Social Network Analysis and Mining (2023) 13:112
https://doi.org/10.1007/s13278-023-01115-7

ORIGINAL ARTICLE

Fast local community discovery relying on the strength of links

Mohammadmahdi Zafarmand[1] · Yashar Talebirad[1] · Eric Austin[1] · Christine Largeron[2] · Osmar R. Zaïane[1]

Received: 30 June 2023 / Revised: 12 August 2023 / Accepted: 13 August 2023 / Published online: 4 September 2023

Abstract

Community detection methods aim to find nodes connected to each other more than other nodes in a graph. As they explore the entire network, global methods suffer from severe limitations when handling large networks due to their time and space complexity. Local community detection methods are based on an egocentric function aiming to find only the community containing a query node (or set of query nodes). However, existing local methods are often sensitive to which query node(s) is used to discover a particular community. Our proposed approach, called SIWO "**S**trong **I**n, **W**eak **O**ut," is a novel community detection method, which can locally discover densely-connected communities precisely, deterministically, and quickly. Moreover, our experimental evaluation shows that the detected community is not dependent on the initial query node within a community. This method works in a one-node-expansion way based on the notions of strong and weak links in a graph. In short, SIWO starts with a community consisting only of the query node(s). Then it checks the set of nodes in the community's neighborhood in each step to add the "best" node and finally returns the desired community around the given query node. It can also be used iteratively to detect the entire partitioning of a network with or without considering overlapping communities, and concurrently identify outliers that may not belong in any community. Moreover, as it does not store the entire graph into main memory, it can also be used to find the core of a community on very large networks, while there is limited time and memory available. Finally, SIWO is also able to handle weighted graphs, making SIWO a general framework for community discovery and detection in various type of social networks.

Keywords Local community detection · Social network analysis · Community search · Link strength

1 Introduction

Over the last few years, networks have proved to be very useful to model complex systems in different domains, including social sciences, biology, pharmacology, criminology, and computer science. They allow representing relational data by a graph where the vertices (or nodes) correspond to the entities and the edges (or links) to their relationships.

As highlighted in the paper "The future is Big Graphs" (Sakr et al. 2021), the unprecedented growth in interconnected data underscores the capital role of graph processing in our society. These complex networks frequently exhibit an intrinsic structure composed of communities—i.e., groups of vertices that are densely connected within the network and sparsely connected with the rest of the network (Girvan and Newman 2002). Community detection algorithms aim to find such structures in a given network and have various applications in different fields. Most of these algorithms attempt to cluster all network vertices in a global approach that needs to store all network information inside the available memory beforehand to be able to process it. Thus, although they are assumed to cluster a network into accurate communities, they are impractical for very large networks. For example, a network with hundreds of thousands of vertices and millions of edges probably makes any global approach hopeless

✉ Christine Largeron
largeron@univ-st-etienne.fr

✉ Osmar R. Zaïane
zaiane@ualberta.ca

Mohammadmahdi Zafarmand
m.zafarmand@ualberta.ca

Yashar Talebirad
talebira@ualberta.ca

Eric Austin
eaustin@ualberta.ca

[1] Alberta Machine Intelligence Institute, University of Alberta, Edmonton, Canada

[2] Hubert Curien Laboratory, Université Jean Monnet, Saint-Etienne, France

to achieve any result in a reasonable time if the network has to be read as a whole in the first place. Moreover, in practice, the user can be more interested in the community of a given entity than in the network's whole community structure. That is notably the case for applications such as social influence analysis or recommendation systems.

To address this need, there is another family of algorithms that proceed locally. Local community discovery (a.k.a. community search) methods need a query node to start a search (De Meo et al. 2014; Luo et al. 2020). Their goal is to find all other nodes of the network belonging to the same community as the query node. Local methods have their advantages, including a targeted search, which reduces the time computation since there is no need to explore the entire graph. Thus, they enable finding communities even in extremely large networks since the time complexity does not usually depend on the network's size. Moreover, they are particularly suited for online search or handling dynamic graphs that evolve over time (Takaffoli et al. 2013).

More precisely, there are specific properties that are desired from local community detection approaches:

(1) High Efficiency: The first significant advantage of local methods over the global ones is the ability to retrieve the query node's community in a reasonable time. Since there is no need to visit all nodes of the graph during the process, the time required by a local approach should be relatively less than the time needed by a global approach (Huang et al. 2019).
(2) High Accuracy: For a query node, the retrieved community should contain the highest possible number of nodes from its true community without including outliers. As local methods usually expand the community one-node-at-a-time, it is worth mentioning that reaching this goal is much harder than for global methods.
(3) Large Graph Handling: Communities exist in both small and large networks, even those with tens of millions of nodes and edges. It is important that the method could be applied to very large networks.
(4) Online Implementation: In real-world problems, it can be necessary to identify almost instantly the queried community. Having this characteristic enables a method to find communities in real-time or a limited amount of time, and handle dynamic graphs.

Toward these goals, in this paper we introduce *SIWO*, our parameter-free method, that finds the community to which a given query node belongs. SIWO, which stands for "**S**trong **I**n, **W**eak **O**ut", firstly places the query node inside the community then expands it one-node-at-a-time, similarly to most modularity-based methods (Clauset 2005; Blondel et al. 2008). However, at each round, it selects the nodes having stronger connections to the current community. This notion of strong inward links and weak outward links was exploited in our global community mining algorithm of the same name (Gharaghooshi et al. 2020) that we extensively rework here for an efficient local community search. SIWO's performance does not depend on any preset parameter, which is a substantial advantage compared with many existing methods, including dense subgraph-based and motif-based methods. SIWO is also much faster than the current state-of-the-art methods when applied on various real-world or synthetic networks because it only loads the required parts of the network into memory and therefore uses much less memory compared to competitors, thanks to a data structure described in Sect. 4.3. Moreover, if interrupted by a time constraint before finding the whole community, the algorithm provides the intermediate set of nodes instead of an empty set for lack of time. This feature makes it a perfect choice for analyzing substantially large graphs. In addition to its preeminent task of locally detecting a query node's community, this method can also find the entire partitioning of a given graph, of overlapping or non-overlapping communities, by applying the local method iteratively on nodes randomly selected in the graph's unexplored part. Finally, it can also handle weighted networks. Various experiments that are conducted on real and synthetic networks show that SIWO outperforms the state-of-the-art local and global methods in both accuracy and robustness and confirm its abilities.

The rest of the paper is organized as follows. In Sect. 2, we introduce some related work on different families of local community detection algorithms. In Sect. 3, we define notions used in the paper and illustrate them with examples. Section 4 provides a detailed description of our method and it explains why SIWO is faster than existing triangle-based approaches. Section 5 describes two variants of SIWO designed to detect the entire partitioning of a given network (SIWO+) or to handle a weighted graph (SIWOw). The experiments and comparisons of SIWO with contenders are presented in Sect. 6 as well as experiments confirming its capability to detect the core of the community when a limited time budget is allocated such that the search must be interrupted. Section 7 illustrates the good behavior of the variants SIWO+ and SIWOw. Finally, Sect. 8 concludes this paper.

2 Related work

In general, community detection and community search have different goals: while community detection usually targets all communities of a network, the latter performs egocentric community discovery for some query vertices. Specifically, community search is aimed at finding a densely-connected subgraph that contains all query nodes. A random-walk, density, or closeness measure can be used to evaluate the qualities of the community resulting from the search. One of the most widely used measures is the minimum degree, defined as the minimum degree of all the vertices in the subgraph induced by a community (Sozio and Gionis 2010). If initially the local search problem has been solved using a global approach that needs to visit the entire input graph (Sozio and Gionis 2010), more efficient methods based on local approaches have been introduced later (Cui et al. 2014; Barbieri et al. 2015).

Community detection consists of grouping the graph vertices into subsets, considering the edge structure of the graph so that there should be many edges within each community and relatively few between the groups (Girvan and Newman 2002). One can speak of graph partitioning when the process builds a partition of the set of nodes. Still, variants of the task can also generate overlapping communities so that one node can belong to several groups or, a sequence of partitions describing the communities' hierarchical organization. Several methods have been proposed in the literature to detect the community structure of the whole network among which we can mention spectral algorithms, dynamic or diffusion-based processes such as Walktrap (Pons and Latapy 2005), Infomap (Rosvall and Bergstrom 2008), and Label propagation (Raghavan et al. 2007), function optimization-based methods including Louvain (Blondel et al. 2008), Leiden (Traag et al. 2019), and EdMot (Li et al. 2019) that exploit the well-known modularity, generative models using Bayesian inference, stochastic block modeling or deep neural networks and embedding techniques (Su et al. 2022). We do not detail them here since it is not the main topic of the paper and, refer the interested reader to (Fortunato 2010; Fortunato and Hric 2016; Dao et al. 2020; Souravlas et al. 2021; Su et al. 2022).

However, there is also another type of methods that functions locally. They need a query node and aim to identify all the other nodes of the network belonging to its community. Thus, by following this local approach, community detection joins community search. Nevertheless, as Baltsou et al. pointed out, "Local community detection (LCD) is used in the literature for two similar problems. On the one hand, it refers to finding the community to which a seed node (or group of seed nodes) belongs. On the other hand, it refers to a method that uses local information to discover all communities in the network" (Dilmaghani et al. 2021; Baltsou et al. 2022). Thus it is important to note that this paper is devoted to the first problem for which, SIWO, the method we introduce, has been designed, even if one of its variants, SIWO+ makes it possible to treat the second problem (see Sect. 5).

Many algorithms have been proposed to search for a high-quality community around a query vertex, but, to our knowledge, there is only one survey related to local community detection which proposes a typology of the techniques in terms of network type (static, dynamic, etc) and techniques (greedy or non-greedy) (Baltsou et al. 2022). In this paper, we consider only static networks and categorize the methods into three families. Methods in the first family are based on various cohesiveness metrics which evaluate the quality of the community and return a dense subgraph which can be a K-core (Fang et al. 2020), a K-truss (Huang et al. 2014), or a K-clique (Palla et al. 2005) to cite a few. In theory, any of these dense subgraphs can be used to model the searched community. However, finding such cohesive subgraphs in a given network is an NP-hard problem, making this family's methods unsuitable for real-time query processing (Huang et al. 2019). To overcome this, methods that use heuristics (Sozio and Gionis 2010) or quasi-subgraphs (Brunato et al. 2008) have been introduced and compared by Fang et al. (2019). Other experiments and theoretical analyses, done by Huang et al. (2019), show using k-clique models (Palla et al. 2005; Cui et al. 2013) leads to the most cohesive structures and that the methods based on K-core (Cui et al. 2014; Barbieri et al. 2015) seem to be the most efficient and suitable for real-time query processes. However, they do not guarantee connectedness in communities, and consequently, they lack cohesiveness. Finally, the K-truss based models (Huang et al. 2014, 2015; Akbas and Zhao 2017) achieve a balance between quality and efficiency on moderate-to-large graphs, making it the best choice for the community detection purpose among all other cohesive-based approaches, including LCTC (Huang et al. 2015) which has been shown to work on large networks. However, in real networks, communities are rarely likely to be perfect cliques or even quasi-cliques, limiting the use of this approach in certain cases.

Another family of local community detection methods functions based on motifs, patterns of interconnections occurring in real networks in higher numbers than in randomized networks (Milo 2002). In this family, MAPPR (Yin et al. 2017) generalizes APPR (Andersen et al. 2006). It seeks clusters of nodes based on higher-order network structures, with minimal motif conductance. This measure has been retained because it has been used with success as a clustering criterion (Schaeffer 2007) notably in that type of community detection methods (Yang and Leskovec 2013). Even though MAPPR introduces cliques of sizes larger than 3 as motifs for local community detection, various experiments show motifs larger than 3 cannot accurately capture the community (Slater et al. 2014). Moreover, in some cases, edges (i.e., cliques of size 2) are more effective, and APPR, which uses edges, performs better than MAPPR. But, both methods have some parameters, especially the tolerance parameter ϵ, and the teleportation parameter for the random walk α, that need to be precisely tuned for accurate community detection.

The third family of local approaches attempts to maximize a quality function, by initially placing the query node in the community and then expanding it. Usually, the quality function compares the intensity of the relationships inside the community and outside it. Modularity R (Clauset 2005) and modularity M (Luo et al. 2006) are two well-known methods in this category, but one can also mention Bagrow and Bollt (2005) or Chen et al. (2010). Recently, Luo et al. (2018) proposed two methods, DMF_R and DMF_M, which are claimed to outperform Modularity R and M. However, the published results are not reproducible, and we cannot use them in comparative experiments due to the lack of publicly available code. Metric T (Fagnan et al. 2014) is another modularity-based method that improves on R and M in terms of accuracy but it is very time-consuming, as it needs to re-count the number of triangles to which each newly added node belongs in each round. Finally, there is also MWC (Bian et al. 2017) which employs multiple walkers to explore the network for local cluster identification.

According to Baltsou et al. (2022), Hamann et al.'s Triangle-Based Community Expansion (TCE) method (Hamann et al. 2017) is the best method focusing on node selection. TCE is fundamentally based on the Local Tightness Expansion (LTE) algorithm (Huang et al. 2011), as both exploit the fact that some edges are more embedded in their neighborhood and have more common neighbors than others. LTE uses an edge similarity score based on triangles for deciding which node to add next and for determining the quality of the community. TCE, on the other hand, also uses an edge score based on triangles, but employs conductance for the quality function of the community. Baltsou et al. (2022) highlight that experimental evaluations show that TCE exhibits solid performance and often finds the correct community, building on the already excellent performance of the computationally more expensive LTE on most tested graphs. However, a significant drawback of both LTE and TCE is that they use NetworKit[1] for their implementation, requiring the entire graph to be loaded into the main memory, making these algorithms infeasible for handling large networks.

In this paper, we propose *SIWO*, a method that also belongs to the third family, as it aims to optimize a quality function. However, unlike the state-of-the-art methods, its quality function is not an extended version of modularity, which is known for its resolution and field of view limits (Lancichinetti and Fortunato 2011). Instead, SIWO directly exploits the notion of edge strengths, a well-known concept in the literature introduced by Granovetter (1983), to capture the neighborhood's density around other nodes, and evaluates an edge's strength using the number of triangles shared by its endpoints. SIWO starts by placing the query node inside an empty community and then expands this community one node at a time by selecting nodes with stronger connections to the current community. The algorithm produces very accurate deterministic results and has the advantage of being parameter-free. Moreover, SIWO is faster than the current state-of-the-art methods, as it does not need to load the whole graph in main memory, unlike other methods. This performance advantage is confirmed by our experiments and discussed further in Sect. 6.

[1] https://networkit.github.io/.

3 Preliminaries

Before presenting our method, we introduce the notations used throughout this paper. We consider an undirected and unweighted graph $G = (V, E)$, where V is the set of vertices ($|V| = n$) and E, the set of edges ($|E| = m$). Without loss of generality, we assume that G is connected, which necessitates there is a path connecting any node $u \in V$ to any other node $v \in V$. As we need to find a community developed of a connected group of nodes, we can argue this premise does not harmfully affect the final detected community.

We assign a *strength value* to all edges inside the graph, representing that edge's tendency to be inside a community rather than between communities. The higher the strength, the higher its tendency to be an inner link. Many previous works, especially methods that attempt to optimize a local modularity criterion, use only the presence of edges to determine the best next node that must be merged to the community and they ignore other kinds of structural property of the network. We consider that exploiting larger patterns namely triangles (i.e., triplets of linked nodes) can help to detect a more cohesive community. There are two advantages in using triangles to determine the strength of an edge:

(1) We search communities that are as cohesive as possible inside a given graph. With this idea in mind, cliques can be considered a reasonable choice to look for, as they are the most cohesive structure that several nodes can construct. Thus we focus on subgraphs consisting of nodes that participate in forming cliques of size 3, or triangles, generally denser than subgraphs made by nodes connected only via edges (such nodes belong to cliques of size 2).
(2) We need to look for patterns that repeat more often in a given graph. Increasing the size of a clique leads to more cohesive patterns, but these are significantly rarer. Various observations on real and synthetic networks show triangles are generally the most reoccurring subgraph compared to any other size cliques.

To explain how cliques of size three are used to compute the strength of an edge, we introduce the following notions.

Definition 1 (*Support*) Given a graph $G = (V, E)$, the support of the edge $e_{u,v}$ is the number of mutual neighbors of u

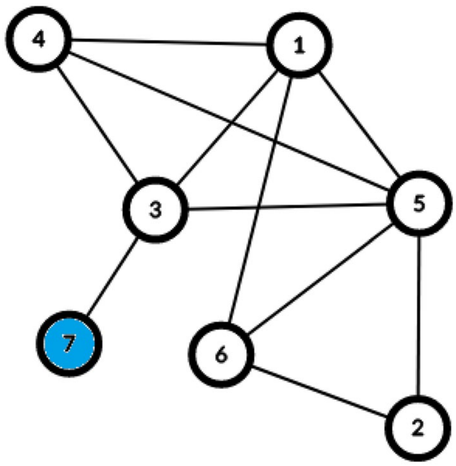

Fig. 1 Peripheral node (7 in blue) (Color figure online)

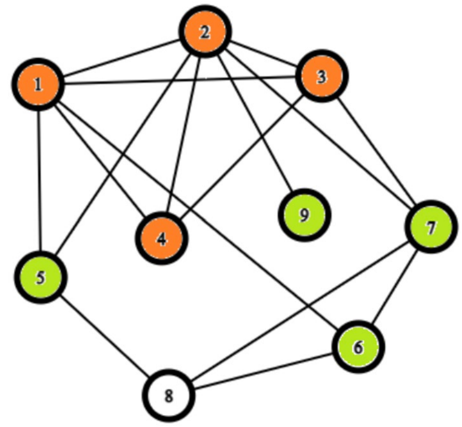

Fig. 2 Community ($\{1, 2, 3, 4\}$ in orange) and its shell ($\{5, 6, 7, 9\}$ in green) (Color figure online)

and v or the number of triangles that $e_{u,v}$ belongs to and it is defined as follows:

$$\sup(u, v) = |\{w \in V, e_{u,w}, e_{v,w} \in E\}| \tag{1}$$

Example 1 Considering Fig. 1, $\sup(e_{1,5}) = 3$ as there are exactly three triangles $\Delta_{1,3,5}$, $\Delta_{1,4,5}$, and $\Delta_{1,5,6}$ that own the edge $e_{1,5}$.

Then, during the community detection process, the neighborhood of the detected community is defined as the **shell set**.

Definition 2 (*Shell set*) Given a graph $G = (V, E)$, the shell of the community C, denoted by shell(C) is defined by:

$$\text{shell}(C) = \{v \in V, v \notin C \text{ s.t. } \exists u \in C, e_{u,v} \in E\} \tag{2}$$

The shell of the community C is the group of nodes that are not in C, but that are connected to at least one node belonging to that community C.

Example 2 Consider Fig. 2. If $C = \{v_1, v_2, v_3, v_4\}$ (orange nodes), then shell(C) contains the green nodes directly connected to one node (nodes v_6, v_9) or multiple nodes (nodes v_5, v_7) in C : shell$(C) = \{v_5, v_6, v_7, v_9\}$.

Definition 3 (*Peripheral node*) Given a graph $G = (V, E)$, a peripheral node is a node with degree one. We formally define the set of all peripheral nodes in G as follows:

$$\text{peripheral-nodes}(G) = \{v \in V \text{ s.t. } |V_N(v)| = 1\} \tag{3}$$

where $V_N(v)$ corresponds to the set of neighbors of node v.

Example 3 Considering Fig. 1, node v_7, depicted with blue color, is a peripheral node, as it is connected only to node v_3.

Finally, we can define the notion of link's strength.

Definition 4 (*Link's strength*) Given a graph $G = (V, E)$ and a vertex $u \in V$, we define the strength of the link connecting u to a node $v \in V_N(u)$, where $V_N(u)$ is the set of neighbors of u, as follows:

$$s_{u,v} = \frac{\sup(u, v)}{\sup_{u,\max}} + \frac{\sup(u, v)}{\sup_{v,\max}} - 1 \tag{4}$$

where $\sup_{u,\max} = \max_{w \in V_N(u)}\{\sup(u, w)\}$ is the maximum support of u and any node in $V_N(u)$. In this formula, since the numerator in each fraction is always less than (or equal to) the denominator, each fraction is always in the range of [0, 1]. By subtracting 1 from the sum of the fractions, the link's strength takes a value between $[-1, +1]$ which gives us a better way to compare the strengths and handle them in an algorithm. However, it can also be normalized to have merely positive values between $[0, +1]$ as follows:

$$s_{u,v} = \frac{\sup(u, v)}{2}\left(\frac{1}{\sup_{u,\max}} + \frac{1}{\sup_{v,\max}}\right) \tag{5}$$

In both Eqs. 4 and 5, if $\sup_{u,\max} = 0$ or $\sup_{v,\max} = 0$, then $\sup(u, v) = 0$ and we assume that part of the equation is zero. Both equations indicate the higher the number of mutual neighbors of a pair of connected nodes, the higher the strength of the link that connects them. These values are computed only for the edges connecting nodes in the shell to the community and exploited by our algorithm to find the best candidate to expand the community locally. Still, this best candidate is added to the community only if it increases the total strength of the edges inside the community. The choice of using $\sup_{u,\max}$ as the denominator in Eqs. 4 and 5 is rooted in our intention to normalize the strength values. By doing so, we aim to provide a relative measure of strength that is consistent across the graph, irrespective of the degree of the nodes. This ensures that the strength value reflects the relative importance of the connection in the context of the node's other connections. Specifically, for a node with a high degree, even if it has a large number of mutual neighbors with another node, the strength value might not be as significant if it has even stronger connections with other nodes. Conversely, for a node with a smaller degree, a few mutual neighbors can be quite significant. This approach allows us to capture the community structure in the graph more effectively, ensuring that the strength value is not just an absolute measure but a relative one that takes into account the broader connectivity patterns of the nodes involved.

4 Proposed approach

This section is dedicated to the proposed approach. First it describes the local community discovery method SIWO, then it explains how SIWO can also be applied in two different settings: with multiple query nodes or with a limited time budget. Finally, it details how the optimizations allow it to handle large networks.

4.1 Local strong and weak link

Our novel local community search method SIWO starts with placing the query node(s) inside an empty community. Then it explores locally to find the best node among the set of nodes in the community's neighborhood in each step to expand it by one node at a time and ultimately return the desired community around this given query node. Our method iteratively performs four steps to construct the community and one last step to adjust it, as stated in Algorithm 1. Its different steps are as follows:

(1) *Update shell set*: Initially, as there is no community, there is no shell either. After placing the query node u inside the community C, all other nodes connected to u go into the shell S. If the query node is a peripheral node, its single neighbor replaces the query node, as a peripheral node cannot usually generate a justifiable community. At the next rounds, we must only update S by removing from it node v, which joined C in the previous round, and adding to the shell the nodes directly connected to v that do not already belong to C. The updated shell denoted S' is determined as follows:

$$S' = (S \cup (V_N(v) - C)) - \{v\} \tag{6}$$

where $V_N(v)$ corresponds to the set of neighbors of node v. The shell set S not only corresponds to the neighborhood of the community C but it also contains all candidates that may join the community in forthcoming steps. It is worth mentioning that by adding nodes only from S to C at each round, we guarantee the final detected community is a connected subgraph.

(2) *Assign strength values*: We identify a node from S with the strongest connections to the community C at each round and claim it is the best node among all candidates, potentially joining C at the end of the current cycle. To find the best candidate, we need to assign strength according to Eq. 4 to each edge that connects any node in S to any node in C. We do this process locally, which means we do not need to access the whole network. We need only to compute the strength value $s_{u,v}$ for each pair of nodes (u, v) where $u \in C$ and $v \in S$.

(3) *Select best candidate node*: After the edge strengths are determined, we compute the strength of each potential community C'_i which is composed of every node in C, plus the ith node from the shell set. The strength of C'_i is defined as the sum of the strength values of all edges inside that community. It is computed by:

$$s(C'_i) = \sum_{u,v \in C} s_{u,v} + \sum_{u \in C} s_{u,v_i} \tag{7}$$

Where v_i is the ith node in the shell. After calculating all $s(C'_i)$s, we find the largest one and declare the corresponding node v_j to be the best candidate only if $s(C'_j) > s(C)$ then continue to the next step. Otherwise, the community C will no longer expand and the algorithm goes to the reformation process (Step 5). The last condition ensures a node joins C only if it improves its strength.

(4) *Expand community*: Now that SIWO found the node which improves the community's strength the most, it has to expand the community C by including that node to C.

$$C' = \begin{cases} C \cup \{v_j\} & \text{if} \quad s(C'_j) > s(C) \\ C & \text{otherwise} \end{cases} \tag{8}$$

If the community expands, the algorithm continues by returning to step (1) with the newly developed community $C = C'$ and S ready to be updated, with the process repeating. If the community remains the same, the algorithm refines the last obtained community.

(5) *Reform community*: Finally, any peripheral node connected to a node of C is added to the community. Indeed, a peripheral node cannot join any community without this last step, as the sole edge that connects it

to the rest of the graph cannot belong to any triangle and thus, such an edge cannot improve the community's total strength. Although appending peripheral nodes in this way is intuitively acceptable because they can only be a part of their sole neighbor's community, in some applications, such nodes are considered to be outliers and stay out of the community. So, adjusting the community is done based on user preference.

merged with Step 1 (Update Shell Set). Indeed, instead of letting a peripheral node lurk in the shell set for the whole process till the end, it can de facto be added to the community once it is identified in the Shell set. Similarly, in Step 3 (Select Best Candidate Node) if there is more than just one best node to add, instead of adding one and leaving the others for the next iterations, we could add all the top nodes

Algorithm 1 *SIWO*: A local community search algorithm

Input: Graph G and query node(s) $\{q\}$
Output: The community C of the query node(s) $\{q\}$

```
▷ Initialization
C = {q}, S = V_N(q)
while S ≠ ∅ do
    ▷ Assign Strength Values
    For all v ∈ S, calculate the strength of s(C ∪ {v}) according to Equation 7
    ▷ Select best candidate node
    Find the node u in S which maximizes s(C ∪ {v}) for all v ∈ S
    if s(C ∪ {u}) ≥ s(C) then
        ▷ Expand Community
        C = C ∪ {u}
        S = S ∪ (V_N(u) − C)) − {u}
    end
    else
        break
    end
end

▷ Reform Community
C = C ∪ P where P is the set of peripheral nodes that are connected to a node
in C
return C
```

In SIWO, the stopping condition is crafted to balance precision and efficiency. Its primary objective is to ensure that the detected community is cohesive, reflecting the local structure around the query node. At the same time, it aims to preserve the locality of the search, preventing unnecessary expansion into distant regions of the graph. When using Eq. 4, the stopping condition is inherently tied to the link strength values, which span from negative to positive. While this provides a natural halting point in smaller datasets, in larger networks it can lead to the inclusion of a vast number of nodes, often beyond the genuine community boundaries. Similarly, when using Eq. 5, the link strengths are non-negative, the challenge of over-expansion persists. This risk is alleviated by introducing a threshold as a heuristic stopping condition. This threshold (currently set to 1) ensures that the community only expands when nodes of significant connection strengths are considered, preventing over-expansion into less relevant regions of the graph.

The choice of this threshold, while empirical, was based on preliminary observations across diverse datasets. It serves as a balance between precision and recall, ensuring that the detected community remains cohesive and accurately represents the local structure around the query node. As seen in our experiments in Sect. 6.1, datasets with well-defined, tightly-knit communities often resonate better with Eq. 4, while those with more intricate, overlapping community structures may benefit from Eq. 5. While further empirical analysis could provide more efficient threshold values for different datasets, the current study offers a foundational approach, with deeper explorations earmarked for future work.

Note that to further improve the efficiency of the algorithm, Step 5 (Reform Community) could be eliminated and that have equal strength contribution in one round avoiding additional iterations.

4.2 Community discovery with a set of query nodes

SIWO is able to start with more than one query node. In such a case, SIWO starts by placing all query nodes in C, and S will contain the neighbors of all query nodes which are not themselves query nodes. Although using an appropriate set of query nodes results in a connected community as expected, querying multiple unrelated nodes of course does not guarantee this connectivity.

4.3 Community discovery with a limited time budget and limited memory

Most search methods require significant time to find a community in dense networks, and do not produce results until the process is finished. In contrast, SIWO can produce a result even if it is interrupted. Given a limited time budget, it generates the core part of the community. When the allotted time is less than the required time to find the whole community, SIWO discovers and returns the part of the community composed of the nodes with the strongest connection to the query node's community found up to the interruption time. Memory is a constraint for algorithms that must load the entire very large network. This can be a significant bottleneck for processing massive networks. However, SIWO is designed to handle very large networks with limited memory resources, making it suitable for a wide range of applications. The way SIWO achieves this is explained in detail in Sect. 4.4, where the optimization techniques and data structures used are outlined.

4.4 Optimizing SIWO

4.4.1 Efficient memory management

In order to efficiently process large networks while conserving memory, SIWO employs a unique approach that involves selectively reading and storing the necessary information from the network file. This is achieved by utilizing a specialized data structure to store relevant information about the required part of the graph, thus minimizing memory usage while also making file access faster. This data structure consists of four components: A list of line break locations up until the maximum node number that was requested from the file, a dictionary of neighbors for each required node, a dictionary of required edge strengths, and a dictionary of required node pair support values.

To optimize memory usage, the graph is first pre-processed using a Map-Reduce program that converts the file to an adjacency list format, with the rule that "line number n consists of all the nodes that are adjacent to node n." During the execution of SIWO, each time the neighbors of a node are requested, the data structure is checked to see if the neighborhood information for that node already exists. If not, the algorithm uses the line break locations to quickly reach the required line and stores the unseen line break locations in the process. Similarly, whenever the calculation of the support of a pair of nodes or the strength of an edge is requested, the corresponding dictionaries are first checked for the required information. If not found, the data structure is updated accordingly after the necessary calculations.

This optimization process not only reduces memory usage but also speeds up the algorithm's execution time, since it avoids repeatedly reading and processing the entire network file, eliminating unnecessary I/O operations. By storing only the necessary information in memory and accessing it efficiently, SIWO can perform its computations more quickly and with lower resource requirements, making it a practical solution for analyzing large-scale networks.

4.4.2 Efficient candidate node selection

The improvement a candidate node can bring to a community's quality can change as the community evolves since the edges connecting this candidate to the community may also alter. An advantage of SIWO, compared to the local modularity-based methods that exploit the support of the edges, such as Fagnan et al. (2014), relies upon its second step. Indeed, the time-consuming part of SIWO, which calculates edge strengths, is independent of the community's current state. Because of this advantage, our method does not require repeating the second step for every node in the shell of the community in each round. More precisely, local strength values are assigned to the edges connected to a candidate node only once during the entire process. Edge strength is agnostic to the current state of the community C. It is the strength that a given node adds to the community C that changes. This reduces the required time for the whole process by a considerable amount.

To make the third step of SIWO more efficient, we noticed that all the edges in the community C will be in all possible next-round communities C_i'. Thus we only need to consider the sum of strengths of the edges that connect the ith node from the shell set S to C when determining which node increases the total community strength by the greatest amount, rather than calculating the total strength of each C_i'. We also note that all edges connecting nodes in $S - \{v_i\}$ to nodes in C_i' will be the same as edges connecting $S - \{v_i\}$ to C plus the edges connecting $S - \{v_i\}$ to v_i. Therefore we can update the sum of edge strengths connecting a node in S to C rather than recalculating it at each iteration. We only perform a full calculation on the first iteration and store these sums for each node in S. When adding a node to the shell set in Step 1 we initialize its sum to 0. For each subsequent iteration, we only need to consider the neighbors of the previously added best node v_i, $V_N(v_i)$. For each $v_j \in V_N(v_i)$ if $v_j \in S$ then we add s_{v_i,v_j} to the strength total for shell node v_j. If $v_j \in C$ then all of its edges with nodes in S have already been added to their strength totals and no updating is needed. The best candidate node is then simply the node in S with the greatest strength sum. The problem of calculating the sum of strengths for all edges in every possible C'_i is reduced to a small number of addition operations and a linear scan through S for the highest sum.

Thus, one significant privilege of SIWO is that the total time required to return the desired community depends on the community's size, not the given network. Indeed, the algorithm's first step can be done in $O(c.d)$, where c is the number of nodes in C, and d is the nodes' average degree, and it would be almost linear for large sparse networks. The second step can be done in $O(c.d^2)$ because we need to investigate each neighbor's neighbors for all nodes in the community. In the third step, only the edges between the last node added to C and its neighbors need to be considered to update the running totals, which is $O(d)$, and finding the maximum strength sum of S is linear in $|S|$, which is $O(c.d)$, and so the whole process takes $O(c.d + d) = O(c.d)$ time. We can fulfill the fourth step in a constant time as we only need to add one new node to the community's current state. The reformation step takes $O(c.d)$ time with proper implementation as we only need to see which neighbors of the nodes in C are peripheral nodes.

5 SIWO variants: SIWO+ and SIWOw

5.1 SIWO+: Global community detection

In addition to finding the community of a given query node, our approach can also detect all of the communities inside the network using only local information by applying SIWO multiple times, each time with a different starting node belonging to uncharted parts of the graph. It is important to note that our intention with SIWO+ is not to compete for efficiency with other global community mining algorithms; rather, we aim to demonstrate that SIWO, designed primarily for community search, can also be effectively employed for global community mining because of its flexibility.

After SIWO discovers the first community out of a random node in G, it initiates another search by picking a new random node from the network's unexplored part. By doing this, the algorithm iteratively finds communities inside the network until all communities are detected, meaning any node of G resides in a community. This variant of our method is called SIWO+.

In most cases, we are interested in having each node inside only one community. However, this is not always the case, and overlapping communities may be of interest for some applications. Thus, to have an implementation compatible with both scenarios, we consider a set I for nodes that need to be ignored during the detection process. These are the nodes already in a community, so the algorithm will not consider them among the candidates for the next communities' expansion during the subsequent discovery processes. When a task demands overlapping communities, meaning a node can be part of more than one community, we may clear set I before initializing any new community search to avoid ignoring nodes already placed in a detected community. Algorithm 2 reveals how our community detection mechanism works in either scenario where parameter ol is true when overlapping communities are sought for.

Algorithm 2 Global Community Detection with SIWO+

Input: Network G, Boolean Overlap Parameter ol (true means with overlap)
Output: Set $P = \{C_1, C_2, ..., C_k\}$ of all communities of G

```
V_P ← {}                      ▷ set of processed nodes
I ← {}                        ▷ set of nodes to be ignored in next discovery
P ← {}                        ▷ final partitioning of the network

while Size(V_P) < |G| do
    u ← a random node from {G.nodes − V_P}
    C ← SIWO(G − I, u)
    P.append(C)
    V_P ← V_P ∪ C

    if ol = False then
        I ← I ∪ C
end
return P
```

While SIWO+ offers the flexibility to detect overlapping communities, it could be susceptible to excessive overlaps, where nodes might be part of multiple communities, if indeed the real data comprises all these overlaps. To manage the degree of overlap a user might want to limit, we could introduce a threshold parameter, θ, which limits the number of communities a node can belong to. After a node is part of θ communities, it is added to the ignore set I, even if the overlap parameter *ol* is set to true. This ensures that while communities can overlap, the extent of overlap is controlled, preventing nodes from being part of an unrestrained number of communities. Scoring criteria can also be introduced to keep only the relevant overlaps; however, this is beyond the scope of our targets for this paper on community search.

5.2 SIWOw: Community detection on weighted networks

The topological structure of a network provides a great deal of information for community search. However, many networks are defined both by the presence of connections and the intensity of those connections. These edge weights can be crucial to understanding the network (Barrat et al. 2004). We therefore extend our approach to deal with weighted networks and call this variant SIWOw.

The extension to handle weights is straightforward. The only necessary modification is to the support calculation of Eq. 1. In the unweighted case, $sup(u, v)$ is the count of common neighbors for nodes u and v, i.e. the number of triangles to which $e_{u,v}$ belongs. In the weighted case, we must assign a value to each triangle based on the edge weights. Several approaches have been tried for the similar problem of defining a weighted clustering coefficient (Saramäki et al. 2007). We adapt the approach of Onnela et al. (2005) who use the geometric mean of the three edge weights of the triangle. This matches our intuition that all three edges contribute to the formation of a triangle. We also experiment with the arithmetic mean, harmonic mean, and minimum to evaluate how sensitivity to large values impacts the community detection. The weighted support of $e_{u,v}$ is defined as:

$$\mathrm{sup}(u, v) = \sum_{w \in V_{CN}} \mathrm{func}(\mathrm{weight}_{u,v}, \mathrm{weight}_{u,w}, \mathrm{weight}_{v,w}) \tag{9}$$

where V_{CN} is the set of common neighbors of nodes u and v, func is one of arithmetic mean, geometric mean, harmonic mean, or minimum, and $\mathrm{weight}_{u,v}$ is the weight of the edge between nodes u and v. In the case of a weighted graph with all edge weights equal to 1, Formulas 1 and 9 will return the same support value and the results of the algorithm will be identical, which is what we desire. All other parts of the algorithm remain the same as in the unweighted case. The strength calculations of Formulas 4 and 5 still result in values in ranges $[-1, +1]$ and $[0, +1]$, respectively, regardless of the magnitude of the edge weights and support values.

Building on the idea of adapting a greedy community search approach for weighted networks, one might wonder why we did not simply adopt the direct edge weights as link strengths. However, note that direct edge weights may not always reflect the underlying community structure, particularly in networks with a broad range of edge weight values. A high edge weight between two nodes might indicate a strong pairwise interaction without suggesting shared community membership. By employing the support calculation from Eq. 9, we integrate information from a node's broader neighborhood, offering more accurate information on community affiliation as compared to only using edge weights. This approach ensures that the strength of a link is not just determined by its weight but also by the surrounding topological and weighted structure. Furthermore, our method ensures consistency with the unweighted version of the algorithm, allowing for a unified approach to community detection in both weighted and unweighted graphs.

6 Evaluation of SIWO

We first evaluate the performance of SIWO, designed for local community discovery, on real-world networks, with or without ground-truth, and compare it against the best methods of the state of the art in Sect. 6.1. Then, in Sect. 6.2, we use synthetic networks to study the behavior of SIWO and its competitors when the community structure is more or less well-defined. Finally in Sect. 6.3, we study the case where a limited time budget is allocated to the discovery of the community. All experiments are conducted on a commodity laptop with 16 GB of memory. Note that the contenders that were selected for our experiments are chosen because of their availability and the ease of their implementation in the case of non-availability.

6.1 Discovering local communities in real-world networks

Data sets: We use only the first six real-world networks in Table 1 since the available communities for Youtube, Orkut, and Friendster have been functionally defined (Yang and Leskovec 2013) which does not constitute a confident ground truth (Rabbany and Zaiane 2015; Peel et al. 2017). An exception is made for DBLP and Amazon with which, knowing the researchers and the books, a sanity check on a sample can be done. The larger datasets are used later in Sect. 6.3. Table 1 shows the number of nodes (n), edges (m), and true communities (no. C) in each network and, the right-most column indicates if a network has overlapping

Table 1 Characteristics of the real-world networks

Networks	n	m	No. C	Overlap
Karate (Zachary 1977)	34	78	2	No
Dolphins (Lusseau 2003)	62	159	2	No
Pol Books (Adamic and Glance 2005)	105	441	3	No
Football (Girvan and Newman 2002)	115	613	12	No
DBLP (Yang and Leskovec 2013)	317,080	1,049,866	13,477	Yes
Amazon (Yang and Leskovec 2013)	334,863	925,872	75,149	Yes
Youtube (Yang and Leskovec 2013)	1,134,890	2,987,624	8,385	Yes
Orkut (Yang and Leskovec 2013)	3,072,441	117,185,083	6,288,363	Yes
UK-2002[a]	18,520,486	298,113,762	N/A	N/A
Twitter (Kwak et al. 2010)	41,652,230	1,468,364,884	N/A	N/A
Friendster (Yang and Leskovec 2013)	65,608,366	1,806,067,135	1,449,666	Yes

[a]https://law.di.unimi.it/webdata/uk-2002/

communities. The first four networks, Karate, Dolphins, Political Books, and Football are smaller and have disjoint communities whereas Amazon and DBLP are initially composed of overlapping communities.

Evaluation protocol: In these experiments, we consider every different node successively as the query node and obtain its community, which is then compared with its true community using precision, recall, and F_1 scores. We report the results corresponding to the average scores of all the query nodes (with standard deviation for F_1). The best score for each data set is indicated in bold font.

Concerning the datasets with overlap (DBLP and Amazon), we adopt the methodology used in the literature (Yin et al. 2017) to adjust the ground-truth for these datasets by merging all communities that a query node belongs to into a new subset and use it as the ground-truth community that is expected to be discovered. We then ignore the obtained subsets having less than 10 nodes (as they probably correspond to noise). Then, we limit our selection of query nodes to the nodes whose adjusted ground-truth community's size does not exceed 100 for both Amazon and DBLP. Following this, the average sizes of communities under evaluation become 39.17 and 25.44 respectively.

Baselines and settings: We compare our method SIWO to R (Clauset 2005), M (Luo et al. 2006), K-Truss (Huang et al. 2014), APPR (Andersen et al. 2006), MAPPR (Yin et al. 2017), MWC (Bian et al. 2017), TCE (Hamann et al. 2017), LTE (Huang et al. 2011) and LCTC (Huang et al. 2015). We do not compare against methods such as Akbas and Zhao (2017) or Fang et al. (2020) as they either require pre-processed information (which makes the comparison unfair) or due to lack of access to functioning executable code. To avoid implementation bias, we use publicly available codes for LCTC and K-Truss,[2] APPR and MAPPR,[3] TCE and LTE,[4] and MWC.[5] However, we implemented SIWO[6] as well as Modularity R and Modularity M in Python.

We run the experiments with different parameters for each method and report the best results. We use different values of K between 3 and 5 for K-Truss, resulting in the best accuracy for different networks. APPR and MAPPR have been executed with $\alpha = 0.98$ and $\epsilon = 0.001$ and MWC has been used with $\alpha = 0.01$, $K = 5$, and $\theta = 0.9$. As these methods are sensitive to ϵ and α, respectively, the authors generally advised using small values to reach a higher F_1 score. Since MWC could not process some nodes in Amazon and DBLP, and Modularity M may return null communities, we retain only the nodes for which all the methods can discover a community.

Experimental results: As shown in Table 2, in 5 out of 6 experiments on these real networks, at least one row dedicated to SIWO achieves the highest F_1 score, although it falls behind TCE only slightly in the case of the Football network. In the case of DBLP, both SIWO and MWC obtained the best F_1 score, while SIWO has significantly higher precision. Concerning the different ways to compute the strength of the edges, we observe that Eq. 4 results in better F_1 score for the Karate network, while Eq. 5 leads to a better score for other networks. For experiments involving Eq. 4, we employed a timeout of 0.01 s. This timeout acts as a secondary stopping condition, ensuring that the algorithm remains computationally efficient without compromising the quality of the detected communities. In smaller graphs, this timeout is often inconsequential due to the rapid computation.

[2] Code available via request to authors.

[3] https://snap.stanford.edu/mappr/code.html.

[4] https://github.com/kit-algo/LCD-cliques-experiments.

[5] https://sites.psu.edu/yuchenbian/files/2019/08/MWC_release.zip.

[6] https://github.com/talebirad/SIWO

However, in larger networks, it serves as a practical constraint, ensuring that SIWO remains both local and efficient. For experiments involving Eq. 5, no such timeout is needed.

Running time: SIWO is also faster than other methods. It finds the communities of queried nodes of Amazon in less than 100 ms on average, while the average times for K-Truss, APPR, MAPPR, MWC, LTE, TCE, and LCTC are 6.76, 2.68, 5.13, 0.57, 35.30, 11.84, and 4.87 s, respectively.

The Orkut dataset (Yang and Leskovec 2013) contains more than 100 million edges and presents a challenge for finding an adequate group of nodes forming a community for a query node. Our experiments demonstrated that none of the existing methods mentioned above could, in a reasonable time, find communities for a set of random query nodes in Orkut, except LCTC, but LCTC is significantly slow as shown in Table 5. In contrast, SIWO is capable of discovering the community, or at least the core of the community (consisting of nodes with the strongest connections to the community) in a matter of minutes on a commodity laptop. Given a limited time budget, the process can halt and provide the nodes of the community with the strongest links to the query node. Other methods return an empty set if halted. In the next experiment, we further analyze and compare the methods in terms of speed.

6.2 Discovering local communities in synthetic networks

We compare SIWO with a number of different algorithms on synthetic LFR benchmark networks (Lancichinetti et al. 2008) to analyze their behavior when the community structure is more or less well-defined.

Synthetic data sets: We study the performances of the methods in function of μ, the fraction of inter-community edges incident to each node. μ increases from 0.10 to 0.45 to deteriorate the community quality in the networks so that it is more difficult to find the true communities. We do not consider values for μ higher than 0.45, which means nodes having more edges out of the community rather than inside and, consequently, contradicts the general definition of a community. The number of nodes n is set to 10,000, the average degree is 20, the communities' sizes range from 15 to 60, the number of communities is between 500 and 540, the power law exponent for the degree distribution (τ_1) is set to 8, and the power law exponent for the community size distribution (τ_2) is set to 5.

Baselines and settings: We do not consider K-Truss, TCE, LTE, and LCTC in this experiment due to their lack of efficiency, particularly for large datasets. MWC is very sensitive to its α parameter such that a large value stops the code and prevents it from finding communities. This issue occurs more frequently as the density or the mixing parameter of the network increase. To avoid such problems, the parameters must be carefully selected. The parameters used in Sect. 6.1 for MWC lead to proper functioning. We use the same parameters as before for APPR and MAPPR, which lead to their best performance. We use every node of these networks as query nodes separately and calculate the average F_1 score over all of them.

Experimental results: Fig. 3 shows similar F_1 score for MAPPR and SIWO which perform better than other methods. All methods' efficiency drops more or less as μ increases. The performance of MAPPR strongly depends on carefully setting up its parameters, whereas SIWO is parameter-free. Although APPR performs accurately when a community's quality is high, it cannot maintain such accuracy for higher values of μ. Similarly, modularity R's performance falls significantly with the increase of μ. Still, Modularity M and MWC seem to be able to discover communities of these synthetic networks with high accuracy; however, they could not reach the level of MAPPR and SIWO.

Running time of SIWO: Table 3 shows the average required time for each method to retrieve the community of a queried node as a function of the mixing parameter μ. Even though SIWO is implemented in Python while MWC, APPR, and MAPPR are all implemented in C++, we observe that SIWO needs significantly less time on average to discover communities, owing to the optimizations mentioned in Sect. 4.4. It is worth noting that no algorithm needs pre-computed information, and they all find the communities on the fly using local properties. As μ increases, the number of interconnections between nodes in different communities raises, leading to more complications for most methods attempting to find a queried node's community, which results in more time.

6.3 Community discovery with a limited time budget and limited memory

One key feature of SIWO is that it can return intermediate results before finishing a search. In other words, SIWO can output the discovered community *up to the allotted time*, which is basically a subset of the targeted community. To compute the quality of the intermediate results by comparing it to the corresponding ground truth using precision, recall, and F_1 Score, we have conducted an experiment on a large synthetic network generated with LFR benchmark (Lancichinetti et al. 2008). The network is generated by setting the number of nodes to 1 million, μ to 0.3, τ_1 and τ_2 to 2 and 1 respectively, average degree to 100, and maximum degree to 300. We took each node of a community of 183 nodes as a query node and ran the algorithm several times for different amounts of timeout. Then, for each value of timeout, we took the average of the metrics calculated on each of the output communities. To show the magnitude of

Table 2 Community discovery: average precision, recall, and F_1 scores (± the standard deviation for F_1) computed over the query nodes on real-world networks

	Karate	Dolphins	Political books	Football	DBLP (10–100)	Amazon (10–100)
R (Clauset 2005)	$P = 0.881$	$P = 0.971$	$P = 0.776$	$P = 0.680$	$P = 0.421$	$P = 0.592$
	$R = 0.589$	$R = 0.323$	$R = 0.481$	$R = 0.735$	$R = 0.239$	$R = 0.288$
	$F_1 = 0.667 \pm 0.246$	$F_1 = 0.446 \pm 0.236$	$F_1 = 0.525 \pm 0.343$	$F_1 = 0.699 \pm 0.372$	$F_1 = 0.261 \pm 0.246$	$F_1 = 0.325 \pm 0.248$
M (Luo et al. 2006)	$P = 0.873$	$P = 0.947$	$P = 0.747$	$P = 0.824$	$P = 0.445$	$P = 0.611$
	$R = 0.689$	$R = 0.422$	$R = 0.558$	$R = 0.894$	$R = 0.285$	$R = 0.351$
	$F_1 = 0.717 \pm 0.292$	$F_1 = 0.503 \pm 0.344$	$F_1 = 0.572 \pm 0.350$	$F_1 = 0.842 \pm 0.260$	$F_1 = 0.281 \pm 0.252$	$F_1 = 0.366 \pm 0.262$
K-Truss (Huang et al. 2014)	$P = 0.592$	$P = 0.718$	$P = 0.717$	$P = 0.854$	$P = 0.259$	$P = 0.529$
	$R = 0.628$	$R = 0.401$	$R = 0.810$	$R = 0.890$	$R = 0.333$	$\mathbf{R = 0.483}$
	$F_1 = 0.554 \pm 0.185$	$F_1 = 0.502 \pm 0.329$	$F_1 = 0.738 \pm 0.280$	$F_1 = 0.865 \pm 0.236$	$F_1 = 0.180 \pm 0.275$	$F_1 = 0.402 \pm 0.274$
APPR (Andersen et al. 2006)	$P = 0.969$	$P = 0.959$	$P = 0.756$	$P = 0.729$	$P = 0.348$	$P = 0.567$
	$R = 0.802$	$R = 0.710$	$R = 0.750$	$\mathbf{R = 0.923}$	$R = 0.369$	$R = 0.419$
	$F_1 = 0.843 \pm 0.217$	$F_1 = 0.749 \pm 0.326$	$F_1 = 0.715 \pm 0.308$	$F_1 = 0.762 \pm 0.319$	$F_1 = 0.247 \pm 0.251$	$F_1 = 0.408 \pm 0.259$
MAPPR (Yin et al. 2017)	$\mathbf{P = 1.000}$	$\mathbf{P = 0.995}$	$P = 0.788$	$P = 0.888$	$P = 0.398$	$P = 0.654$
	$R = 0.712$	$R = 0.281$	$R = 0.603$	$R = 0.897$	$R = 0.341$	$R = 0.340$
	$F_1 = 0.775 \pm 0.295$	$F_1 = 0.387 \pm 0.274$	$F_1 = 0.630 \pm 0.343$	$F_1 = 0.858 \pm 0.231$	$F_1 = 0.271 \pm 0.250$	$F_1 = 0.354 \pm 0.256$
MWC (Bian et al. 2017)	$P = 0.906$	$P = 0.947$	$P = 0.777$	$P = 0.872$	$P = 0.371$	$P = 0.569$
	$R = 0.806$	$R = 0.518$	$R = 0.616$	$R = 0.893$	$\mathbf{R = 0.373}$	$R = 0.403$
	$F_1 = 0.852 \pm 0.021$	$F_1 = 0.643 \pm 0.191$	$F_1 = 0.657 \pm 0.310$	$F_1 = 0.875 \pm 0.245$	$\mathbf{F_1 = 0.312 \pm 0.265}$	$F_1 = 0.406 \pm 0.258$
LTE (Huang et al. 2011)	$P = 0.946$	$P = 0.985$	$P = 0.807$	$P = 0.906$	$\mathbf{P = 0.573}$	$P = 0.678$
	$R = 0.560$	$R = 0.355$	$R = 0.399$	$R = 0.839$	$R = 0.261$	$R = 0.356$
	$F_1 = 0.673 \pm 0.225$	$F_1 = 0.480 \pm 0.238$	$F_1 = 0.472 \pm 0.321$	$F_1 = 0.863 \pm 0.233$	$F_1 = 0.308 \pm 0.249$	$F_1 = 0.403 \pm 0.271$
TCE (Hamann et al. 2017)	$P = 0.920$	$P = 0.971$	$P = 0.796$	$P = 0.900$	$P = 0.540$	$P = 0.649$
	$R = 0.580$	$R = 0.316$	$R = 0.536$	$R = 0.894$	$R = 0.235$	$R = 0.359$
	$F_1 = 0.681 \pm 0.202$	$F_1 = 0.440 \pm 0.225$	$F_1 = 0.572 \pm 0.345$	$\mathbf{F_1 = 0.895 \pm 0.231}$	$F_1 = 0.282 \pm 0.246$	$F_1 = 0.398 \pm 0.279$
LCTC (Huang et al. 2015)	$P = 0.990$	$P = 0.987$	$\mathbf{P = 0.865}$	$\mathbf{P = 0.922}$	$P = 0.563$	$\mathbf{P = 0.812}$
	$R = 0.227$	$R = 0.125$	$R = 0.113$	$R = 0.731$	$R = 0.131$	$R = 0.185$
	$F_1 = 0.364 \pm 0.099$	$F_1 = 0.216 \pm 0.095$	$F_1 = 0.196 \pm 0.072$	$F_1 = 0.798 \pm 0.224$	$F_1 = 0.191 \pm 0.208$	$F_1 = 0.274 \pm 0.188$
SIWO Eq. 4	$\mathbf{P = 1.000}$	$P = 0.985$	$P = 0.759$	$P = 0.894$	$P = 0.115$	$P = 0.410$
	$\mathbf{R = 0.913}$	$R = 0.621$	$R = 0.708$	$R = 0.895$	$R = 0.343$	$R = 0.427$
	$\mathbf{F_1 = 0.952 \pm 0.045}$	$F_1 = 0.735 \pm 0.203$	$F_1 = 0.700 \pm 0.249$	$F_1 = 0.890 \pm 0.222$	$F_1 = 0.140 \pm 0.155$	$F_1 = 0.298 \pm 0.289$
SIWO Eq. 5	$\mathbf{P = 1.000}$	$P = 0.985$	$P = 0.738$	$P = 0.792$	$P = 0.514$	$P = 0.582$
	$R = 0.694$	$\mathbf{R = 0.753}$	$\mathbf{R = 0.830}$	$R = 0.897$	$R = 0.304$	$R = 0.433$
	$F_1 = 0.812 \pm 0.095$	$\mathbf{F_1 = 0.821 \pm 0.242}$	$\mathbf{F_1 = 0.766 \pm 0.286}$	$F_1 = 0.824 \pm 0.227$	$\mathbf{F_1 = 0.312 \pm 0.255}$	$\mathbf{F_1 = 0.418 \pm 0.275}$

Bold numbers indicate the best score for each data set

the expansion, we have also counted the number of nodes visited by the algorithm, as well as the strength and size of the community found so far, per each combination of query node (inside the community) and timeout. The results for this experiment are shown in Table 4.

We can see that recall and F_1 Score have a direct relation with the input timeout, which is expected. This means that SIWO is generating meaningful results, even if there is a limited time. Also, as we give the algorithm more time, it brings nodes that increase the community strength inside the community, until the stopping condition is met (at which point the algorithm will stop even if we give it more time, as illustrated in the last row of the table). To the best of our knowledge, there are no claims regarding the possibility of yielding a result before completion by the previously proposed algorithms, and cannot be (at least trivially) extended to do so. Based on the standard deviations, we can also see that the results do not differ greatly by selecting different query nodes inside the same community. In fact, SIWO seems to provide nearly the same results for all query nodes in a community, if enough time is given to the algorithm. This shows that SIWO does not suffer from one of the most important issues that was imposed by similar local community search algorithms (especially modularity-based methods), which is the sensitivity of the output community to the initial query node. In other words, the expansion using

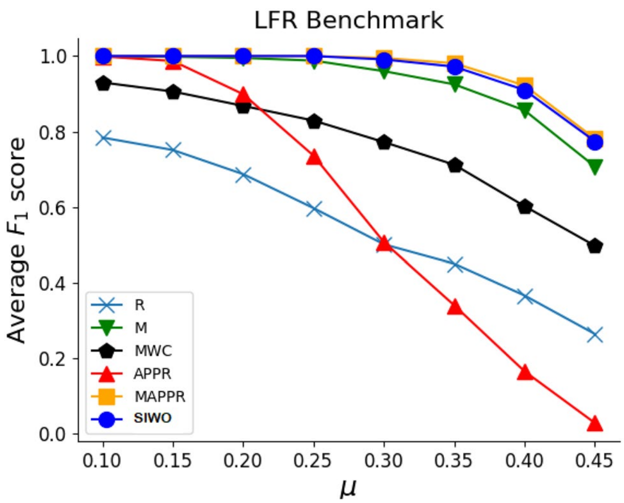

Fig. 3 Community discovery: F_1 average scores computed over all nodes used as query nodes on LFR generated networks as a function of the mixing parameter μ

the SIWO strength function is more likely not to cross community boundaries.

6.3.1 Scalability in large synthetic networks

In an attempt to validate the efficiency and scalability of our algorithm even further, we conducted an experiment on an even larger synthetic LFR graph consisting of 2 million nodes. The graph was generated with parameters $\tau_1 = 3$, $\tau_2 = 1.5$, $\mu = 0.3$, an average degree of 10, and a maximum degree of 30. This particular choice of parameters, resulted in a network with 154,047 communities that is not overly dense, allowing our algorithm to not spend excessive time per node.

We selected 1000 random nodes, ensuring that no two nodes belonged to the same community. The results were again promising, with an average precision of 0.947, an average recall of 0.904, and an average F1 score of 0.920, with a standard deviation of 0.214. The average time spent per node was 1.768s. The low average time compared to the experiment on the LFR network with 1 million nodes highlights the truly local nature of our method, demonstrating its capability to efficiently navigate large networks without being hindered by their size. This also indicates that SIWO's time complexity is indeed a consequence of the community size of the given query node and the magnitude of the expansion (determined by the average degree of the nodes in that community), rather than the size of the whole network.

It is crucial to note that the sheer size of this graph necessitates methods that are truly local and can handle large files without loading them into main memory. This constraint rendered other contenders infeasible for this experiment, as they require substantial memory resources. Our approach's ability to perform efficiently on such a large-scale graph underscores its practicality and robustness in real-world scenarios where memory constraints are often a significant consideration.

Table 3 Average run time (ms) over all query nodes to discover community as a function of the mixing parameter μ

μ	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45
R	09.4	15.3	20.8	25.7	30.6	33.8	45.3	46.7
M	26.6	49.1	78.1	93.3	122	210	330	307
MWC	22.3	44.1	96.5	186	213	315	321	437
APPR	372	669	831	937	1011	1175	1136	1163
MAPPR	404	439	440	433	445	440	487	418
SIWO	**02.1**	**02.9**	**03.8**	**03.9**	**04.4**	**06.5**	**17.3**	**35.4**

Table 4 Quality of SIWO partial search results for given timeout (seconds) in terms of found community size $|C|$, number of nodes visited by algorithm, sum of edge strengths in community $S(C)$, precision P, recall R, and F_1 score ($\pm$ the standard deviation for F_1)

Timeout	|C|	Visited	$S(C)$	P	R	F_1
10	5	35,810	7	1.0	0.027	0.053 $\pm$ 0.013
30	29	209,955	282	1.0	0.160	0.277 $\pm$ 0.011
50	72	374,825	1181	1.0	0.398	0.569 $\pm$ 0.023
70	168	500,589	2154	1.0	0.922	0.955 $\pm$ 0.067
90	182	512,803	2226	1.0	0.999	0.999 $\pm$ 0.003
100	183	512,880	2227	1.0	1.0	1.0 $\pm$ 0.0
110	183	512,880	2227	1.0	1.0	1.0 $\pm$ 0.0

Results averaged over runs using each of 183 nodes as query node in a given ground truth community

6.3.2 Scalability in large real-world networks

Further experiments have also been done to observe the performance of SIWO on five very large real-world networks. The characteristics of these networks (Youtube, Orkut, UK-2002, Twitter, and Friendster) are given in Table 1. The algorithms previously used for local search comparison such as MAPPR and MWC must load the whole network in the main memory, which is infeasible on these very large networks, whereas SIWO only needs to load the nodes encountered during the search.

This is also true for the current versions of Triangle-Based Community Expansion (TCE) (Hamann et al. 2017) and Local Tightness Expansion algorithm (LTE) (Huang et al. 2011), as they are implemented in such a way that the whole network is loaded in the main memory before the algorithm starts. Therefore, we compare SIWO to LCTC (Huang et al. 2015) as it was shown to perform local search on Orkut. However, while the LCTC search algorithm itself is local, it must first create an index of the entire network unlike SIWO which indexes only as needed while the search progresses. The LCTC indexing process requires a significant amount of time and memory to complete, e.g., around 50 GBs of RAM and 4 h for Orkut, and produces a large index file.

Five query nodes were used for each dataset with 1200s as the maximum timeout per node. Table 5 reports the size of the discovered community $|C|$, the time in seconds and memory in MB used by each algorithm. As SIWO indexes while searching, the reported time includes both. LCTC starts searching after the indexing has been done, so the times can be broken down. For a fair comparison, we compare the total time of indexing plus searching as this would be the time required to find the community given a specific node and network without any preprocessing. Nonetheless, even just comparing solely the search time of LCTC with the time SIWO requires, apart from two nodes on Table 5 for the Youtube dataset, SIWO was considerably faster. For SIWO, we have also reported the number of nodes that were visited by the algorithm to show the magnitude of the search. The results confirm that SIWO finds larger communities than LCTC in less time while using much less memory. In fact, the memory requirements of LCTC meant that we were not able to run it on the laptop with 16 GB and had to use a machine with much larger memory. Even with this machine we were not able to complete the indexing process of LCTC on the two largest networks and cannot present results. This highlights how SIWO is a truly local algorithm that can handle very large networks on reasonable commodity hardware.

7 Evaluation of the variants of SIWO

7.1 Evaluation of SIWO+

In this part, we evaluate the extension of SIWO, called SIWO+, described in Sect. 5.1. As SIWO+ has been designed to find the entire partitioning of a given network, we compare it against Louvain (Blondel et al. 2008), EdMot (Li et al. 2019), and Leiden (Traag et al. 2019) which are among the best known global methods, each one optimizing a different objective function to find communities. We also compare it against CPM (Palla et al. 2005), and SCAN (Xu et al. 2007) that find communities of a given graph based on expansion from a random starting node in the network. Codes for Louvain, Leiden, and SCAN can be found in CDlib (Rossetti et al. 2019) python software package.[7] EdMot[8] and CPM[9] also have publicly available codes (Rozemberczki et al. 2020).

Using synthetic networks, we conduct two sets of experiments to analyze the methods' performance when either the size of the network grows or the quality of its community structure decreases.

Evaluation protocol: In both experiments, we use the LFR benchmark (Lancichinetti et al. 2008) to generate the networks with ground-truth communities. In the first experiment, μ is equal to 0.15 and n, the size of the networks, varies from 100 to 30,000. Since the size of the network increases, the number and the size of the communities increase accordingly. In the second experiment, we use the networks described in Sect. 6.2. For each data set, the partition provided by each method is compared with the ground truth according to Normalized Mutual Information (NMI) (Danon et al. 2005). We report the average score, with standard deviation, computed over 10 runs.

Experimental results: Table 6 shows that SIWO+ achieves perfect NMI scores, which means it can find communities equivalent to the ground truth whatever the size of the networks. Louvain, EdMot, and Leiden cannot maintain such a high accuracy when the network's size n is higher than 1000 as they inherently suffer from the resolution limit, which means communities that are smaller than a scale cannot be resolved (Lancichinetti and Fortunato 2011). Although CPM and SCAN can perform relatively well in most cases, they both fail when the network's size becomes large. More importantly, they require a fine-tuning of their parameters to fulfill the task successfully. SIWO+, which does not need any predetermined parameter, finds the best results nonetheless. Even if it has been initially designed to

[7] https://github.com/GiulioRossetti/cdlib.

[8] https://github.com/benedekrozemberczki/EdMot.

[9] https://bit.ly/3q1QBJj.

Table 5 Performance of SIWO and LCTC on 5 large networks with 5 query nodes each in terms of maximum memory required (RAM) and size of the index file needed by LCTC (Index Size) in MB, number of nodes visited for SIWO, size of the found community $|C|$ and time in seconds taken by each algorithm

Dataset	Node ID	SIWO				LCTC (Huang et al. 2015)			
		RAM	Visited	|C|	Time	RAM	Index size	|C|	Time (indexing + search)
YouTube	3	7410	4999	4	0.8	557	61	4	64.0 (49.6 + 14.4)
	2002		11,599	7	2.0			3	64.3 (49.6 + 14.7)
	19,973		29,400	7	8.5			2	65.1 (49.6 + 15.5)
	9659		101,026	20	1200.0			8	63.2 (49.6 + 13.6)
	97,648		545,630	4308	943.7			4	63.1 (49.6 + 13.5)
UK-2002	14,347	1878	29	6	0.1	18,834	2299	6	15,100.9 (13,663.5 + 1437.4)
	31,025		6176	357	15.8			3	15,121.8 (13,663.5 + 1458.3)
	78,984		21,851	928	30.1			54	15,109.3 (13,663.5 + 1445.8)
	44,379		157,161	1608	316.3			14	15,097.2 (13,663.5 + 1433.7)
	61,384		335,346	459	496.7			22	15,069.1 (13,663.5 + 1405.6)
Orkut	98,171	7419	1,092,307	1216	522.4	56,934	6021	12	20,131.2 (19,438.1 + 693.1)
	2		1,131,011	1400	392.1			14	20,154.7 (19,438.1 + 716.6)
	52,002		1,174,254	1310	418.9			19	20,144.2 (19,438.1 + 706.1)
	86,525		1,196,757	1273	572.5			3	20,139.2 (19,438.1 + 701.1)
	79,847		1,201,051	1334	439.3			109	20,163.8 (19,438.1 + 725.7)
Twitter	30,634	6918	36	9	17.9	N/A			
	90,205		47	6	22.9				
	20,648		745	25	51.38				
	101		1684	2	184.7				
	43,748		17,186	10	53.32				
Friendster	72,759	7428	1262	6	54.2	N/A			
	18,427		1623	14	46.2				
	34,455		179,325	104	1200.0				
	69,947		253,145	473	1200.0				
	84,063		321,648	730	1200.0				

The wall time for a community search is set to 1200 s

be a local approach, it outperforms state-of-the-art global methods for identifying the whole community structure of a network. Moreover, its null standard deviations indicate that the random selection of a node from the unexplored part of the network to initiate the search and then detect the other communities has an insignificant impact on the final partitioning. As shown by the small standard deviation obtained over the 10 conducted runs, it is also the case for Louvain, EdMot, and Leiden, which are not deterministic. Another set of experiments has been performed. to evaluate the seed choice effect on the community structure identified by our algorithm. The results are not presented due to the lack of space. However, they demonstrate that the results of SIWO remain stable regardless of the seeds' degrees: low, middle, or high.

Table 7 shows that SIWO+ is very robust to the community's mixing parameter as it achieves satisfactory performances even when μ is equal to 0.45. It is essential to notice that, in this experiment, CPM and SCAN have not been able to find the community of many nodes when μ is higher than 0.3. SCAN should be used with ϵ equals to 0.3 to perform relatively well, whereas the recommended value is between 0.5 and 0.8 (Xu et al. 2007). Even though EdMot and Leiden are known to improve Louvain in terms of motif-awareness and connectedness, they achieve similar NMI scores. Thus, in this experiment, SIWO+ outperformed the existing methods in all trials, and it has been able to find the community of all the network nodes, while CPM leaves many of them unassigned (up to 1000 nodes).

Concerning the detection of outliers, we added 1% outlier nodes by randomly selecting 100 nodes and attaching then to these outliers, making these last ones peripheral. EdMot, Louvain, and Leiden included these into communities. SIWO+, with the optional Step 5 skipped, identified all outliers as singletons. CPM and SCAN identified outliers by not inserting them in any detected communities but when $\mu > 0.25$, up to 75% of nodes are also erroneously considered outliers by CPM.

Table 6 Community detection: average NMI scores ± the standard deviation computed over 10 runs, in function of the network's size n

	$n = 100$	$n = 300$	$n = 1000$	$n = 3000$	$n = 10,000$	$n = 30,000$
CPM (Palla et al. 2005)	0.000 ± 0.000	0.884 ± 0.000	0.995 ± 0.000	0.998 ± 0.000	0.999 ± 0.000	0.999 ± 0.000
SCAN (Xu et al. 2007)	0.000 ± 0.000	**1.000 ± 0.000**	**1.000 ± 0.000**	0.999 ± 0.000	0.999 ± 0.000	0.998 ± 0.000
Louvain (Blondel et al. 2008)	**1.000 ± 0.000**	**1.000 ± 0.000**	0.996 ± 0.000	0.974 ± 0.000	0.921 ± 0.000	0.887 ± 0.000
EdMot (Li et al. 2019)	**1.000 ± 0.000**	0.997 ± 0.007	0.995 ± 0.003	0.969 ± 0.001	0.917 ± 0.001	0.881 ± 0.001
Leiden (Traag et al. 2019)	**1.000 ± 0.000**	0.997 ± 0.006	0.995 ± 0.001	0.977 ± 0.001	0.921 ± 0.001	0.884 ± 0.001
SIWO+	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**

Bold numbers indicate the best score for each data set

Table 7 Community detection: Average NMI scores ± the standard deviation computed over 10 runs, in function of the mixing parameter μ. Bold numbers indicate the best score for each data set

	$\mu = 0.10$	$\mu = 0.15$	$\mu = 0.20$	$\mu = 0.25$	$\mu = 0.30$	$\mu = 0.35$	$\mu = 0.4$	$\mu = 0.45$
CPM	0.997 ± 0.000	0.993 ± 0.000	0.987 ± 0.000	0.981 ± 0.000	0.966 ± 0.000	0.955 ± 0.000	0.901 ± 0.000	0.801 ± 0.000
SCAN	**1.000 ± 0.000**	**1.000 ± 0.000**	**0.999 ± 0.000**	0.991 ± 0.000	0.938 ± 0.000	0.849 ± 0.000	0.658 ± 0.000	0.398 ± 0.000
Louvain	0.902 ± 0.000	0.885 ± 0.000	0.874 ± 0.000	0.861 ± 0.000	0.835 ± 0.000	0.822 ± 0.000	0.800 ± 0.000	0.735 ± 0.000
EdMot	0.897 ± 0.002	0.883 ± 0.002	0.869 ± 0.002	0.852 ± 0.002	0.831 ± 0.003	0.815 ± 0.004	0.799 ± 0.004	0.734 ± 0.004
Leiden	0.899 ± 0.002	0.884 ± 0.002	0.870 ± 0.002	0.849 ± 0.003	0.824 ± 0.003	0.807 ± 0.003	0.777 ± 0.003	0.680 ± 0.005
SIWO+	**1.000 ± 0.000**	**1.000 ± 0.000**	**0.999 ± 0.000**	**0.999 ± 0.000**	**0.997 ± 0.000**	**0.988 ± 0.001**	**0.952 ± 0.001**	**0.834 ± 0.004**

7.2 Evaluation of SIWOw on weighted networks

This section aims to evaluate the quality of our approach for handling edge weights through experiments done with the variant of SIWO, called SIWOw, described in Sect. 5.2. While SIWO is first and foremost a local community search algorithm, most algorithms for weighted networks are global community detection algorithms. To get a fair idea of the performance of weighted SIWO compared to other methods, we thus use SIWOw for global community detection (i.e., we extend SIWO+ to handle edge weights) and compare it against two of the same algorithms used in Sect. 7.1: Leiden and Louvain. The other previously evaluated methods are not able to handle weighted networks and so are excluded here. Moreover, Louvain has been shown to be one of the best performing community detection algorithms in a comparative analysis (Yang et al. 2016) and Leiden improves upon Louvain by guaranteeing well-connected communities (Traag et al. 2019) so we are confident in using them as a benchmark for SIWOw. Another closely related work to SIWOw is Zheng et al. (2017), in which Zheng et al. introduce a novel community model that takes edge weights into consideration. However, we were unable to include the approach in our comparative analysis due to the lack of publicly available executable code for their method.

Synthetic data sets: We evaluate all algorithms on a set of synthetic networks which have ground-truth communities using NMI. There is an unfortunate lack of real-world weighted networks with ground-truth communities available. We thus use a range of parameter settings for the synthetic network generator to ensure a variety of network structures. More precisely, we generated 720 unique networks with the LFR weighted network generator benchmark (Lancichinetti and Fortunato 2009). We generate five different networks for each parameter combination of average degree in {15, 20, 25}, maximum degree in {50, 75, 100}, exponent for weight distribution $\beta \in \{1.5, 2\}$, and topology mixing parameter μ_t and weight mixing parameter μ_w taking the same values in $\{0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45\}$.

Baselines and settings: We compare SIWOw to the weighted versions of two community detection algorithms: Louvain and Leiden. We evaluate weighted SIWOw using the arithmetic mean, geometric mean, harmonic mean, and minimum for the weight combination function. We denote these different versions as SIWOw(a), SIWOw(g), SIWOw(h), and SIWOw(m), respectively.

Experimental results: Table 8 shows that SIWOw outperforms Leiden and Louvain at all levels of μ_w. SIWOw achieves perfect or near-perfect NMI even as the modularity of the ground truth community structure drops as μ_w increases. The performance of all the algorithms drops as μ_w increases but SIWOw is still able to achieve an NMI score of 0.995 when μ_w equals 0.45, demonstrating the algorithm's robustness to the mixing parameter of the synthetic networks. We note that all of the algorithms achieve better performance than in the unweighted evaluation, demonstrating the extra information provided by the edge weights and the value of an algorithm's ability to consider this information.

Table 8 Community detection: Average NMI scores ± the standard deviation computed over 90 runs, as a function of the mixing parameters with $\mu_w = \mu_t$. Q value indicates the modularity of the ground-truth community structure

	$\mu_w = 0.10$ $Q = 0.888$	$\mu_w = 0.15$ $Q = 0.841$	$\mu_w = 0.20$ $Q = 0.792$	$\mu_w = 0.25$ $Q = 0.743$	$\mu_w = 0.30$ $Q = 0.693$	$\mu_w = 0.35$ $Q = 0.643$	$\mu_w = 0.40$ $Q = 0.594$	$\mu_w = 0.45$ $Q = 0.544$
SIWOw(a)	**0.999 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**0.999 ± 0.000**	**0.998 ± 0.000**	**0.995 ± 0.001**
SIWOw(g)	**0.999 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**0.999 ± 0.000**	**0.998 ± 0.000**	**0.995 ± 0.001**
SIWOw(h)	**0.999 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**0.999 ± 0.000**	**0.998 ± 0.000**	**0.995 ± 0.001**
SIWOw(m)	**0.999 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	0.999 ± 0.000	**0.999 ± 0.000**	**0.998 ± 0.000**	**0.995 ± 0.001**
Leiden	0.966 ± 0.001	0.964 ± 0.002	0.963 ± 0.002	0.962 ± 0.002	0.960 ± 0.002	0.955 ± 0.002	0.951 ± 0.002	0.946 ± 0.003
Louvain	0.924 ± 0.002	0.915 ± 0.002	0.907 ± 0.003	0.898 ± 0.003	0.888 ± 0.003	0.877 ± 0.003	0.865 ± 0.003	0.851 ± 0.004

SIWOw also exhibits a lower standard deviation than Leiden or Louvain. Recall that the 90 runs for each μ_w value in Table 8 consist of networks that have varying parameter values and thus varying structures. The low standard deviation of SIWOw indicates that it does not suffer from significantly worse performance for networks in our data set generated with certain combinations of parameters and is thus robust to network features such as average degree.

Table 8 also shows that none of the weight combination functions for SIWOw significantly outperforms the other. In fact, there is only one case where one version performs worse than the others: SIWOw(m) when μ_w equals 0.30. This result may seem counter-intuitive as the different functions will result in different values in many cases. For example, the geometric mean will result in a value close to zero whenever one edge has a weight very close to zero but the arithmetic may not, and the minimum does not use most of the information. However, our results show that the choice does not have a material impact on performance which is consistent with work on the weighted clustering coefficient which found a similar value regardless of the mean used (Opsahl and Panzarasa 2009).

Table 9 shows that SIWOw finds a number of communities much closer to the true value than Leiden and Louvain. Leiden and Louvain both work by attempting to optimize modularity and thus suffer from the resolution limit and fail to separate small communities. They find far fewer communities than the ground truth. SIWOw tends to find almost the same number of communities as the true number, and this difference increases as μ_w increases. SIWO relies on triangle structures to assign edge strength. As μ_w increases and communities become less dense there may be fewer triangles within communities. If there are few enough triangles within a community, the intra-community edges will be considered weak by SIWO and the nodes of the community will not be assigned to one community. Of course, there is the question of whether a ground-truth community that consists of nodes without many common neighbors is truly a community.

Our experiments on weighted networks show that SIWOw performs well at uncovering community structure with edge weights. Combined with our results from Sect. 7, this highlights two of the main advantages of SIWO. Unlike many other local community search algorithms, SIWO is able to handle weighted networks. And unlike weighted community detection algorithms such as Louvain and Leiden, SIWO only requires local information and can thus be applied to large networks that are unable to fit in main memory. The flexibility to be used for either global community detection or local community search for both weighted and unweighted networks of greatly varying sizes distinguishes SIWO from other algorithms.

8 Conclusion and discussion

Our method, SIWO, is parameter-free and uses the notion of triangles, which enables accurate, deterministic, and fast local community discovery in networks. While other algorithms have used triangles before, SIWO distinguishes itself through its distinctive approach of utilizing strength values on edges for normalizing the number of triangles containing that edge, which has proven experimentally to be very effective.

SIWO is non-parametric, which offers a significant advantage over many local community detection methods that require fine-tuning of their parameters, often difficult to interpret and adjust. SIWO is also robust to the query node used to start the search and will find the same community whether it begins in the core or on the periphery of a community.

Furthermore, SIWO does not necessitate loading the entire graph into main memory, allowing it to operate efficiently even on a regular laptop with limited resources for large networks with tens of millions of nodes and hundreds of millions of edges. This allows SIWO to provide the core of the community within a given time budget, even if the network is too large and dense for the search to complete. In contrast, most current approaches require a massive amount

Table 9 Community detection: Absolute number of detected communities ± the standard deviation computed over 90 runs, as a function of the mixing parameters with $\mu_w = \mu_t$. $\bar{N}_{GT}$ represents the average number of ground truth communities

	$\mu_w = 0.10$ $\bar{N}_{GT} = 339 \pm 6$	$\mu_w = 0.15$ $\bar{N}_{GT} = 343 \pm 6$	$\mu_w = 0.20$ $\bar{N}_{GT} = 345 \pm 6$	$\mu_w = 0.25$ $\bar{N}_{GT} = 344 \pm 7$	$\mu_w = 0.30$ $\bar{N}_{GT} = 344 \pm 6$	$\mu_w = 0.35$ $\bar{N}_{GT} = 345 \pm 7$	$\mu_w = 0.40$ $\bar{N}_{GT} = 343 \pm 6$	$\mu_w = 0.45$ $\bar{N}_{GT} = 342 \pm 7$
SIWOw(a)	**339 ± 6**	**344 ± 6**	**346 ± 6**	**345 ± 7**	**346 ± 6**	**349 ± 7**	**350 ± 7**	358 ± 7
SIWOw(g)	**339 ± 6**	**344 ± 6**	**346 ± 6**	**345 ± 7**	**347 ± 6**	**349 ± 7**	**351 ± 7**	**357 ± 7**
SIWOw(h)	**340 ± 6**	**345 ± 6**	**347 ± 6**	**346 ± 7**	**347 ± 6**	**350 ± 7**	**353 ± 7**	**360 ± 7**
SIWOw(m)	**341 ± 6**	**346 ± 6**	**348 ± 6**	**348 ± 7**	**349 ± 6**	**352 ± 7**	**354 ± 7**	**362 ± 7**
Leiden	206 ± 4	203 ± 4	203 ± 5	202 ± 4	201 ± 4	196 ± 4	194 ± 4	190 ± 4
Louvain	134 ± 3	124 ± 3	116 ± 2	108 ± 2	100 ± 2	93 ± 2	85 ± 2	78 ± 1

of memory to be able to operate on large networks, either to fit the whole graph to process or to build indexes, and often return an empty set if interrupted.

SIWO serves as a versatile framework capable of handling different types of networks. In this paper, we demonstrated how SIWO can be generalized to perform local search on weighted networks, or perform global community detection by iteratively applying local search on unexplored parts of the network. In addition, it can detect overlapping communities or outliers.

The above conveniences are characteristics that no other method has and fundamentally differentiate SIWO from the state-of-the-art methods.

In summary, SIWO is a fast, highly performant, and flexible algorithm for local community detection that distinguishes itself from existing methods through its non-parametric nature, efficient memory usage, and strong performance in various scenarios.

Funding Funding were provided by Alberta Machine Intelligence Institute, Laboratoire Hubert Curien and Canadian Institute for Advanced Research.

References

Adamic LA, Glance N (2005) The political blogosphere and the 2004 U.S. Election: divided they blog. In: Proceedings of the 3rd international workshop on link discovery. LinkKDD '05. Association for Computing Machinery, New York, NY, USA, pp 36–43. https://doi.org/10.1145/1134271.1134277

Akbas E, Zhao P (2017) Truss-based community search. Proc VLDB Endow 10(11):1298–1309. https://doi.org/10.14778/3137628.3137640

Andersen R, Chung F, Lang K (2006) Local graph partitioning using pagerank vectors. In: 2006 47th annual IEEE symposium on foundations of computer science (FOCS'06), pp 475–486. https://doi.org/10.1109/FOCS.2006.44

Bagrow JP, Bollt EM (2005) Local method for detecting communities. Phys Rev E 72:046108. https://doi.org/10.1103/PhysRevE.72.046108

Baltsou G, Christopoulos K, Tsichlas K (2022) Local community detection: a survey. IEEE Access 10:110701–110726

Barbieri N, Bonchi F, Galimberti E, Gullo F (2015) Efficient and effective community search. Data Min Knowl Disc 29(5):1406–1433. https://doi.org/10.1007/s10618-015-0422-1

Barrat A, Barthelemy M, Pastor-Satorras R, Vespignani A (2004) The architecture of complex weighted networks. Proc Natl Acad Sci 101(11):3747–3752. https://doi.org/10.1073/pnas.0400087101

Bian Y, Ni J, Cheng W, Zhang X (2017) Many heads are better than one: local community detection by the multi-walker chain. In: 2017 IEEE international conference on data mining (ICDM), pp 21–30. https://doi.org/10.1109/ICDM.2017.11

Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech: Theory Exp 2008(10):10008. https://doi.org/10.1088/1742-5468/2008/10/p10008

Brunato M, Hoos HH, Battiti R (2008) On effectively finding maximal quasi-cliques in graphs. In: Maniezzo V, Battiti R, Watson J-P (eds) Learning and Intelligent Optimization. Springer, Berlin, pp 41–55

Chen J, Zaïane OR, Goebel R (2010) In: Memon N, Alhajj R (eds) Detecting communities in social networks using local information. Springer, Vienna, pp 197–214. https://doi.org/10.1007/978-3-7091-0294-7_11

Clauset A (2005) Finding local community structure in networks. Phys Rev E 72(2):026132. https://doi.org/10.1103/physreve.72.026132

Cui W, Xiao Y, Wang H, Lu Y, Wang W (2013) Online search of overlapping communities. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data. SIGMOD '13. Association for Computing Machinery, New York, NY, USA, pp 277–288. https://doi.org/10.1145/2463676.2463722

Cui W, Xiao Y, Wang H, Wang W (2014) Local search of communities in large graphs. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. SIGMOD '14. Association for Computing Machinery, New York, NY, USA, pp 991–1002. https://doi.org/10.1145/2588555.2612179

Danon L, Díaz-Guilera A, Duch J, Arenas A (2005) Comparing community structure identification. J Stat Mech Theory Exp 2005(09):09008–09008. https://doi.org/10.1088/1742-5468/2005/09/p09008

Dao VL, Bothorel C, Lenca P (2020) Community structure: a comparative evaluation of community detection methods. Netw Sci 8(1):1–41. https://doi.org/10.1017/nws.2019.59

De Meo P, Ferrara E, Fiumara G, Provetti A (2014) Mixing local and global information for community detection in large networks. J Comput Syst Sci 1:72–87

Dilmaghani S, Brust MR, Danoy G, Bouvry P (2021) Community detection in complex networks: a survey on local approaches. In:

Asian conference intelligent information and database systems, pp 757–767
Fagnan J, Zaïane O, Barbosa D (2014) Using triads to identify local community structure in social networks. In: IEEE/ACM international conference on advances in social networks analysis and mining, pp 108–112. https://doi.org/10.1109/ASONAM.2014.6921568
Fang Y, Huang X, Qin L, Zhang Y, Zhang W, Cheng R, Lin X (2019) A survey of community search over big graphs. VLDB J 29(1):353–392. https://doi.org/10.1007/s00778-019-00556-x
Fang Y, Yang Y, Zhang W, Lin X, Cao X (2020) Effective and efficient community search over large heterogeneous information networks. Proc VLDB Endow 13(6):854–867. https://doi.org/10.14778/3380750.3380756
Fortunato S (2010) Community detection in graphs. Phys Rep 486(3–5):75–174. https://doi.org/10.1016/j.physrep.2009.11.002
Fortunato S, Hric D (2016) Community detection in networks: a user guide. Phys Rep 659:1–44. https://doi.org/10.1016/j.physrep.2016.09.002
Gharaghooshi SZ, Zaiane OR, Largeron C, Zafarmand M, Liu C (2020) Addressing the resolution limit and the field of view limit in community mining. In: Symposium on intelligent data analysis (IDA'20)
Girvan M, Newman MEJ (2002) Community structure in social and biological networks. Proc Natl Acad Sci 99(12):7821–7826. https://doi.org/10.1073/pnas.122653799
Granovetter M (1983) The strength of weak ties: a network theory revisited. Sociol Theory 1:201–233. https://doi.org/10.2307/202051
Hamann M, Röhrs E, Wagner D (2017) Local community detection based on small cliques. Algorithms 10(3):90. https://doi.org/10.3390/a10030090
Huang J, Sun H, Liu Y, Song Q, Weninger T (2011) Towards online multiresolution community detection in large-scale networks. PLOS ONE 6(8):1–11. https://doi.org/10.1371/journal.pone.0023829
Huang X, Lakshmanan LVS, Yu JX, Cheng H (2015) Approximate closest community search in networks. Proc VLDB Endow 9(4):276–287. https://doi.org/10.14778/2856318.2856323
Huang X, Lakshmanan LVS, Xu J (2019) Community search over big graphs. Synth Lect Data Manag 14(6):1–206. https://doi.org/10.2200/s00928ed1v01y201906dtm061
Huang X, Cheng H, Qin L, Tian W, Yu JX (2014) Querying k-truss community in large and dynamic graphs. In: Proceedings of the 2014 ACM SIGMOD international conference on management of data. SIGMOD '14. Association for Computing Machinery, New York, NY, USA, pp 1311–1322. https://doi.org/10.1145/2588555.2610495
Kwak H, Lee C, Park H, Moon S (2010) What is twitter, a social network or a news media? In: Proceedings of the 19th international conference on world wide web, New York, NY, USA, pp 591–600 . https://doi.org/10.1145/1772690.1772751
Lancichinetti A, Fortunato S (2009) Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Phys Rev E 80(1):016118. https://doi.org/10.1103/physreve.80.016118
Lancichinetti A, Fortunato S (2011) Limits of modularity maximization in community detection. Phys Rev E 84(6):066122. https://doi.org/10.1103/physreve.84.066122
Lancichinetti A, Fortunato S, Radicchi F (2008) Benchmark graphs for testing community detection algorithms. Phys Rev E 78(4):046110. https://doi.org/10.1103/physreve.78.046110
Li P-Z, Huang L, Wang C-D, Lai J-H (2019) Edmot: an edge enhancement approach for motif-aware community detection. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '19. Association for Computing Machinery, New York, NY, USA, pp 479–487. https://doi.org/10.1145/3292500.3330882
Luo W, Zhang D, Jiang H, Ni L, Hu Y (2018) Local community detection with the dynamic membership function. IEEE Trans Fuzzy Syst 26(5):3136–3150. https://doi.org/10.1109/tfuzz.2018.2812148
Luo D, Bian Y, Yan Y, Liu X, Huan J, Zhang X (2020) Local community detection in multiple networks. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '20. Association for Computing Machinery, New York, NY, USA, pp 266–274. https://doi.org/10.1145/3394486.3403069
Luo F, Wang JZ, Promislow E (2006) Exploring local community structures in large networks. In: 2006 IEEE/WIC/ACM international conference on web intelligence (WI 2006 main conference proceedings)(WI'06), pp 233–239. https://doi.org/10.1109/WI.2006.72
Lusseau D (2003) The emergent properties of a dolphin social network. Proc R Soc Lond Ser B Biol Sci 270:S186–S188. https://doi.org/10.1098/rsbl.2003.0057
Milo R (2002) Network motifs: simple building blocks of complex networks. Science 298(5594):824–827. https://doi.org/10.1126/science.298.5594.824
Onnela J-P, Saramäki J, Kertész J, Kaski K (2005) Intensity and coherence of motifs in weighted complex networks. Phys Rev E 71(6):065103. https://doi.org/10.1103/physreve.71.065103
Opsahl T, Panzarasa P (2009) Clustering in weighted networks. Soc Netw 31(2):155–163. https://doi.org/10.1016/j.socnet.2009.02.002
Palla G, Derényi I, Farkas I, Vicsek T (2005) Uncovering the overlapping community structure of complex networks in nature and society. Nature 435(7043):814–818. https://doi.org/10.1038/nature03607
Peel L, Larremore DB, Clauset A (2017) The ground truth about metadata and community detection in networks. Sci Adv 3(5):1602548. https://doi.org/10.1126/sciadv.1602548
Pons P, Latapy M (2005) Computing communities in large networks using random walks. In: Yolum P, Güngör T, Gürgen F, Özturan C (eds) Computer and information sciences—ISCIS 2005. Springer, Berlin, pp 284–293
Rabbany R, Zaiane OR (2015) Evaluation of community mining algorithms in the presence of attributes. In: Li X-L, Cao T, Lim E-P, Zhou Z-H, Ho T-B, Cheung D (eds) Trends and applications in knowledge discovery and data mining. Lecture Notes in Computer Science. Springer, pp 152–163. https://doi.org/10.1007/978-3-319-25660-3_13
Raghavan UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76(3):036106. https://doi.org/10.1103/physreve.76.036106
Rossetti G, Milli L, Cazabet R (2019) CDLIB: a python library to extract, compare and evaluate communities from complex networks. Appl Netw Sci 4:1–26. https://doi.org/10.1007/s41109-019-0165-9
Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci 105(4):1118–1123. https://doi.org/10.1073/pnas.0706851105
Rozemberczki B, Kiss O, Sarkar R (2020) Karate club: An API oriented open-source python framework for unsupervised learning on graphs. In: Proceedings of the 29th ACM international conference on information and knowledge management. https://doi.org/10.1145/3340531.3412757
Sakr S, Bonifati A, Voigt H, Iosup A, Ammar K, Angles R, Aref W, Arenas M, Besta M, Boncz PA, Daudjee K, Valle ED, Dumbrava S, Hartig O, Haslhofer B, Hegeman T, Hidders J, Hose K, Iamnitchi A, Kalavri V, Kapp H, Martens W, Özsu MT, Peukert E,

Plantikow S, Ragab M, Ripeanu MR, Salihoglu S, Schulz C, Selmer P, Sequeda JF, Shinavier J, Szárnyas G, Tommasini R, Tumeo A, Uta A, Varbanescu AL, Wu H-Y, Yakovets N, Yan D, Yoneki E (2021) The future is big graphs: a community view on graph processing systems. Commun ACM 64(9):62–71. https://doi.org/10.1145/3434642

Saramäki J, Kivelä M, Onnela J-P, Kaski K, Kertesz J (2007) Generalizations of the clustering coefficient to weighted complex networks. Phys Rev E 75(2):027105. https://doi.org/10.1103/physreve.75.027105

Schaeffer SE (2007) Graph clustering. Comput Sci Rev 1(1):27–64. https://doi.org/10.1016/j.cosrev.2007.05.001

Slater N, Itzchack R, Louzoun Y (2014) Mid size cliques are more common in real world networks than triangles. Netw Sci 2(3):387–402. https://doi.org/10.1017/nws.2014.22

Souravlas S, Sifaleras A, Tsintogianni M, Katsavounis S (2021) A classification of community detection methods in social networks: a survey. Int J Gen Syst 50(1):63–91

Sozio M, Gionis A (2010) The community-search problem and how to plan a successful cocktail party. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '10. Association for Computing Machinery, New York, NY, USA, pp 939–948. https://doi.org/10.1145/1835804.1835923

Su X, Xue S, Liu F, Wu J, Yang J, Zhou C, Hu W, Paris C, Nepal S, Jin D, Sheng QZ, Yu PS (2022) A comprehensive survey on community detection with deep learning. IEEE Trans Neural Netw Learn Syst 1–21 (2022)

Su X, Xue S, Liu F, Wu J, Yang J, Zhou C, Hu W, Paris C, Nepal S, Jin D, Sheng Q, Yu P (2022) A comprehensive survey on community detection with deep learning. IEEE Trans Neural Netw Learn Syst 1–21. https://doi.org/10.1109/TNNLS.2021.3137396

Takaffoli M, Rabbany R, Zaiane OR (2013) Incremental local community identification in dynamic social networks. In: IEEE/ACM international conference on social networks analysis and mining

Traag VA, Waltman L, van Eck NJ (2019) From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep 9(1):5233. https://doi.org/10.1038/s41598-019-41695-z

Xu X, Yuruk N, Feng Z, Schweiger TAJ (2007) SCAN: A structural clustering algorithm for networks. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '07. Association for Computing Machinery, New York, NY, USA, pp 824–833. https://doi.org/10.1145/1281192.1281280

Yang J, Leskovec J (2013) Defining and evaluating network communities based on ground-truth. Knowl Inf Syst 42(1):181–213. https://doi.org/10.1007/s10115-013-0693-z

Yang Z, Algesheimer R, Tessone CJ (2016) A comparative analysis of community detection algorithms on artificial networks. Sci Rep 6(1):1–18. https://doi.org/10.1038/srep30750

Yin H, Benson AR, Leskovec J, Gleich DF (2017) Local higher-order graph clustering. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. KDD '17. Association for Computing Machinery, New York, NY, USA, pp 555–564. https://doi.org/10.1145/3097983.3098069

Zachary WW (1977) An information flow model for conflict and fission in small groups. J Anthropol Res 33(4):452–473. https://doi.org/10.1086/jar.33.4.3629752

Zheng Z, Ye F, Li R-H, Ling G, Jin T (2017) Finding weighted k-truss communities in large networks. Inf Sci 417:344–360. https://doi.org/10.1016/j.ins.2017.07.012