-score by subtracting the mean and dividing by the standard deviation of the feature.

We include four feature selecting pre-processors, which are additionally important as we incorporate 6 additional preprocessors that construct combined features. The additional preprocessors include - (i) Decomposition: PCA (Principal Component Analysis), FastICA (Independent Component Analysis), (ii) Feature Agglomeration, (iii) Kernel Approximation methods: Nystroem, Radial Basis Function RBFSampler, (iv) Adding Polynomial Features, (v) Zero counts: Adds the count of zeros and non-zeros per sample as features and (vi) OneHotEncoder for numeric categorical variables. The feature selecting preprocessors include - (i) SelectPercentile, which selects the highest-scoring percentage of features based on 3 univariate statistical tests, FPR - False Positive Rate, FDR - False Discovery Rate and FWE - Family-wise error rate; (ii) VarianceThreshold which removes low variance features, (iii) RFE (recursive feature elimination) using ExtraTrees and (iv) SelectFromModel using ExtraTrees based on importance weights. The ML models included are - (i) Naive Bayes methods using Gaussian, Bernoulli, and Multinomial distributions (ii) Decision Trees, (iii)

Ensemble methods of ExtraTrees, Random Forest, Gradient Boosting and XGB (XGBoost), (iv) K-nearest neighbors, (v) Linear SVMs and (vi) Linear models for Logistic Regression.

The population size and number of generations are provided as parameters for the genetic algorithm of the AutoML pipeline. In practice for our application, giving a value of 50 for each yielded good results. There is an option for a warm start, where you can run additional generations and with additional population sizes starting from the latest results, if the results are unsatisfactory. Additionally, several other machine learning models and preprocessors can also be incorporated into this pipeline, including neural networks. Note that in our experiments, we also obtained pipelines that stack different ML models. We run the pipeline in a distributed manner, setting the number of jobs as the number of processes that run parallelly on a single computer. All the processes on the computer can be used for maximum utilization, however, the documentation notes that memory issues may arise for large datasets. In practice, we set the number of jobs as 20 on a skylake compute node (Intel Xeon Platinum 8160 with 48 cores @2GHz clock rate).

Evaluation. By default 5-fold cross-validation is performed, although this can be modified by a parameter. The pipelines with high cross-validation average precision scores (area under the PR curve) are evaluated on the test dataset to find the best pipeline for our data, to use this for the community fitness function. A one hidden-layer perceptron is also available for training, and comparison with the autoML output to select the best model. We evaluate the performance of the ML binary classifier using accuracies, precision-recall-f1 score measures, average precision score, and PR curves for the test sets while also evaluating these measures for the training set to compare with the test measures and check the bias and variance of the algorithm to make sure it is not underfitting or overfitting the data. We also plot and analyze the size-wise accuracies of the model to understand how the model performs w.r.t to the size of the subgraph it is evaluating.

Candidate community search

Finding a set of maximally scoring candidate communities in a network is an NP-hard problem, as proved by [2]. The proof involves reducing the problem to the problem of finding maximal cliques, which is an NP-hard problem. This is done by using only density as a feature and defining a community fitness function that detects cliques by being equal to 0 unless the density is 1, in which case it is equal to a score $n/(n+1)$ that increases with the size of the clique (n). Since this is an NP-hard problem, algorithms based on heuristics are required to solve it. We explore seeding and growth strategies.

Design and distributed architecture. First, we need to select seeds. Options for seeds include specifying all the nodes of the graph (recommended for best accuracy), all the nodes of the graph present in known communities, a specified number of nodes that will be selected randomly from the graph, or maximal cliques. In the distributed setting using multiple compute nodes, the specified seeds are partitioned equally across compute nodes, and each compute node deals only with the task of growing the seeds assigned to it. In practice, the partitioning is done by a main compute node which partitions the list of seeds and stores the partitioned lists as separate files on the file server. Then it launches one task per compute node (including itself) using the launcher module [32], where a task instructs a compute node to read its respective file containing the seed nodes and run the sampling algorithm starting with each of the seed nodes. On each compute node, we take advantage of all the cores by employing multiprocessing with the *joblib* python library. Each process grows a single seed node into a candidate community and writes it to the compute node's temporary storage. For this process, the graph and the community fitness function are needed. To optimize RAM and as it is impractical to store large networks and machine learning models in memory, we store the graph and the parameters of the community fitness function on disk for each compute node. For this, we copy the graph node files and the trained machine learning model that gives the community fitness function from the file server onto each compute node's temporary storage. Each process reads the model into its memory and uses it to evaluate the neighbors, to pick the neighbor to add to the current subgraph in the growth process from the seed node. The neighbors of the subgraph under consideration at each step of the growth are read from disk on-demand and stored in memory only until they have been evaluated by the fitness function. In this way, we ensure that the processes have a low memory footprint, which can otherwise quickly become a bottleneck for large graphs. We also minimize disk storage by storing each resulting candidate community compactly using only its nodes, as its edges can be inferred if/when necessary by inducing the nodes on the graph. After all the child processes of growing seeds complete on a compute node, the compute node reads the set of learned community files it had stored on its disk and compiles them into a list of candidate communities before writing the list to the file server. The same code also runs in a distributed setting with only one multi-core compute node. There also exists a serial option to run the code without invoking parallel constructs, useful for running on a single core.

Heuristics. Only for the first step of growth, we add the neighbor connected with the highest edge-weight. We provide 2 options for growing the subgraph at each step- an exhaustive neighbor search that is suitable for graphs that are not very large, and an option that optimizes performance by evaluating only a subset of neighbors. In the latter, using a large user-defined threshold t_1, if the number of neighbors of the current subgraph is greater than the threshold, a random sample of the neighbors equal to the provided threshold is chosen for evaluation. Now, of the neighbors, first, an ϵ-greedy heuristic is used to select the neighbor to add to the subgraph. In an ϵ- greedy heuristic, with ϵ probability, a random neighbor is added instead of the maximum scoring neighbor.

In the non-exhaustive search case, in the event of 1-ϵ probability, if the number of neighbors is greater than a 2nd user-defined threshold t_2, a 2nd optimization of cutting down the number of neighbors is applied before evaluating each of the neighbors for choosing the greedy neighbor, as follows. Here, the $t_2$highest neighbors are chosen for evaluation, where the order is decided by sorting the neighbors in descending order based on their maximum edge weight (*i.e.* the highest edge weight among all the edges connecting a neighbor to the subgraph). Note how the first threshold $t_1$ensures that the sorting complexity $O(t_1 log(t_1))$does not blow up.

Note that for efficient constant-time (O(1)) lookup of the maximum edge weight of a neighbor, we store the neighbors of the subgraph as a hash map, where looking up a neighbor yields its maximum edge weight. This hash map also stores, for each neighbor, a list of edges connecting it to the subgraph and was constructed efficiently when the neighbors of each of the subgraph nodes were read from the corresponding file. After selecting the neighbor to add to the subgraph in the current iteration, this hash map is also used to efficiently add the neighbor to the subgraph by providing constant-time lookup to the edges that need to be added.

Instead of the base ϵ-greedy heuristic, we also have a simple base heuristic option, termed greedy edge weight, where we add the neighbor with the highest maximum weight edge at each step of the iteration. Note that since the ML model is not used at each stage of the growth, this is fast enough and does not require the optimization steps used in the ϵ- greedy approach where subsets of neighbors were selected for evaluation by the community fitness function.

For both base heuristics, in any iteration, if no neighbors for the subgraph exist, the growth process terminates. If the community score of the subgraph in any iteration is less than 0.5, the node last added is removed and the growth process terminates. We provide additional heuristics that can be applied on top of the base ϵ-greedy heuristic. Based on the scores of the current and previous iterations of the subgraph, we accept or reject the latest node addition using the user-defined heuristic - iterative simulated annealing (ISA), or a variant of ISA, termed pseudo-metropolis in which the acceptance probability (equation 9) is a constant, *i.e.* $P(S_{new}, S_{old}) = k$. In ISA, at each stage of growth of the current subgraph, its maximum scoring neighbor is added, except in the case when the new community score of the subgraph $S_{new}$is lesser than S_{old}, the value before adding the new node (*i.e.* $S_{new} < S_{old}$). In this case, the new node addition is accepted with a probability of,

$$P(S_{old,}S_{new,}T) = e^{\frac{(S_{new} - S_{old})}{T}} \quad (9)$$

here, starting with hyperparameters T_0 and α, we update the temperature as $T \leftarrow \alpha T$after every iteration.

When ISA or pseudo-metropolis heuristics are applied, we also evaluate an additional heuristic where the algorithm terminates if it has been 10 (or can be user-defined) number of iterations since the score of the subgraph has increased.

In the implementation, we provide four options to the user - greedy edge weight,ϵ-greedy, ϵ-greedy + ISA and ϵ-greedy + pseudo-metropolis. In all options, the algorithm terminates after a number of steps equal to a user-specified threshold. The default threshold provided is the maximum size of the known communities, and we also provide a smart option for when a few communities have a large number of nodes, where it is set to choose the maximum size after ignoring outliers. This number can also be improved by visual inspection of a boxplot of community sizes that is generated. Future work can also explore greedy edge weight + ISA and greedy edge weight + pseudo-metropolis heuristic algorithms and observe their performance. Note how there are 2 possibilities for exploration in the 3 algorithms other than the greedy edge weight heuristic algorithm. In the 1st stage, we pick a neighbor at random with low probability. In the 2nd stage, we accept the neighbor we picked in the 1st stage with low probability, if it yields a lower score than the original subgraph.

Post-processing and cross-validation. Communities with only 2 nodes are removed. We then merge communities that have a Jaccard similarity greater than a specified overlap threshold employing the merging algorithm discussed in the data cleaning section. The only difference is while merging, for two overlapping communities, the final community retained out of the 2 communities or the merged variant is the one that obtains the highest score with the community fitness function. In another variant of the merging algorithm, instead of the Jaccard similarity threshold, we use equation 10 as the threshold and call this the Qi overlap measure.

The parameters ϵin the ϵ-greedy heuristic, k in pseudo-metropolis, $T_0$and αin iterative simulated annealing, and the overlapping threshold in the post-processing step are varied in parameter sweeps to select the best ones that work using the Qi et al F1 score from equation 13.

For evaluation, a similarity measure is used to compare two communities $C_1$and $C_2$and they are determined to be a match if the similarity measure value for the two communities is above a threshold, *i.e.*

$$\text{Qi overlap measure: } \frac{|C_1 \cap C_2|}{|C_1|} > t \text{ and } \frac{|C_1 \cap C_2|}{|C_2|} > t, \text{ where } t \text{ is a set threshold} \quad (10)$$

Here,$|C|$is the number of nodes in community C, and $|C_1 \cap C_2|$ is the number of nodes common to both communities. Once we have a way to match a learned community to a known community or vice-versa as described above, standard machine learning metrics such as precision, recall, and F1 scores can be computed.

$$\text{Qi et al Precision } p = \frac{No.of\ learned\ communities\ that\ match\ known\ communities}{No.of\ learned\ communities} \quad (11)$$

$$\text{Qi et al Recall } r = \frac{No.of\ known\ communities\ that\ match\ learned\ communities}{No.of\ known\ communities} \quad (12)$$

$$Qi\ et\ al\ F1\ score = \frac{2pr}{(p+r)} \quad (13)$$

After parameter sweeps, the results of different heuristics are examined and the one that yields the best F1 score is chosen.

Evaluation

To compare the set of learned communities with known communities, we first construct a set of reduced communities containing nodes only present in known communities. We retain communities with 3 or more nodes only. We use a plethora of evaluation measures, including the proposed measures detailed in the results section, along with existing measures such as equation 10 based precision, recall and F1 score, sensitivity. PPV, their associated accuracy, and MMR. The equation 10 based precision, recall, and F1 score consider one-to-many associations, *i.e.* a learned community can be matched to multiple known communities and vice versa. One-to-one matches are made by measures such as the MMR - maximum matching ratio which calculates the weighted recall (the weights here are neighborhood affinity similarity scores) after selecting a set of one to one matches such that the sum of their similarity scores is maximal. Implicit one-to-one associations are made by measures such as sensitivity (Sn) and positive predictive value (PPV) which resemble a recall-like measure and a precision-like measure respectively. The implicit one-to-one associations made here correspond to picking the community that best matches another community in terms of the number of common nodes. The Sn-PPV accuracy (SPA) combines sensitivity and PPV, resembling a composite measure like the F1 score that combines Precision and Recall. Note that this measure makes one-to-many matches. Letting the set of learned communities be *L* and the set of known communities be *K*, we have,

$$Sn = \frac{\sum_{C_k \in K} \max_{C_l \in L} |C_l \cap C_k|}{\sum_{C_k \in K} |Ck|}, PPV = \frac{\sum_{C_l \in L} \max_{C_k \in K} |C_l \cap C_k|}{\sum_{C_l \in L} \sum_{C_k \in K} |C_l \cap C_k|}, SPA = \sqrt{(Sn^{*} PPV)} \quad (14)$$

We also plot PR curves for edges learned. The learned communities are evaluated against all known communities, only training communities, and only testing communities. We report the results on the test communities for the yeast experiment in **Table 2**.

Algorithm Guarantees

The train and test communities are guaranteed to be independent by the proposed splitting algorithm. Due to the seeding and growth heuristics we employed, the learned communities are guaranteed to be internally connected at every step of the iteration. With an overlap threshold t, the merging algorithm guarantees that no 2 learned communities have an overlap greater than or equal to t. If an overlap threshold of 0 is provided, the merging algorithm yields disjoint, non-overlapping learned communities which guarantee separation, *i.e.* no pair of learned communities can be merged to yield a community of higher community score. The number of maximum iterations provided guarantees that the seeding and growth converge to a solution.

Data and Code Availability

We make interactive visualizations of our learned protein complexes available as a resource at https://sites.google.com/view/supercomplex/super-complex-v3-0, which includes downloadable sets of interactions and complexes, including the 234 complexes that are potentially linked to COVID-19 and SARS-CoV-2 infection, and the set of 111 uncharacterized proteins implicated in 103 complexes. Our code is available at https://github.com/marcottelab/super.complex. To simplify reanalysis, the full interactome datasets are additionally deposited in Zenodo, DOI: http://doi.org/10.5281/zenodo.4814944

Acknowledgments

The authors gratefully acknowledge Benjamin Liebeskind for a tpot code wrapper, Kevin Drew for computing some previous evaluation measures, Claire McWhite for critical reading of the manuscript, and funding from the Welch Foundation (F-1515) and National Institutes of Health (R35 GM122480, R01 HD085901, R21 HD103588) to E.M.M..

References

1. Fortunato S. Community detection in graphs. Phys Rep. 2010 Feb 1;486(3):75–174.
2. Qi Y, Balem F, Faloutsos C, Klein-Seetharaman J, Bar-Joseph Z. Protein complex identification by supervised graph local clustering. Bioinformatics. 2008 Jul 1;24(13):i250–68.
3. Giurgiu M, Reinhard J, Brauner B, Dunger-Kaltenbach I, Fobo G, Frishman G, et al. CORUM: the comprehensive resource of mammalian protein complexes-2019. Nucleic Acids Res. 2019 Jan 8;47(D1):D559–63.
4. Drew K, Lee C, Huizar RL, Tu F, Borgeson B, McWhite CD, et al. Integration of over 9,000 mass spectrometry experiments builds a global map of human protein complexes. Mol Syst Biol. 2017 08;13(6):932.

5. Lee C, Reid F, McDaid A, Hurley N. Detecting highly overlapping community structure by greedy clique expansion. ArXiv10021827 Phys [Internet]. 2010 Jun 15 [cited 2020 Oct 28]; Available from: http://arxiv.org/abs/1002.1827
6. Dongen SM van. Graph clustering by flow simulation [Internet]. 2000 [cited 2019 Dec 9]. Available from: http://dspace.library.uu.nl/handle/1874/848
7. Bader GD, Hogue CW. An automated method for finding molecular complexes in large protein interaction networks. BMC Bioinformatics. 2003 Jan 13;4(1):2.
8. Adamcsek B, Palla G, Farkas IJ, Derényi I, Vicsek T. CFinder: locating cliques and overlapping modules in biological networks. Bioinforma Oxf Engl. 2006 Apr 15;22(8):1021–3.
9. Mete M, Tang F, Xu X, Yuruk N. A structural approach for finding functional modules from large biological networks. BMC Bioinformatics. 2008 Aug 12;9(Suppl 9):S19.
10. Liu G, Wong L, Chua HN. Complex discovery from weighted PPI networks. Bioinformatics. 2009 Aug 1;25(15):1891–7.
11. Wu M, Li X, Kwoh C-K, Ng S-K. A core-attachment based method to detect protein complexes in PPI networks. BMC Bioinformatics. 2009 Jun 2;10(1):169.
12. Nepusz T, Yu H, Paccanaro A. Detecting overlapping protein complexes in protein-protein interaction networks. Nat Methods. 2012 Mar 18;9(5):471–2.
13. Wu J, Lin M. Protein Complex Detection Based on Semi-Supervised Matrix Factorization. In: 2018 37th Chinese Control Conference (CCC) [Internet]. Wuhan: IEEE; 2018 [cited 2020 Dec 9]. p. 8205–8. Available from: https://ieeexplore.ieee.org/document/8484055/
14. Shi L, Lei X, Zhang A. Protein complex detection with semi-supervised learning in protein interaction networks. Proteome Sci. 2011;9(Suppl 1):S5.
15. Yu F, Yang Z, Tang N, Lin H, Wang J, Yang Z. Predicting protein complex in protein interaction network - a supervised learning based method. BMC Syst Biol. 2014;8(Suppl 3):S4.
16. Dong Y, Sun Y, Qin C. Predicting protein complexes using a supervised learning method combined with local structural information. Keskin O, editor. PLOS ONE. 2018 Mar 19;13(3):e0194124.
17. Ziwei Zhou, Yingyi Gui, Yang Z, Xiaoxia Liu, Lei Wang, Yin Zhang, et al. Disease-specific protein complex detection in the human protein interaction network with a supervised learning method. In: 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) [Internet]. Shenzhen, China: IEEE; 2016 [cited 2019 Dec 8]. p. 1296–301. Available from: http://ieeexplore.ieee.org/document/7822705/
18. Randy Olson, Weixuan Fu, Nathan, PGijsbers, Grishma Jena, Tom Augspurger, et al. EpistasisLab/tpot: v0.10.1 minor release [Internet]. Zenodo; 2019 [cited 2019 Dec 9]. Available from: https://zenodo.org/record/2647523#.Xe7Q5Px7nv9
19. Traag VA, Waltman L, van Eck NJ. From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep. 2019 Mar 26;9(1):5233.
20. Borgeson BC. All-by-all discovery of conserved protein complexes by deep proteome fractionation [Internet] [Thesis]. 2016 [cited 2020 Dec 9]. Available from: https://repositories.lib.utexas.edu/handle/2152/46875
21. Karp RM. An algorithm to solve the m × n assignment problem in expected time O(mn log n). Networks. 1980;10(2):143–52.
22. Brohée S, van Helden J. Evaluation of clustering algorithms for protein-protein interaction networks. BMC Bioinformatics. 2006 Dec;7(1):488.
23. Xenarios I, Salwínski L, Duan XJ, Higney P, Kim S-M, Eisenberg D. DIP, the Database of Interacting Proteins: a research tool for studying cellular networks of protein interactions. Nucleic Acids Res. 2002 Jan 1;30(1):303–5.
24. Mewes HW, Amid C, Arnold R, Frishman D, Güldener U, Mannhaupt G, et al. MIPS: analysis and annotation of proteins from whole genomes. Nucleic Acids Res. 2004 Jan 1;32(Database issue):D41-44.
25. Gavin A-C, Aloy P, Grandi P, Krause R, Boesche M, Marzioch M, et al. Proteome survey reveals modularity of the yeast cell machinery. Nature. 2006 Mar 30;440(7084):631–6.
26. Gordon DE, Jang GM, Bouhaddou M, Xu J, Obernier K, White KM, et al. A SARS-CoV-2 protein interaction map reveals targets for drug repurposing. Nature. 2020 Jul;583(7816):459–68.
27. The UniProt Consortium. UniProt: the universal protein knowledgebase in 2021. Nucleic Acids Res. 2021 Jan 8;49(D1):D480–9.
28. Thul PJ, Åkesson L, Wiking M, Mahdessian D, Geladaki A, Blal HA, et al. A subcellular map of the human proteome. Science [Internet]. 2017 May 26 [cited 2021 Apr 16];356(6340). Available from: https://science.sciencemag.org/content/356/6340/eaal3321

29. Wainberg M, Kamber RA, Balsubramani A, Meyers RM, Sinnott-Armstrong N, Hornburg D, et al. A genome-wide atlas of co-essential modules assigns function to uncharacterized genes. Nat Genet. 2021 Apr 15;1–12.
30. Li H, Rukina D, David FPA, Li TY, Oh C-M, Gao AW, et al. Identifying gene function and module connections by the integration of multispecies expression compendia. Genome Res. 2019 Dec 1;29(12):2034–45.
31. Palukuri M, Marcotte E. Supervised Community Detection in Protein-interaction Networks. TACCSTER 2019 Proc [Internet]. 2019 [cited 2020 Oct 29]; Available from: https://repositories.lib.utexas.edu/handle/2152/79826
32. Wilson LA, Fonner JM. Launcher: A Shell-based Framework for Rapid Development of Parallel Parametric Studies. In: Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment [Internet]. New York, NY, USA: Association for Computing Machinery; 2014 [cited 2020 Dec 5]. p. 1–8. (XSEDE '14). Available from: https://doi.org/10.1145/2616498.2616534