AutoGF: Runtime Graph Filter Tuning for Community Node Ranking

Emmanouil Krasanakis, Symeon Papadopoulos, and Ioannis Kompatsiaris

Abstract A recurring graph analysis task is to rank nodes based on their relevance to overlapping communities of shared metadata attributes (e.g. the interests of social network users). To achieve this, approaches often start with a few example community members and employ graph filters that rank nodes based on their structural proximity to the examples. Choosing between well-known filters typically involves experiments on existing graphs, but their efficacy is known to depend on the structural relations between community members. Therefore, we argue that employed filters should be determined not during algorithm design but at runtime, upon receiving specific graphs and example nodes to process. To do this, we split example nodes into training and validation sets and either perform supervised selection between well-known filters, or account for granular graph dynamics by tuning parameters of the generalized graph filter form with a novel optimization algorithm. Experiments on 27 community node ranking tasks across three real-world networks of various sizes reveal that runtime algorithm selection selects near-best AUC and NDCG among a list of 8 popular alternatives, and that parameter tuning yields similar or improved results in all cases.

Keywords Node ranking · Graph signal processing · Parameter tuning

1 Introduction

When graph nodes are attributed (e.g. they are social network users and attributes are their areas of interest), they can be organized into communities of shared metadata

E. Krasanakis (✉) · S. Papadopoulos · I. Kompatsiaris
Centre for Research and Technology Hellas, Information Technologies Institute, Thessaloniki, Greece
e-mail: maniospas@iti.gr

S. Papadopoulos
e-mail: papadop@iti.gr

I. Kompatsiaris
e-mail: ikom@iti.gr

H. Cherifi et al. (eds.), *Complex Networks and Their Applications XI*,
Studies in Computational Intelligence 1078,
https://doi.org/10.1007/978-3-031-21131-7_15

attributes [26]. By definition, these communities are not tied to specific high-level structural characteristics, such as strong connectivity between nodes. Still, it is commonly accepted that attributes could correlate to low-level dynamics leading to the creation of edges, in which case graph structure can help predict metadata. For example, nodes of social network graphs often exhibit homophilous behavior [23], a term describing their tendency to form edges with others of similar attributes. Then, tightly knit structural communities become good predictors of parts of -but not of whole- metadata communities [35].

A recurring graph analysis task, which we tackle in this work, is to rank nodes based on their relevance to communities sharing metadata attributes of interest [16, 19, 29, 32, 33]. Ranking provides greater granularity than clear-cut predictions, for example in the scope of recommending more community members. It also respects overlaps and fuzzy boundaries between communities [21]. Furthermore, node relevance scores obtained during ranking are often the core of more sophisticated systems, such as graph neural networks for classification after initial neural estimations [13, 15] and post-processing strategies that threshold transformations of scores to predict community membership [3].

A popular use case for community node ranking, which we also follow, is to start with a few known community members serving as examples, and inferring the relatedness of all nodes to respective communities based on their structural proximity to the examples [19, 34, 37]. This task is performed independently for one or more communities. Assumptions about what constitutes proximity have coalesced under the field of graph signal processing [Sect. 2], where they are modeled with ad-hoc graph filters and controlled by a small number of parameters [8, 25].[1] Different filters and parameters match different types of communities. For example, filter efficacy could depend on the number of community members [1, 12, 19]. As a result, deployed filters may work well in certain graphs but not necessarily in others. By extension, running filter-based tools 'off-the-shelf' in deployed systems risks producing node ranks of lesser quality.

In this work, we address the above issue by exploiting autotune principles [17] for runtime selection of graph filters. We explore two strategies: a) choosing the best among a list of promising filters, and b) tuning the parameters of a generalized filter form. For the second strategy, we also introduce a novel tuning algorithm that keeps examining a wide search breadth in the solution space but converges within a bounded number of filter runs. The effectiveness of our approach is corroborated on 27 community node ranking tasks across 3 real-world graphs of different domains. Results indicate that neither strategy falls significantly behind best-performing ad-hoc filters when optimizing popular node rank quality measures. Furthermore, parameter tuning frequently captures structural proximity better than ad-hoc assumptions and improves rank quality.

[1] Most non-filter node ranking algorithms, such as k-shell decomposition and variations [36], blindly rank the importance of nodes within graph structures and can not personalize ranks in terms of importance to specific communities.

This paper is organized as follows. In Sect. 2 we present graph filters as an approach for ranking nodes with respect to metadata communities, alongside a generalized literature filter form. In Sect. 3, we describe our runtime filter selection approach and its implementation choices. We also present a novel algorithm for tuning parameters of generalized graph filters. In Sects. 4 and 5 we evaluate our approach in real-world data and discuss practical applicability and potential risks. Finally, in Sect. 6 we summarize our findings and present promising research directions.

2 Background

Graph edges are often represented by adjacency matrices A with elements $A[u, v] = \{1 \text{ if edge } (u, v) \text{ exists, } 0 \text{ otherwise}\}$. These are symmetrically normalized by weighing edges to mitigate the importance of highly connected nodes per:

$$W = D^{-1/2} A D^{-1/2}$$

where D with elements $D[u, v] = \{\sum_{v'} A[u, v'] \text{ if } u = v, 0 \text{ otherwise}\}$ are diagonal matrices of node degrees. The graph's spectrum can be defined as the eigenvalues of the normalized adjacency matrix.[2] In detail, eigenvalue decomposition yields $W = U \Lambda U^{-1}$, where Λ are diagonal matrices of eigenvalues $\Lambda = diag([\lambda_1, \lambda_2, \ldots, \lambda n])$ and U are orthogonal matrices whose columns hold the corresponding eigenvectors. For connected graphs, eigenvalues of the normalized adjacency matrix are real-valued and reside in the unit range $\lambda_i \in [-1, 1]$.

Graph signal processing [24, 27, 28] manipulates signals p whose elements $p[u]$ correspond to values stored at nodes u. To do this, it defines their graph Fourier transform as $\mathcal{F}\{p\} = U^{-1} p$ and its inverse as $\mathcal{F}^{-1}\{\mathcal{F}\{p\}\} = U \mathcal{F}\{p\}$. Then, it observes that $W^n = U \Lambda^n U^{-1} \Rightarrow H(W) = U H(\Lambda) U^{-1}$ for function forms $H(\cdot)$ whose Taylor expansions exist around zero, and defines filters $H_{\mathcal{F}} = [H(\lambda_1), H(\lambda_2), \ldots H(\lambda_n)]$ in the Fourier space, whose parameters arise through transformations $H(\lambda_i)$ of eigenvalues λ_i. Graph filters can be applied on signals via an element-wise multiplication $\odot$ on their Fourier transform $\mathcal{F}\{p\}$. The outcome of filtering in the node space becomes:

$$\mathcal{F}^{-1}(H_{\mathcal{F}} \odot \mathcal{F}\{p\}) = U H(\Lambda) U^{-1} p = H(W) p$$

During the above analysis, the function forms $H(\cdot)$ determining graph filters can be parameterized in terms of their Taylor coefficients $h_0, h_1, \ldots$ per:

$$H(W) = \sum_{k=0}^{\infty} h_k W^k$$

[2] The graph's spectrum can also be defined as the eigenvalues $1 - \lambda_i$ of its normalized Laplacian $I - W$. This, too, can express filters as infinite-degree polynomials of W.

As $W^k p$ propagates graph signals p at k hops away through normalized adjacency matrices W, the above formula describes a weighted aggregation of multi-hop signal propagation. Filters matching different structural assumptions arise from different coefficients h_k. Two well-known filters are personalized PageRank [3, 4] and heat kernels [16]. These respectively adopt degrading hop weights $h_k = (1-a)a^k$ and the kernel $h_k = e^{-t}t^k/k!$ for parameters $a \in [0, 1)$ and $t \in \{1, 2, 3, \dots\}$.

Given the above formulation, graph filtering can rank how nodes pertain to communities of interest [19, 34, 37]. Approaches start with signals p whose values capture whether nodes v belong to sets $\mathcal{C}$ of known community members per:

$$p[v] = \{1 \text{ if } v \in \mathcal{C}, 0 \text{ otherwise}\}$$

Then, for graphs with normalized adjacency matrices W, graph filters $H(W)$ yield new signals $r = H(W)p$ with elements $r[u]$ corresponding to how proximate nodes u are to known member sets C under some understanding of proximity. Finally, nodes are ranked by order of their proximity to known members.

3 Tuning Graph Filters at Runtime

As previously mentioned, graph filters for community node ranking should ideally be selected at runtime, after graphs and example community members become known and therefore can be used to understand underlying structural features. We consider best-performing filters those with higher node rank quality, for instance measured with the area under curve of the receiver operating characteristics (AUC) [6] and the normalized discounted cumulative gain across all graph nodes (NDCG) [14]. Employed measures should coincide with practical objectives on unknown test data. For example, high AUC indicates higher ranks for community members than non-members, whereas high NDCG verifies the community membership of top-ranked nodes.

To optimize node rank quality at runtime, we follow an autotune paradigm that searches through the parameter space of black box algorithms to optimize validation objectives. Originally, the term was associated with specific approaches [17], but nowadays broadly describes automatic selection of machine learning model parameters. This comes at the expense of multiple algorithm runs, but there exist mature solutions for fast computation of graph filters [18].

Our approach starts with sets $\mathcal{C}$ of known community members among graph nodes, which are organized into binary graph signals p per the formulation of Sect. 2. We split known members into non-overlapping subsets $\mathcal{C}_{train}, \mathcal{C}_{valid} \subseteq \mathcal{C}$, which correspond to "training" graph signals p_{train} to be used as filter inputs, and desired output validation signals p_{valid}. We employ evaluation measures $\mathcal{M}(\cdot, \cdot)$, such as AUC or NDCG, that assess node rank quality via pairwise comparison between predictions and ground truth, and select filters with high $\mathcal{M}(r_{train}, p_{valid})$ for predicted ranking scores $r_{train} = H(W)p_{train}$. We avoid overfitting by computing measures

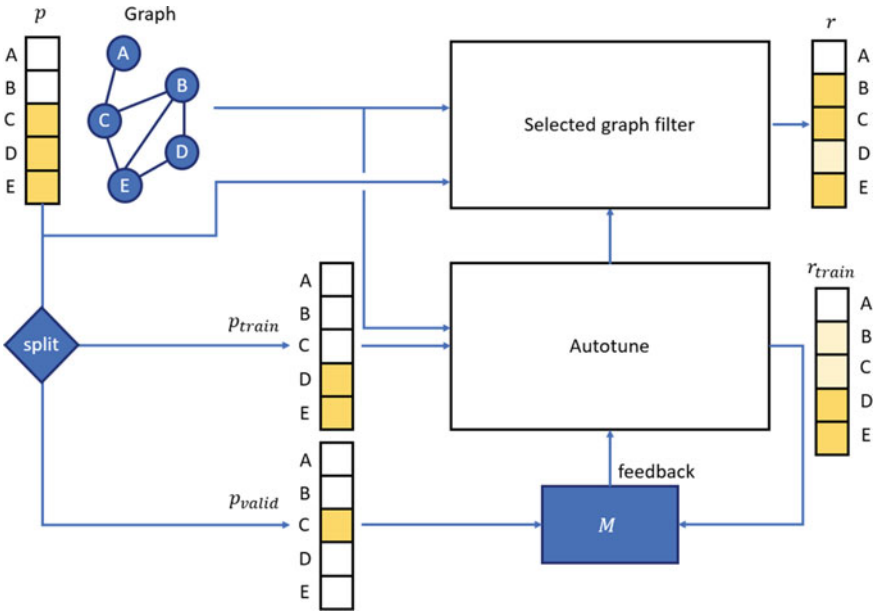

Fig. 1 Overview of graph filter autotuning under measures $\mathcal{M}$. Example nodes are split between training and validation graph signals, where the latter assume the role of ideal training outputs. Highlighted signal elements correspond to higher node values

only across non-training nodes. As long as graphs exhibit homogeneous correlations between communities and edges, filters maximizing validation evaluation are expected to also maximize $\mathcal{M}(r, p_{test})$ for $r = H(W)p$ on nodes other than known community members, where p_{test} are unknown ideal test labels. Our pipeline's data flow is summarized in Fig. 1.

We follow two strategies for graph filter selection by the autotune component of our approach. The first is to perform a simple selection among a list of popular filters, such as those we experiment with later on. The second is to start with the parameterized graph filter form presented in Sect. 2 and tuning a vector of its parameters $h = [h_0, h_1, \ldots, h_K]^T$ to optimize validation objectives. We explore only non-negative parameters to match the widespread literature practice of introducing only non-negative correlations between hops and high-quality node ranks. Then, without loss of generality, we tune all parameters in the range [0, 1].

When tuning graph filter parameters on non-differentiable (potentially even non-convex) validation objectives, a first take is to adopt existing generic black box optimization algorithms [7, 10]. However, these do not guarantee convergence for all deployed system inputs. At the same time, adjusting one graph filter parameter to control the importance of propagating graph signals a fixed number of hops away could drastically affect the validity of other propagation weights. This hypothesis is also corroborated by experiments later on.

To address the above concerns, we propose an algorithm for graph filter parameter tuning that maintains a broad parameter search space while converging in finite time. This involves cycling through parameters, and progressively minimizing a loss function $\ell(h) = 1 - \mathcal{M}(H(W)p_{train}, p_{valid})$ by finding the best permutation around each parameter with coarse linear search. As tuning progresses, we shrink the search range, so that small permutations around ideal values are eventually found. Intuitively, this is equivalent to moving the center of the selected rectangle chosen for each parameter based on subsequent selections of other parameters. If shrinking is slow enough, by the time when parameter permutation breadths become small, potential combinations with drastically different permutations of other parameters have already been considered.

Conceptually, this procedure is a variation of divided rectangles (DIRECT) [9] that, instead of keeping many candidate rectangles to divide, keeps only one, though of larger width than the partition. This practice corresponds to the shrinking radius technique proposed for non-convex block coordinate optimization [22], although the two are not mathematically equivalent due to the finite sum of rectangle widths that limits the optimization within the hypercube of searched parameters instead of looking at an unconstrained range.

In detail, we start from the center of the parameter hypercube and cycle through parameters i. For each of those, we consider the range $\Delta h[i]$ in which to search for new solutions and partition it uniformly to $2P + 1$ candidate points, P of which examine higher parameter values and an equal number lower values. Values are snapped to the search bounds 0 or 1 if they subceed or exceed those respectively. Perturbations form a set H_{search} of potential parameter vectors, of which we select the one minimizing the loss. Finally, we contract the search range by division with constant $T > 1$ and move on to the next parameter. Cycling through parameters stops when loss reduction becomes smaller than a tolerance ϵ across all parameters. This process is outlined in Algorithm 1.

Algorithm 1 Parameter tuning

Inputs: parameter loss $\ell(h)$, tolerance ϵ, line search partitions P, range shrinking T
Outputs: near-optimal vector of K parameters
$h \leftarrow [0.5] \times K$, $\Delta h \leftarrow [0.5] \times K$, $err \leftarrow [\infty] \times K$, $i \leftarrow 0$
while $\max_i err[i] > \epsilon$ **do**
  $u_i \leftarrow$unit vector with $u_i[j] = \{1 \text{ if } i = j, 0 \text{ otherwise}\}$
  $H_{search} \leftarrow \{max(0, min(1, h + u_i \cdot \Delta h[i] \cdot (p/P - 1))) \mid p = 0, 1, \ldots, 2P\}$
  $err[i] \leftarrow \ell(h) - \min_{h \in H_{search}} \ell(h)$
  $h \leftarrow \arg\min_{h \in H_{search}} \ell(h)$
  $\Delta h[i] \leftarrow \Delta h[i]/T$
  $i \leftarrow (i + 1) \, mod \, K$
return h

If the objective $\ell(h)$ is Lipschitz continuous with Lipschitz constant $L < \infty$ (when the loss is differentiable, this means that $\sup \|\nabla \ell(h)\| \leq L$), it is easy to see that the division of the parameter permutation radius by T every K iterations lets the algo-

rithm run in amortized time $O\big(K(\text{run } \ell(h)) \log_T \frac{L}{\epsilon}\big)$. If graph nodes are fewer than edges (as happens for connected graphs), in which case the running time of $\ell(h)$ is not dominated by node validation. Using sparse matrix multiplication to iteratively compute Krylov space elements $\{W^k p_{train} |, k = 0, \ldots, K\}$ by left-multiplying previous ones with W, graph filters run in time $O(KE)$, where E is the number of graph edges. Thus, our graph filter parameter tuning mechanism can be implemented to run in amortized time:

$$O\big(K^2 E(\log_T L - \log_T \epsilon)\big)$$

Running time scales linearly with the number of edges and quadratically with the number of parameters. We recommend and employ default parameters $P = 2, T = 1.01$, which suffice to minimize the Beale and Booth functions often used in optimization benchmarks [2] to 10^{-6} parameter (instead of loss) tolerance.

4 Experiment Setup

We experiment on three publicly available real-world graphs with metadata communities. First is the Amazon co-purchasing graph [20], whose nodes and edges correspond to products and frequent co-purchases. Products are organized into metadata communities based on their type (e.g. book, movie) attribute. Second is the Citeseer citation graph [11], whose nodes and edges correspond to scientific publications and citations. Publications are organized into communities based on scientific field. Third is the Maven dependency graph [5], whose nodes and edges correspond to software projects and dependencies. Projects are organized into communities based on the organization responsible for their development.

These graphs were chosen for experimentation on merit of comprising metadata communities with enough member nodes to conduct robust validation. To not overrepresent graphs with many communities and obtain enough validation nodes later on, we experiment with the first three communities of each graph with at least 500 nodes. We treat all edges as undirected so that symmetric normalization of filters is applicable. Community details are summarized in Table 1.

For each of the the above-described communities, we generate three splits of known-test members by assuming that known members are uniformly sampled to comprise 10, 20, or 30% of total members. We remind that validation nodes can only be subsampled from known members. Sampling is seeded to ensure reproducibility and fair comparison between approaches. In total, experiments on 9 communities create $9 \cdot 3 = 27$ different known-test member splits. For each split, we consider two different node ranking objectives; optimizing AUC, and optimizing NDCG. Thus, we obtain $27 \cdot 2 = 54$ experiment setups. Sampling and splits are seeded so that evaluations of different graph filters in the same setups are comparable.

We investigate the ability of our approach to produce high-quality community node ranks compared to ad-hoc graph filters and parameters often encountered in the

Table 1 Details of communities we experiment on

Community	Graph	Nodes	Edges	Members
amazon0	Amazon	554,789	3,577,450	280,507
amazon1	Amazon	554,789	3,57,7450	64,915
amazon2	Amazon	554,789	3,577,450	17,966
citeseer0	Citeseer	3327	9464	596
citeseer1	Citeseer	3327	9464	668
citeseer2	Citeseer	3327	9464	701
maven0	Maven	1,965,359	19,431,302	1687
maven1	Maven	1,965,359	19,431,302	1043
maven2	Maven	1,965,359	19,431,302	49,883

literature. We compare the following alternatives, all of which we integrated in the *pygrank* Python library [18] alongside experiment setups:

- *ppr a* [3, 4, 25]. Personalized PageRank that performs stochastic random walks with restart probabilities $1 - a$ at each step [29]. We test common values $a \in \{0.5, 0.85, 0.9, 0.99\}$ and compute filters to numerical tolerance 10^{-9}.
- *hk k* [8]. Heat kernels that form bandpass windows around desired propagation hops k. We test common window centers $k \in \{2, 3, 5, 7\}$.
- *select* [this work]. Runtime selection of the best among ppr a and hk k by withholding a 10% validation subset of known community members. When graphs are unknown during algorithm selection, this becomes a baseline for tuning.
- *tune* [this work]. Tuning a generalized graph filter with 40 parameters, where the filter is obtained with non-zero Taylor coefficients $h_0 = 1$ and tuned $h_1, \ldots, h_{40}$ via Algorithm 1 towards maximizing measures of choice on the same 10% validation subset as in *select*. Optimization absolute deviation tolerance is set to $\epsilon = 10^{-6}$.
- *tuneLBFGSB* [ablation study]. A variation of *tune* that substitutes our tuning algorithm with the L-BFGS-B optimizer [7] provided by the *scipy* library [31] with default parameters and 10^{-6} percentage decrease on the evaluation function as a stropping criterion to make sure that tuning does not stop early. This is a popular optimizer still used for parameter search [30] and approximates Neuton's method while limiting the number of computations to only first-order gradients. Experiments with the Nelder-Mead optimizer yielded similar or worse results that we do not report due to space constraints.

5 Experiment Results

Tables 2 and 3 present the quality of community node ranking across experiment setups in terms of AUC and NDCG respectively. Before exploring graph filter selection, we verify that individual ad-hoc filter efficacy varies across communities and

training-test splits. Indeed, no explored filter outperforms the rest in all experiments. For instance, ppr0.99 is often the best in Amazon communities, but also the worst in Maven communities, where it lags behind others up to 0.035 AUC. Runtime filter selection would be useful as long as it lags less behind.

Choosing between ad-hoc filters with our validation strategy does not always retrieve the best-performing ones. We attribute this behavior to few missing examples still impacting the ideal filter propagation weights needed for high-quality node ranking. Withholding fewer nodes could degrade validation robustness and future research could investigate new mechanisms to improve generalization. For the time being, selection of best among existing alternatives at runtime chooses the best filters in 31/54 settings. But, even when this scheme fails to identify the best filter, it often retrieves near-best ones that at worst lag behind only by 0.011 in terms of AUC or NDCG, where this gap usually shrinks to 0.001.

Parameter tuning with Algorithm 1 outperforms all ad-hoc filters in 40/54 experiment settings. This induces up to 0.010 AUC and 0.033 NDCG improvements, indicating that it manages to discover nuanced notions of structural proximity. It lags behind by at worst 0.007 on account of either measure, and often by much less. Compared to selecting among filters, tuning yields better evaluation outcomes in 49/54 of experiment settings. As such, we recommend it as an out-of-the-box solution for community node ranking in new graphs, especially if structural characteristics correlating to the formation of communities are not known beforehand. Finally, comparing our optimization algorithm to L-BFGS-B, the latter induces marginal improvements in the Citeseer graph, but falls significantly behind -even compared to filter selection- in the Amazon and Maven graphs. This corroborates the need for retaining a wide parameter search space.

In relation to applying our methodology, we experimented on communities with enough example members to achieve a robust evaluation when randomly withholding 10% of them. Fewer known members may not yield robust validation strategies and we hereby caution against blindly applying our methodology when too few members are known. In principle, we expect our approach to work well -and therefore be applicable on- community node ranking based on at least the same number of known members (at least 50) as in our experiments.

As evidence that tuning discovers non-trivial graph propagation schemes, Fig. 2 shows the first 41 parameters of high-AUC filters for citeseer0 with 30% known members. There, tuning discovers a different propagation strategy than ad-hoc filters, which subsequently manages to (slightly) improve the best filter in Table 2. Moreover, Fig. 3 shows that tuning is tailored not only to communities but even to specific sets of example nodes, yielding drastically different filters for the same communities. Given that tuned graph filters generally outperform others, this finding corroborates our hypothesis that filters should be selected at runtime to match the characteristics of data they are about to process. Finally, filter differences between different fractions of community examples support our practice of withholding only a small fraction of validation nodes.

Table 2 Test set AUC of community node ranks for ad-hoc filters and those obtained through runtime tuning on the same measure

		Ad-hoc								Autotune		
Com.	Examples (%)	ppr0.5	ppr0.85	ppr0.9	ppr0.99	hk2	hk3	hk5	hk7	Select	Tune	tuneLBFGSB
amazon0	10	0.825	0.844	0.853	0.901	0.823	0.825	0.832	0.844	0.901	**0.903**	0.883
amazon0	20	0.820	0.855	0.870	0.910	0.817	0.824	0.844	0.865	0.910	**0.918**	0.900
amazon0	30	0.815	0.868	0.884	0.914	0.810	0.824	0.855	0.880	0.914	**0.924**	0.908
amazon1	10	0.930	0.934	0.935	**0.944**	0.930	0.930	0.931	0.933	**0.944**	0.943	0.940
amazon1	20	0.941	0.946	0.948	0.954	0.941	0.941	0.944	0.946	0.954	**0.955**	0.953
amazon1	30	0.946	0.953	0.955	0.960	0.946	0.947	0.951	0.954	0.960	**0.961**	0.959
amazon2	10	0.960	0.964	0.965	**0.968**	0.959	0.960	0.961	0.963	**0.968**	**0.968**	0.967
amazon2	20	0.970	0.974	0.974	0.975	0.970	0.971	0.972	0.974	0.975	**0.977**	0.976
amazon2	30	0.973	0.976	0.977	0.976	0.972	0.973	0.975	0.976	0.977	**0.979**	0.978
citeseer0	10	0.778	0.789	0.792	0.793	0.775	0.777	0.781	0.784	0.793	**0.795**	**0.795**
citeseer0	20	0.841	0.849	0.850	0.845	0.838	0.840	0.844	0.847	0.845	**0.852**	**0.852**
citeseer0	30	0.859	0.867	0.868	0.861	0.856	0.859	0.863	0.866	0.868	**0.869**	**0.869**
citeseer1	10	0.805	**0.807**	**0.807**	0.798	0.806	0.805	0.805	0.806	0.798	0.805	0.806
citeseer1	20	0.820	0.823	**0.824**	0.813	0.820	0.820	0.821	0.823	0.820	0.819	0.823
citeseer1	30	0.816	0.821	0.821	0.811	0.816	0.816	0.819	0.820	0.816	**0.821**	**0.821**
citeseer2	10	0.659	0.669	0.672	0.675	0.656	0.657	0.661	0.664	0.656	**0.670**	0.676
citeseer2	20	0.718	0.727	0.730	0.731	0.716	0.717	0.721	0.724	0.716	0.732	**0.733**
citeseer2	30	0.767	0.773	0.774	0.769	0.766	0.767	0.770	0.773	0.773	0.775	**0.776**
maven0	10	**0.998**	0.997	0.995	0.942	**0.998**	**0.998**	**0.998**	0.997	**0.998**	0.994	0.989
maven0	20	0.997	0.995	0.994	0.925	0.997	0.997	0.997	0.996	0.997	**0.998**	0.984
maven0	30	0.997	0.995	0.993	0.911	**0.998**	0.997	0.997	0.995	**0.998**	**0.998**	0.982
maven1	10	0.996	0.993	0.991	**0.967**	0.996	0.996	0.996	0.995	0.996	0.995	0.986
maven1	20	**0.996**	0.992	0.990	0.961	**0.996**	**0.996**	0.995	0.994	0.994	0.995	0.984
maven1	30	0.997	0.992	0.989	0.956	**0.998**	0.997	0.996	0.995	0.992	0.992	0.982
maven2	10	**0.998**	**0.998**	**0.998**	0.993	0.997	**0.998**	**0.998**	**0.998**	**0.998**	**0.998**	0.997
maven2	20	**0.997**	**0.997**	**0.997**	0 .991	**0.997**	**0.997**	**0.997**	**0.997**	**0.997**	**0.997**	0.996
maven2	30	0.997	0.997	0.997	0.990	0.997	0.997	**0.998**	0.997	0.997	0.997	0.996
Average		0.897	0.903	0.905	0.898	0.896	0.897	0.901	0.904	0.909	**0.912**	0.908

The highest value is bolded

Table 3 Test set NDCG of community node ranks for ad-hoc filters and those obtained through runtime tuning on the same measure

		Ad-hoc								Autotune		
Com.	Examples (%)	ppr0.5	ppr0.85	ppr0.9	ppr0.99	hk2	hk3	hk5	hk7	select	tune	tuneLBFGSB
amazon0	10	0.970	0.976	0.978	0.985	0.970	0.971	0.975	0.978	0.985	**0.987**	0.983
amazon0	20	0.968	0.976	0.979	0.984	0.967	0.970	0.975	0.978	0.984	**0.987**	0.983
amazon0	30	0.965	0.976	0.978	0.983	0.964	0.968	0.974	0.978	0.983	**0.987**	0.982
amazon1	10	0.955	0.962	0.964	**0.972**	0.954	0.956	0.960	0.963	**0.972**	**0.972**	0.969
amazon1	20	0.954	0.963	0.966	0.972	0.953	0.956	0.962	0.965	0.972	**0.976**	0.971
amazon1	30	0.951	0.963	0.966	0.970	0.950	0.955	0.961	0.966	0.970	**0.976**	0.970
amazon2	10	0.912	0.923	0.927	**0.933**	0.910	0.913	0.920	0.925	0.920	0.927	0.934
amazon2	20	0.914	0.928	0.932	0.933	0.913	0.917	0.926	0.931	0.933	**0.941**	0.937
amazon2	30	0.912	0.926	0.929	0.929	0.910	0.915	0.924	0.929	0.929	**0.938**	0.933
citeseer0	10	0.896	0.900	0.902	0.902	0.894	0.895	0.897	0.899	0.902	0.903	**0.904**
citeseer0	20	0.922	0.931	0.933	0.926	0.920	0.923	0.928	0.931	0.926	0.932	**0.934**
citeseer0	30	0.915	0.925	0.926	0.923	0.907	0.918	0.923	0.926	0.923	**0.930**	0.929
citeseer1	10	0.905	0.905	0.905	0.904	0.905	0.905	0.904	0.903	0.904	**0.907**	**0.907**
citeseer1	20	0.901	0.903	0.904	0.895	0.900	0.901	0.903	0.903	0.903	0.897	**0.905**
citeseer1	30	0.892	0.893	0.892	0.877	0.891	0.892	**0.893**	0.892	0.892	0.890	0.889
citeseer2	10	0.869	0.876	0.878	0.879	0.867	0.868	0.872	0.875	0.868	**0.883**	0.881
citeseer2	20	0.887	0.896	0.897	0.899	0.886	0.888	0.890	0.895	0.886	0.898	**0.900**
citeseer2	30	0.894	0.904	0.905	0.903	0.893	0.896	0.902	0.905	0.905	**0.908**	**0.908**
maven0	10	0.803	0.786	0.778	0.723	0.803	0.804	0.796	0.774	0.796	**0.813**	0.745
maven0	20	0.738	0.708	0.693	0.615	0.743	0.741	0.724	0.694	0.743	**0.744**	0.637
maven0	30	0.673	0.637	0.625	0.554	**0.676**	0.674	0.651	0.629	0.674	0.670	0.581
maven1	10	0.812	0.823	0.821	0.781	0.807	0.817	0.828	0.830	0.828	**0.863**	0.814
maven1	20	0.806	0.814	0.812	0.767	0.788	0.800	0.817	0.818	0.818	**0.831**	0.801
maven1	30	0.770	0.774	0.773	0.728	0.767	0.775	0.780	0.773	0.773	**0.825**	0.762
maven2	10	0.904	0.907	0.906	0.845	0.903	0.908	0.911	0.909	0.911	**0.935**	0.889
maven2	20	0.863	0.864	0.861	0.785	0.862	0.867	0.869	0.865	0.869	**0.903**	0.840
maven2	30	0.812	0.812	0.808	0.729	0.811	0.816	0.818	0.813	0.818	**0.861**	0.786
Average		0.880	0.883	0.883	0.863	0.878	0.882	0.885	0.883	0.888	**0.899**	0.877

The highest value is bolded

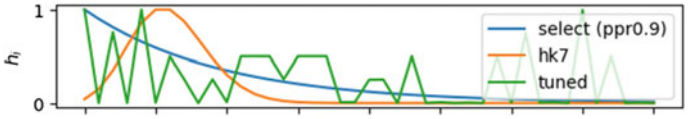

Fig. 2 Parameters h_i of filters with high AUC on citeseer0 with 30% examples

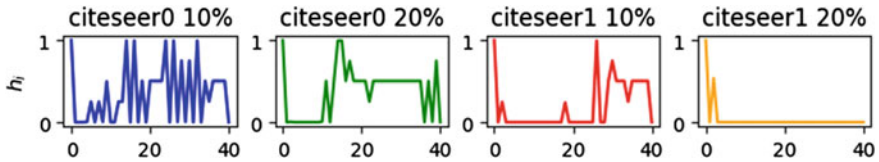

Fig. 3 Parameters tuned on citeseer0 and citeseer1 with 10 and 20% examples

6 Conclusions and Future Work

This work introduces a runtime graph filter selection scheme for community node ranking based on known member nodes. Selection involves either choosing between promising filters or tuning the parameters of a generalized filter form. For the latter, we introduced a novel algorithm that meshes parts of previous alternatives to satisfy both scalability and a wide parameter search breadth needed by graph filters. We verified the efficacy of our approach with experiments across real-world graph communities, where we found that, given enough example community members to satisfy robust evaluation by withholding a few of them, our methodology (especially tuning) yields filters with similar or better AUC and NDCG than alternatives. Thus, we recommend its adoption in practice.

In the future, we are interested in experimenting on more graphs, improving our tuning algorithm, and theoretically probing its optimality. More robust evaluation could also be devised to autotune from fewer known community members.

Acknowledgements This work was partially funded by the European Commission under contract number H2020-951911 AI4Media.

References

1. Abu-El-Haija, S., Kapoor, A., Perozzi, B., Lee, J.: N-gcn: Multi-scale graph convolution for semi-supervised node classification. In: Uncertainty in Artificial Intelligence, pp. 841–851. PMLR (2020)
2. Al-Roomi, A.R.: Unconstrained Single-Objective Benchmark Functions Repository (2015). https://www.al-roomi.org/benchmarks/unconstrained
3. Andersen, R., Chung, F., Lang, K.: Local partitioning for directed graphs using pagerank. Internet Math. **5**(1–2), 3–22 (2008)
4. Bahmani, B., Chowdhury, A., Goel, A.: Fast incremental and personalized pagerank. Proc. VLDB Endow. **4**(3) (2010)

5. Benelallam, A., Harrand, N., Valero, C.S., Baudry, B., Barais, O.: Maven central dependency graph (2018)
6. Bradley, A.P.: The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recogn. **30**(7), 1145–1159 (1997)
7. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. **16**(5), 1190–1208 (1995)
8. Chung, F.: The heat kernel as the pagerank of a graph. Proc. Nat. Acad. Sci. **104**(50), 19735–19740 (2007)
9. Finkel, D.E., Kelley, C.: Additive scaling and the direct algorithm. J. Glob. Optim. **36**(4), 597–608 (2006)
10. Galántai, A.: Convergence of the Nelder-Mead method. Numer. Algorithms, 1–30 (2021)
11. Getoor, L.: Link-based classification. In: Advanced Methods for Knowledge Discovery from Complex Data, pp. 189–207. Springer (2005)
12. Grover, A., Leskovec, J.: node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864 (2016)
13. Huang, Q., He, H., Singh, A., Lim, S.N., Benson, A.R.: Combining label propagation and simple models out-performs graph neural networks. arXiv:2010.13993 (2020)
14. Järvelin, K., Kekäläinen, J.: Ir evaluation methods for retrieving highly relevant documents. In: ACM SIGIR Forum, vol. 51, pp. 243–250. ACM, New York, NY, USA (2017)
15. Klicpera, J., Bojchevski, A., Günnemann, S.: Predict then propagate: graph neural networks meet personalized pagerank. arXiv:1810.05997 (2018)
16. Kloster, K., Gleich, D.F.: Heat kernel based community detection. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1386–1395 (2014)
17. Koch, P., Golovidov, O., Gardner, S., Wujek, B., Griffin, J., Xu, Y.: Autotune: a derivative-free optimization framework for hyperparameter tuning. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 443–452 (2018)
18. Krasanakis, E., Papadopoulos, S., Kompatsiaris, I., Symeonidis, A.: pygrank: a python package for graph node ranking. arXiv:2110.09274 (2021)
19. Krasanakis, E., Schinas, E., Papadopoulos, S., Kompatsiaris, Y., Symeonidis, A.: Boosted seed oversampling for local community ranking. Inf. Process. Manage. **57**(2), 102053 (2020)
20. Leskovec, J., Adamic, L.A., Huberman, B.A.: The dynamics of viral marketing. ACM Trans. Web (TWEB) **1**(1), 5-es (2007)
21. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. Internet Math. **6**(1), 29–123 (2009)
22. Lyu, H.: Convergence of block coordinate descent with diminishing radius for nonconvex optimization. arXiv:2012.03503 (2020)
23. McPherson, M., Smith-Lovin, L., Cook, J.M.: Birds of a feather: homophily in social networks. Ann. Rev. Sociol. **27**(1), 415–444 (2001)
24. Ortega, A., Frossard, P., Kovačević, J., Moura, J.M., Vandergheynst, P.: Graph signal processing: overview, challenges, and applications. Proc. IEEE **106**(5), 808–828 (2018)
25. Page, L., Brin, S., Motwani, R., Winograd, T.: The Pagerank Citation Ranking: Bringing Order to the Web. Tech. rep. Stanford InfoLab (1999)
26. Papadopoulos, S., Kompatsiaris, Y., Vakali, A., Spyridonos, P.: Community detection in social media. Data Min. Knowl. Disc. **24**(3), 515–554 (2012)
27. Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A., Vandergheynst, P.: The emerging field of signal processing on graphs: extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Process. Mag. **30**(3), 83–98 (2013)
28. Stanković, L., Daković, M., Sejdić, E.: Introduction to graph signal processing. In: Vertex-Frequency Analysis of Graph Signals, pp. 3–108. Springer (2019)
29. Tong, H., Faloutsos, C., Pan, J.Y.: Fast random walk with restart and its applications. In: Sixth International Conference on Data Mining (ICDM'06), pp. 613–622. IEEE (2006)

30. Tooley, R.: Auto-tuning spark with Bayesian optimisation (2021)
31. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: SciPy 1.0: fundamental algorithms for scientific computing in python. Nat. Methods **17**, 261–272 (2020)
32. Whang, J.J., Gleich, D.F., Dhillon, I.S.: Overlapping community detection using seed set expansion. In: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management, pp. 2099–2108 (2013)
33. Whang, J.J., Gleich, D.F., Dhillon, I.S.: Overlapping community detection using neighborhood-inflated seed expansion. IEEE Trans. Knowl. Data Eng. **28**(5), 1272–1284 (2016)
34. Wu, F., Huberman, B.A.: Finding communities in linear time: a physics approach. Euro. Phys. J. B **38**(2), 331–338 (2004)
35. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. Knowl. Inf. Syst. **42**(1), 181–213 (2015)
36. Zareie, A., Sheikhahmadi, A.: A hierarchical approach for influential node ranking in complex social networks. Expert Syst. Appl. **93**, 200–211 (2018)
37. Zhang, T., Wu, B.: A method for local community detection by finding core nodes. In: 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, pp. 1171–1176. IEEE (2012)