IMpart: A Partitioning-based Parallel Approach to Accelerate Influence Maximization

Reet Barik
Washington State University
Pullman, WA, USA
reet.barik@wsu.edu

Marco Minutoli
Pacific Northwest National Laboratory
Richland, WA, USA
marco.minutoli@pnnl.gov

Mahantesh Halappanavar
Pacific Northwest National Laboratory
Richland, WA, USA
mahantesh.halappanavar@pnnl.gov

Ananth Kalyanaraman
Washington State University
Pullman, WA, USA
ananth@wsu.edu

Abstract*—Influence maximization (IM) is a fundamental operation among graph problems that involve simulating a stochastic diffusion process on real-world networks. Given a graph $G(V, E)$, the objective is to identify a small set of key influential "seeds"—i.e., a fixed-size set of k nodes, which when influenced is likely to lead to the maximum number of nodes in the network getting influenced. The problem has numerous applications including (but not limited to) viral marketing in social networks, epidemic control in contact networks, and in finding influential proteins in molecular networks. Despite its importance, application of influence maximization at scale continues to pose significant challenges. While the problem is NP-hard, efficient approximation algorithms that use greedy hill climbing are used in practice. However those algorithms consume hours of multithreaded execution time even on modest-sized inputs with hundreds of thousands of nodes. In this paper, we present `IMpart`, a partitioning-based approach to accelerate greedy hill climbing based IM approaches on both shared and distributed memory computers. In particular, we present two parallel algorithms—one that uses graph partitioning (`IMpart-metis`) and another that uses community-aware partitioning (`IMpart-gratis`)—with provable guarantees on the quality of approximation. Experimental results show that our approaches are able to deliver two to three orders of magnitude speedup over a state-of-the-art multithreaded hill climbing implementation with negligible loss in quality. For instance, on one of the modest-sized inputs (Slashdot: 73K nodes; 905K edges), our partitioning-based shared memory implementation yields 4610× speedup, reducing the runtime from 9h 36m to 7 seconds on 128 threads. Furthermore, our distributed memory implementation enhances problem size reach to graph inputs with $\times 10^6$ nodes and $\times 10^8$ edges and enables sub-minute computation of IM solutions.

I. Introduction

Finding influential actors in a network is a fundamental problem in many real-world applications—e.g., in viral marketing on social networks [1], or finding important proteins in a protein-protein interaction network [2]. Influence refers to node activations that can be either deterministically or stochastically simulated through a diffusion process. For instance in networked epidemiology, compartmental Susceptible-Infected-Recovered (SIR) models can be expressed as a diffusion process over a network. The equivalent formulation considers each node of the graph to be in one of the three states of the SIR model by retaining their conventional semantics. The additional constraint introduced is that an epidemic spreads only through the edges of the contact (social) network. Such models enable us to identify a small cohort of key actors who optimize the underlying diffusion processes.

The computational problem of identifying such cohort of actors in a social network is known as the *Influence Maximization* (or IM, for short) problem [3]. While IM has been of significant interest due to an increase in networked applications, it is also particularly interesting from a theoretical stand point. Kempe *et al.* [4] showed that the problem is NP-hard under two simple but generic diffusion models. However, the objective function for IM has been shown to be submodular, leading to a greedy hill climbing (`GHC`) algorithm with $(1 - 1/e - \varepsilon)$-approximation guarantee [4], where ε is a parameter to control accuracy.

To avoid the high computational cost of approximation algorithms, many heuristic schemes have been proposed in the literature [5]. Several lines of work have attempted to identify important vertices by leveraging centrality measures [6], [7] or other cheaper heuristics based on topological traits of vertices (e.g., degree count or bridges). Based on the intuition that most vertices have a limited range of influence, another line of research uses schemes to leverage the community structure of the input network to identify the seed set [8]–[13].

Our work extends this latter line of work by taking a more generic partition-based approach to IM. In particular, in this paper, we explore the use of graph partitioning and graph clustering—both individually and in combination—to devise efficient parallel influence maximization implementations. Intuitively, decomposing the graph into internally well-connected partitions can give seed candidates closer access to vertices that they are more likely to influence. The objective functions for graph partitioning aims to optimize for balanced partitions of a graph such that the number of edges across two partitions (cut edges) is minimized while those for graph clustering aims to find subsets of vertices that are densely connected within a partition (also called as a cluster or a community) and sparsely with the rest of the graph. Both partitioning and clustering are known to be NP-hard [14], [15].

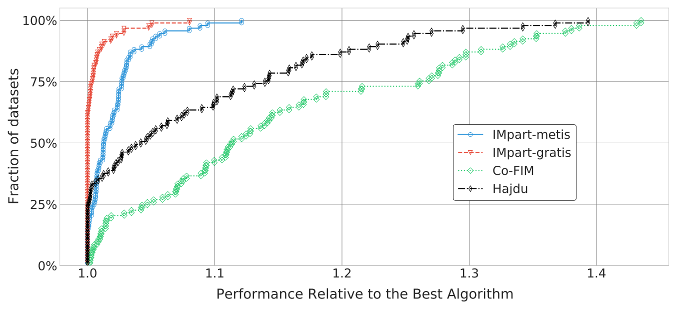


Fig. 1: Performance chart showing the relative *quality* of solutions achieved by two of our `IMpart` implementations and two state-of-the-art community-based IM solutions. Here, quality is measured by the expected influence of the solution computed. Each method was run on 14 inputs for a range of k values: $\{10, 20, 30, 40, 50, 75, 100, 200, 400, 800\}$. The closer a tool is to the y-axis, and the longer it stays along the y-axis, the more superior it is.

Contributions: In this paper, we propose a new parallel framework, `IMpart`, to exploit partitioning and clustering to approximate IM. We first partition a given graph using partitioning, clustering or cluster-aware partitioning, and then compute the seed set from the partitions, using greedy hill-climbing. Under a partitioning assumption, we provide approximation guarantees and other provable properties of the solution. Our algorithm is parallel and our implementations support shared and distributed memory systems. We conducted an extensive empirical evaluation of `IMpart` on numerous real-world and synthetic graph inputs. Our results demonstrate that `IMpart` achieves two to three orders of magnitude speedup over a state-of-the-art multithreaded `GHC` implementation with negligible loss in quality (§ V). For instance, on a modest-sized input, the partitioning-based shared memory implementation yields $4610\times$ speedup, reducing the runtime from 9h 36m to 7 seconds on 128 threads. We also demonstrate significantly better quality of solutions compared to state-of-the-art approaches that use community detection for accelerating IM. Fig. 1 shows a performance chart showing superiority of `IMpart` over other approaches.

II. Related Work

A. Influence Maximization

Domingos and Richardson [3] presented one of the first known formulations and a heuristic for influence maximization. The seminal work by Kempe *et al.* [4] formulated IM as an optimization problem, and showed that IM is NP-hard under two diffusion models, namely Independent Cascade (IC) and Linear Threshold (LT). The IC model uses a transmission probability $p(e)$ along each edge. The vertices that become active at time-step t have a single attempt at activating their neighbors at time step $t+1$. The LT model instead uses a threshold for activating vertices, i.e., vertex i becomes active when the sum of weights of its incident edges exceeds the threshold. Kempe *et al.* [4] also proved that the expected influence function is a monotone non-negative submodular function and leveraged its approximability cardinality constraints [16], to present a greedy hill-climbing (`GHC`) strategy that provides $1 - 1/e - \varepsilon$ approximation guarantee. Subsequently, several improvements to `GHC` were proposed [17]–[19]. Borgs *et al.* [20] developed an alternative to `GHC`, by introducing the concept of Reverse Influence Sampling (RIS) which is rooted on the idea that highly influential vertices will appear frequently in Random Reverse Reachable sets.

A central problem in these two approach is determining the sampling effort (θ) to provide the stochastic space in which to compute expected influence. While the existence of tight bounds for the sampling complexity of `GHC` is still an open question, the work of Tang *et al.* [21] built connections between θ and the approximation parameter ϵ. In particular, the IMM algorithm [21] leverages a practically efficient martingale strategy to determine the sampling effort.

Recently, parallel and scalable implementation of the IMM algorithm have been developed [2], [22], [23] for shared memory and distributed memory machines, as well as for multi-GPU systems. While they are generally faster than `GHC`, IMM requires that the diffusion process can be reformulated in reverse settings, and also limits reuse of samples across different (k, ϵ) experiments. `GHC` does not pose these two limitations. Furthermore, `GHC` has broader applicability to several other convex optimization applications [24]–[26]. The state-of-the-art in parallel `GHC` for IM is [25], which we use as the baseline for our experiments.

B. Community-based Influence Maximization

Real-world networks have known to exhibit community-based organization [15]. Consequently, several approaches have tried to exploit this community structure in order to efficiently identify the seeds. Wang *et al.* [8] present a diffusion-aware label propagation community detection algorithm to mine the top-k influential nodes in mobile social networks. The CIM method [9] uses a heat diffusion model alongside hierarchical clustering to classify nodes as "homeless" (outliers or hubs connecting different communities) or those belonging to communities for the seed selection process. INCIM [10] uses a two-step hierarchical approach, by combining a node's influence on a coarsened community graph and its local influence at each community level to aid in seed selection. Halappanavar *et al.* [11] allocate a seed budget to each community based on the sizes of the communities prior to computing seeds from each community. COFIM [12] uses a fast heuristic to greedily select seeds based on the number of distinct communities around the immediate neighborhood of a vertex. The method by Hajdu *et al.* [13] computes overlapping communities [27] in a preprocessing step, and uses those vertices that belong to multiple communities as its seeds. Open source implementations are available only for COFIM [12] and Hajdu *et al.* [13]. As they also represent recent works, we use these two tools as our state-of-the-art baselines for comparison.

The `IMpart` approach presented in this paper differs from the current state-of-the-art reviewed above in several ways. First, it takes a partitioning-based approach to accelerate influence maximization, making it more generic to the use of community detection or graph partitioning techniques. Partitioning also helps in distributing the problem space making it more amenable to parallel processing. Second, by using greedy hill climbing at the partition-scale, the quality of approximation is trivially maintained at the partition level. The challenge is in establishing the quality of approximation over the entire graph, for which we provide provable guarantees in §IV. Finally, our method is parallel and supports implementations on shared and distributed memory settings.

III. Influence Maximization Preliminaries

Let $G = (V, E, \omega)$ be a (di)graph where V is a set of vertices, E is a set of (ordered) pairs (i, j) such that $i, j \in V$, and ω is a set of non-negative edge weights representing the probability of vertex i influencing vertex j for an edge (i, j).

Definition III.1 (Influence Maximization (IM) Problem)**.** *Given a (di)graph G, an integer k, and a model of diffusion M, the Influence Maximization Problem is to identify a subset $S \subseteq V$ of size k such that the expected number of activated vertices $(\mathbb{E}[I_G(S)])$ is maximized when the diffusion process M starts from the vertices in S.*

The greedy hill climbing algorithm for IM by Kempe *et al.* provides an approximation guarantee of $1 - 1/e - \varepsilon$. An efficient parallelization scheme for this greedy hill-climbing algorithm was proposed by Minutoli *et al.* and is summarized in Algorithm 1. It starts by sampling the space of possible realizations of the diffusion process M over G and obtains a number (θ) of subgraphs G_i of G by retaining edges that triggered activation under M. The samples are subsequently used in seed selection (`GetNextSeed`) to construct an estimate of the expected influence for each remaining seed candidate (i.e., $v \in V \setminus S$). This estimation procedure is implemented as a Breadth First Search (BFS) over each sample, treating v as the root and counting the number of new activations over all samples. The algorithm has k iterations; with the i^{th} iteration selecting the i^{th} seed by greedily picking the next vertex with the largest marginal gain in expected influence.

Algorithm 1: `ParallelGHC`(G, k, θ, D): Parallel Greedy Hill Climbing for IM using Sampling

```
Data: $G = (V, E, w)$, $k$, $\theta$
Result: Seed set $S$ of $k$ vertices
// generate $\theta$ random samples
$\mathbb{S}_{all} \leftarrow \emptyset$
for $i \in [1, \theta]$ do in parallel
    $\mathcal{G}_i \leftarrow$ Generate a random subgraph of $G$ based on
      the diffusion model $D$ and add to $\mathbb{S}_{all}$
end
$S \leftarrow \emptyset$; // init seed set
while $|S| < k$ do
    $[s, gain] \leftarrow$GetNextSeed $(G(V, E), S, \mathbb{S}_{all})$
    $S \leftarrow S \cup \{s\}$
end
Return $S$
```

```
Function GetNextSeed ($G(V, E, \omega)$, $S$, $\mathbb{S}_{all}$):
    Initialize $c[v] \leftarrow 0, \forall v \in V$
    for each $v \in V \setminus S$ do in parallel
        for each $\mathcal{G}_i \in \mathbb{S}_{all}$ do
            $c[v] \leftarrow c[v] + (I_{\mathcal{G}_i}(S \cup \{v\} - I_{\mathcal{G}_i}(S))$
        end
    end
    $s \leftarrow \arg\max_{v \in V \setminus S} c[v]$
    Return $s$, $c[s]$
```

IV. Partitioning-based Approaches to Accelerate Influence Maximization

Our approach to accelerate IM takes a partitioned view. Consequently it's named `IMpart`, short for *Influence Maximization by partitioning*. Figure 2 illustrates the main workflow for `IMpart`. In what follows, we first describe the partitioning step (§IV-A), and then we describe our parallel greedy hill climbing algorithm that uses those partitions (§IV-B).

A. Partitioning for `IMpart`

Intuitively, by partitioning the network *a priori*, we are cutting the weak links that separate otherwise well-connected parts of the network. The hypothesis is that those cut edges seldom contribute to influence spread. This simple and yet powerful idea can be effective in decomposing a large problem instance into smaller subproblems (i.e., individual partitions) without significantly impacting solution quality (as will be shown in §V). However, the efficacy may depend on the quality of the partitioning.

As noted in Section II-B, past approaches to partitioning a network have heavily relied on community based structural information inherent in most real-world networks [28]. However, there are also classical graph partitioning methods [14], [29] that are yet to be explored for IM. While both these problems divide the vertex set into a disjoint set of partitions, the objectives are different: With community detection, the goal is to identify a set of tightly-knit vertex groups (or communities) that are not as strongly connected to the rest of the network by using a clustering objective such as modularity [15] to partition the graph into disjoint communities. Depending on the input network, the number of communities and their individual sizes can vary significantly—potentially posing challenges to load balancing in a parallel setting.

On the other hand, graph partitioning aims to partition a graph into a defined number (say m) of vertex partitions, each with roughly equal load (measured by the sum of weights of the respective edge or vertex sets in a partition). The optimization objective function here is to minimize the number of edges cut between partitions. Under a weighted setting, the edge cuts are likely to include the weaker links of transmission.

In this paper, we present a unified framework, `IMpart`, to explore both graph partitioning and community detection to generate the partitions for IM. Given an input graph G,

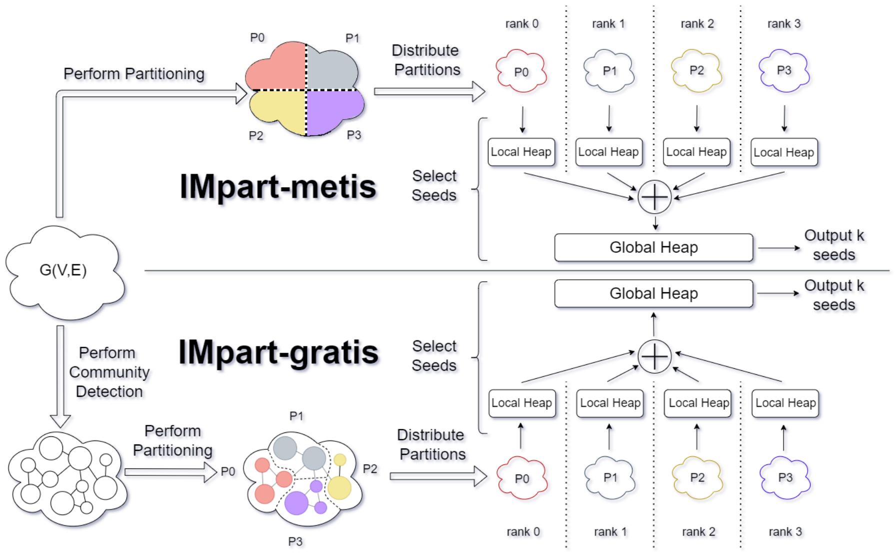


Fig. 2: `IMpart`: A partitioning-based approach to accelerate Influence Maximization. The example shows four partitions being processed on four MPI ranks. However, in general, this can be m partitions processed across on p processes, where $m \geq p$.

`IMpart` first partitions the set of vertices into a disjoint set of m partitions $\{P_0, P_1, \ldots, P_{m-1}\}$, and subsequently processes the partitions across p processes in a distributed manner. Here, each P_i corresponds to a subgraph of G, such that the vertices over all P_i cover V and no two vertex sets from any two partitions intersect. A conceptual example of the workflow is shown in Figure 2.

`IMpart` supports two implementations, namely, `IMpart-metis` and `IMpart-gratis`.

a) **`IMpart-metis`:** This implementation uses a graph partitioning tool to generate the partitions in the first step. Since we use the `METIS` [14] shared memory parallel partitioner, we call this implementation `IMpart-metis`. However, the framework allows the use of any graph partitioner of choice [30]–[32]. Since the time for seed selection depends on the number of edges, we partition the graph into m parts such that the number of *edges* per partition is roughly balanced. This is achieved by setting the vertex degree of each node $v \in V$ as its weight. `METIS` then uses this information to generate an edge-balanced partitioning.

b) **`IMpart-gratis`:** As noted earlier, direct use of community detection for partitioning may give rise to potential imbalances across the partitions and also little control over the number of partitions. However community detection has the advantage of identifying tightly-knit groups of vertices which has the advantage of keeping highly related vertices (i.e., those that can mutually influence one another) within the same partition. To address this tradeoff, we developed a hybrid strategy that performs community-aware partitioning. We call the resulting community approach, `IMpart-gratis`, named after `Grappolo`-based community detection [33] followed by `Metis`-based partitioning [14]. The framework itself is more generic to allow use of other community detection tools. We chose `Grappolo` because of its parallel support.

The major steps of `IMpart-gratis` are as follows (see lower half of Figure 2 for an illustration).

S1) We first detect communities in G. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_t\}$ denote the set of output communities. In most cases we expect $t \geq m$.

S2) Since the target is to generate m partitions, our next step groups the t communities into m partitions. This is achieved by first building a new "community graph" $G_c(V_c, E_c, \omega_c)$, where each community $C \in \mathcal{C}$ is represented as a vertex in V_c, and two communities C_i and C_j sharing e_{ij} (inter-cluster) edges between them will create an edge $e \in E_c$ between the two corresponding nodes with the edge weight e_{ij}. Every vertex V_c representing community $C \in \mathcal{C}$ is weighted to reflect the size of C.

S3) Subsequently, we run `METIS` partitioner on G_c to generate m partitions.

B. *The `IMpart` parallel approach*

Next, we describe the parallel algorithm for `IMpart`. We assume that the input graph has been partitioned by the partitioning step (Section IV-A) into m partitions, $\mathcal{P} = \{P_0, P_1, \ldots, P_{m-1}\}$. Let $G_P(V_P, E_P, \omega_P)$ be the subgraph induced by partition $P \in \mathcal{P}$. We use the notation $\preceq$ for subgraphs—i.e., $G_P \preceq G$.

Algorithm 2 shows the distributed memory parallel algorithm of `IMpart` (with multithreading enabled within each process). We use p to denote the number of processes, r to denote the local process rank, and $\mathcal{P}_r$ to denote the subset of partitions in $\mathcal{P}$ that rank r is responsible for. The partitions are loaded in a distributed manner such that each process gets approximately $\lceil \frac{m}{p} \rceil$ partitions. Let $G_r(V_r, E_r, \omega_r) \preceq G(V, E, \omega)$ denote the subgraph induced by $\mathcal{P}_r$. Each process r only loads its local subgraph $\mathcal{G}_r$. The key steps are as follows.

S1) The partitions are evenly loaded in a distributed manner across all processes, such that each process loads

Algorithm 2: `IMpart`$(G_r, k, \theta, M, \mathcal{P})$: Parallel partition-based Greedy Hill Climbing (at rank r)

```
Data: $G_r$: subgraph at rank $r$, $k$: no. seeds, $\theta$: no.
      samples, $M$: model, $\mathcal{P}$: set of $m$ partitions
Result: Seed set $S$ of $k$ vertices
$\mathcal{P}_r : \{P_{r*size}, \dots, P_{(r+1)*size-1}\}$, where $size = \lceil \frac{m}{p} \rceil$
Initialize samples $\mathbb{S}_r \leftarrow \emptyset$
for $i \in [1, \theta]$ do in parallel
    Generate a random sample from $\mathcal{G}_r$ based on
      model $M$ and add sample to set $\mathbb{S}_r$
end
Initialize $status[P] \leftarrow$ active, $\forall P \in \mathcal{P}_r$
Initialize an empty heap $\mathcal{H}_r$ of size $k$
repeat
    for $P \in \mathcal{P}_r$ do in parallel
        continue if $status[P] =$ terminated
        $[s_{new}, gain] \leftarrow GetNextSeed(\mathcal{G}_r, \mathcal{H}_r, \mathbb{S}_r)$
        if $GetMin(\mathcal{H}_r) < gain$ then
            $P_{min} \leftarrow \mathcal{H}_r.DeleteMin()$.partid
            $status[P_{min}] \leftarrow$ terminated
            $\mathcal{H}_r.push(gain, < s_{new}, P >)$
        end
        else
            $status[P] \leftarrow$ terminated
        end
    end
until no active partitions;
$S \leftarrow$ Allreduce $\mathcal{H}_{local}$ to the $k$ global best seeds
Return $S$
```

approximately the same number of partitions (and more specifically, the subgraphs induced by them). Using $\mathcal{P}_r$, each process constructs its local subgraph $\mathcal{G}_r$.

S2) Next, each process generates θ samples but only using subgraph $\mathcal{G}_r$. This step is multithreaded within each process. Let $\mathbb{S}_r$ be the resulting sample set.

S3) Each partition is assigned a flag to indicate it is *active* initially. The process also initializes an empty local heap $\mathcal{H}_r$ with maximum capacity k.

S4) Each process then starts multiple rounds until there are no more active partitions left in $\mathcal{P}_r$. At each round, all partitions that are still active are visited and using the `GetNextSeed(.)` function, the next best seed candidate (say, s_{new}) is nominated. However, for the nomination to succeed (i.e., to get inserted into $\mathcal{H}_r$), the contribution of s_{new} to the expected influence should exceed the smallest expected influence of any seed in $\mathcal{H}_r$; let us refer this minimum seed in the heap as s_{min}. If nomination is successful, then s_{min} is removed from $\mathcal{H}_r$ and s_{new} is inserted. Furthermore, the flag corresponding to partition (P_{min}) containing s_{min} is changed to *terminated*—effectively shutting it down from contributing any further seeds in the subsequent rounds. Locks are used to ensure thread-safe insertion into $\mathcal{H}_r$.

S5) In the final step, we perform a single `Allreduce` operation on the heaps to select the top k global best seeds from p individual local heaps.

For the shared memory-only implementation, the same algorithm applies with the exception that there is only one shared global heap that all threads use. Secondly, our implementation also supports nested thread parallelism to enable a group of threads to work on a single partition, while multiple partitions are concurrently being processed.

C. Algorithmic properties and guarantees

We now prove that `IMpart` computes a $1/m$-approximate solution with respect to the solution computed by `GHC`. An important assumption for the proof is that partitioning of the graph is done in such a manner that a vertex exerts *maximal* influence on its own partition compared to any other partition. While in theory there exist worst-cases that would break this assumption, we argue that this is not a limiting assumption and that for most practical cases we expect the assumption to hold if we use a high quality partitioner such as `METIS`. This is because a good graph partitioner (or a community detection objective such as modularity) would nearly always try to place a vertex in a partition where it shares most of its neighbors. Consequently, much of the vertex' influence is also likely to be concentrated locally in that partition. To test the assumption we conducted an experiment on an arbitrarily chosen set of inputs and compared the influence of the top seed chosen by `GHC` on its owning partition versus the seed's *maximum* influence on any of the m partitions. Results are shown in Fig. 3. We observed that the influence on the assigned partition is generally within 90% of the maximum.

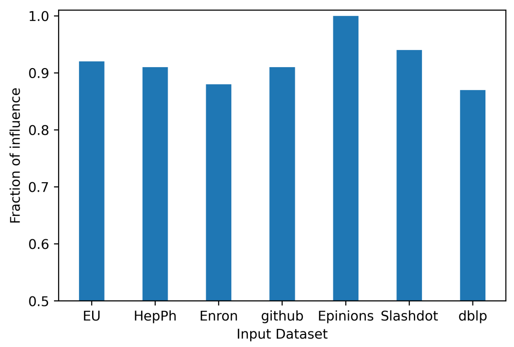


Fig. 3: The expected influence of the top seed by `GHC` on its local partition, as a fraction of its maximum influence on any partition.

We use $\sigma_H(u)$ to denote the expected influence of a vertex u on any graph H. Now, let us consider the input graph $G = (V, E)$ with m partitions ($\mathcal{P}$) as partitioned by `IMpart`. Note that each partition in $\mathcal{P}$ represents a subgraph in G. The partitioning assumption for $\mathcal{P}$ is such that for any vertex u located in partition $p \in \mathcal{P}$, $\sigma_p(u) \geq \sigma_q(u)$, where $q \in \mathcal{P} \setminus \{p\}$ (i.e., maximal local influence). In what follows, we prove the approximation guarantee for `IMpart`.

Lemma IV.1. *Given input $G = (V, E)$, let S^1_{imp} and S^1_{ghc} be the first seeds computed by `IMpart` and `GHC`, respectively. Then $\sigma_G(S^1_{imp}) \geq \sigma_G(S^1_{ghc})/m$.*

Proof. We consider the non-trivial case where $S^1_{imp} \neq S^1_{ghc}$. Let seeds S^1_{imp} and S^1_{ghc} come from the partitions p and q respectively. Irrespective of whether $p = q$, $\sigma_p(S^1_{imp}) \geq \sigma_q(S^1_{ghc})$, as otherwise `IMpart` would have also selected S^1_{ghc}. However, it is possible that $\sigma_G(S^1_{imp}) < \sigma_G(S^1_{ghc})$ if `GHC` selects a seed that has additional influence on the other partitions while the influence of `IMpart`'s seed is limited to its partition p. This is illustrated through a worst-case example in Figure 4, where the influence of S^1_{imp} on its local partition p (denoted by c) is also equal to its influence on the entire graph G; whereas S^1_{ghc} has influence on all partitions including its local partition q. However, because of the partitioning assumption for maximal local influence, the overall influence of S^1_{ghc} on G cannot exceed $m \times c$. Thus, $\sigma_G(S^1_{imp}) \geq \sigma_G(S^1_{ghc})/m$. □

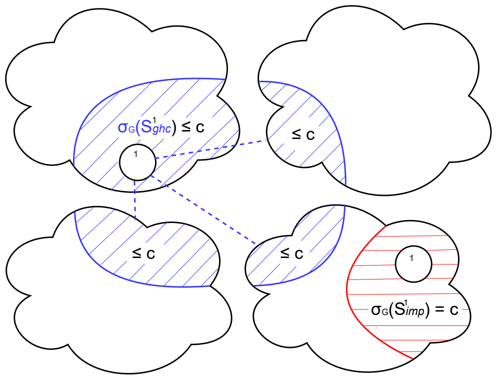


Fig. 4: Worst-case scenario for `IMpart` during the first seed selection. `IMpart` selects S^1_{imp} which has influence only within partition p, while `GHC` selects S^1_{ghc} with influence on multiple partitions.

Lemma IV.2. *Given a subgraph $G_P = (V_P, E_P)$ corresponding to a partition $P \in \mathcal{P}$, the solution $\mathbb{S}_P$ (local set of seeds) computed by `IMpart` is submodular.*

Proof. Follows directly from the fact that `IMpart` calls `GHC` on G_P, and therefore the $(1 - 1/e - \varepsilon)$-approximation and submodular property provided by Kempe *et al.* [4] on G also extend to G_P. □

Lemma IV.3. *Let $\overline{G} = \{G_1 \cup G_2 \ldots \cup G_m\}$ represent the graph from the union of m partitions. The solution $\mathbb{S}_{\overline{G}}$ computed by `IMpart` is submodular.*

Proof. Follows from Lemma IV.2, and the fact that seed selection in `IMpart` picks the best from m partitions at each step, and that it uses `GHC` within each partition to compute marginal gains of influence. □

Note that $\overline{G}$ represents the original graph G with its inter-partition edges removed.

Theorem IV.4. *Consider input G with m partitions. Let $\mathbb{S}_{imp}$ and $\mathbb{S}_{ghc}$ be the solutions computed by `IMpart` and `GHC`, respectively. The condition $\sigma_G(\mathbb{S}_{imp}) \geq \sigma_G(\mathbb{S}_{ghc})/m$ holds.*

Proof. Intuitively, the proof works by showing that for every seed x selected by `GHC` there is a corresponding vertex a that is selected by `IMpart` such that $\sigma_G(a) \geq \sigma_G(x)/m$.
Base case: Let x_1 and a_1 denote the first seeds selected by `GHC` and `IMpart` respectively. The condition that $\sigma_G(a_1) \geq \sigma_G(x_1)/m$ for m partitions holds from Lemma IV.1.
Step $k-1$: Assume that the condition holds true for the first $k-1$ seeds selected by both schemes. Due to submodularity, we know that: for `GHC`, $\sigma_G(x_1) \geq \sigma_G(x_2) \ldots \geq \sigma_G(x_{k-1})$, and for `IMpart`, $\sigma_{\overline{G}}(a_1) \geq \sigma_{\overline{G}}(a_2) \ldots \geq \sigma_{\overline{G}}(a_{k-1})$. From Lemma IV.3, the following also hold: $\sigma_{\overline{G}}(a_1) \geq \sigma_G(x_1)/m$, $\sigma_{\overline{G}}(a_2) \geq \sigma_G(x_2)/m, \ldots \sigma_{\overline{G}}(a_{k-1}) \geq \sigma_G(x_{k-1})/m$.
Step k: Let x_k and a_k denote the k^{th} seeds selected by `GHC` and `IMpart` respectively. While `GHC` selects x_k based on the marginal gain w.r.t. $x \ldots x_{k-1}$ in G, `IMpart` selects a_k based on the marginal gain w.r.t. $a \ldots a_{k-1}$ in $\overline{G}$. Therefore, it does not matter from which partition a_k is selected from, it will be guaranteed that $\sigma_{\overline{G}}(a_k) \geq \sigma_G(x_k)/m$, whether a_k and x_k are the same vertex or not.

Since the approximation ratio holds for every single seed chosen by `IMpart` w.r.t. `GHC`, the summation of expected influence over the seed set is: $\sigma_G(\mathbb{S}_I) \geq \sigma_G(\mathbb{S}_G)/m$. Therefore, `IMpart` computes $1/m$-approximate solutions w.r.t. `GHC`. □

Theorem IV.5. *Let `IMpart` partition an input graph G into m partitions, to output k seeds. Then, the number of rounds taken to terminate by the `IMpart` algorithm (Algorithm 2) is at most $(k + 1)$.*

Proof. The number of rounds refers to the `repeat-until` loop in Algorithm 2. Recall that the size of the local heap $\mathcal{H}_{local}$ at each process is k. Let m' denote the number of local partitions held by a process. If $m' \leq k$, then the algorithm would take at most $\lceil \frac{k}{m'} \rceil$ rounds. If $m' > k$, then after the first round, $(m' - k)$ local partitions will get terminated (because $|\mathcal{H}_{local}|$ is limited to k). At each subsequent round, at least one of the remaining partitions will get terminated. This upper bounds the number of rounds to $(k + 1)$. □

Complexity analysis: For $G(V, E)$, in the standard `GHC` algorithm, each edge in E can be present in at most θ samples. The same is applicable to the `IMpart` framework irrespective of how G is partitioned—except that the total number of edges being considered can only be less than in G. As a result the sum of sample sizes across partitions is upper-bounded by the sample size of `GHC`.

Next, we analyze the runtime complexity of parallel `IMpart` (Algorithm 2). Let p and t denote the number of processes and number of threads per process respectively. Thus, the number of cores used is $(p \cdot t)$. For the purpose of analysis, we assume $m \geq p$. In Algorithm 2, the cost of sampling is the product of the number of samples and the generation cost per sample, i.e., $O(\frac{\theta \cdot |E|}{p \cdot t})$. As for seed selection, the number of rounds is bounded by $(k + 1)$ (by Theorem IV.5). Within each round, $\sim\frac{m}{p}$ partitions are processed using t threads per process. For each part, BFS takes $O(\frac{|E|}{m})$ time, and there are θ samples, each with $|V|$

worst-case number of BFS roots. This yields an overall time complexity for seed selection as $O(\frac{k \cdot \theta \cdot |E| \cdot |V|}{m \cdot p \cdot t})$. The time to reduce the global heap is $O(\tau \log p + \mu k)$, where τ and μ are network latency and bandwidth. Therefore, the overall time is dominated by the cost of seed selection.

V. Experimental Results

A. Experimental Setup

Test platforms: All shared memory experiments were conducted on a 128-core system with AMD EPYC 7502 CPUs (2.5 GHz), and 256 GB of octa-channel DDR4-3200 memory. The distributed memory experiments were conducted on the Haswell partition of the NERSC Cori supercomputer, which is a 2,388-node Cray® XC40™ machine with the Cray® XC™ series interconnect (Cray® Aries™ with Dragonfly topology). Each node has two sockets, and a socket is equipped with Intel® Xeon™ E5-2698v3 CPUs (16 cores at 2.3 GHz), 128 GB DDR4 2133 MHz memory, 40 MB L3 cache/socket.

Input data: We use a total of 18 input graphs for evaluation, as summarized in Table I. We use: i) 14 real-world inputs (from SNAP [34]) to assess quality and performance on the shared memory platforms; and ii) in addition, 4 synthetic inputs to evaluate performance on distributed memory platforms. These graphs were generated using the GTgraph synthetic graph generator suite [35] with power-law degree distributions and small-world characteristics according to the R-MAT graph model [36].

TABLE I: *Input datasets.* Δ is the maximum degree, and Column 5 lists the standard deviation of the vertex degrees.

Input	#Vertices	#Edges	Δ	Std Dev
Small Instances for Analysis Against Hill-Climbing				
AstroPh	18,772	198,110	504	30.6
musae_facebook	22,470	171,002	709	26.4
CondMat	23,133	93,497	281	10.6
HepTh	27,770	352,807	2468	45.3
EU_Deezer	28,281	92,752	172	7.9
HepPh	34,546	421,578	846	30.9
email_enron	36,692	183,831	1383	36.1
musae_github	37,700	289,003	9458	80.8
RO_Deezer	41,773	125,826	112	5.5
HU_Deezer	47,538	222,887	112	7.4
HR_Deezer	54,573	498,202	420	17.9
Epinions	75,879	508,837	3079	52.7
Slashdot	77,360	905,468	5048	73.2
DBLP	317,080	1,049,866	343	10.1
Large Instances of Synthetic R-MAT graphs				
SynGraph1	0.52E+06	0.34E+08	4880	136.4
SynGraph2	1.05E+06	0.69E+08	6138	141.6
SynGraph3	2.10E+06	1.39E+08	7550	147.2
SynGraph4	4.19E+06	2.77E+08	9808	152.5

Software and tools: `IMpart` was implemented using MPI+OpenMP programming model. The results reported from our experiment were obtained by compiling our implementation with GCC 11.2.0 with `-O3` and `-mtune=native` compilation flags and using Openmpi 4.1.2 in our distributed memory experiments. We consider three variants of `IMpart`:

- **`IMpart-metis`:** Partitions are obtained by using the `METIS` partitioner before seed selection.
- **`IMpart-gratis`:** A community-based coarsened graph is obtained by using `Grappolo` on the input graph, which is then partitioned by `METIS`.
- **`IMpart-grappolo`:** We also include results from treating the community outputs by `Grappolo` as the partitions.

For experiments in this paper, we used the IC diffusion model, as it is more computationally challenging relative to LT and has wider use in applications [25]. The edge probabilities were drawn from a normal distribution with a mean value of 0.5 and variance of 0.5, resulting in values in the range of [0,1].

B. Qualitative evaluation

1) Comparison against state-of-the-art IM tools: We compared `IMpart` against the state-of-the-art parallel implementation of classical `GHC` [25], as well as two other recent community-based IM tools: Co-FIM [12], and Hajdu *et al.* [13]. For these experiments, we set $k = 100$. Quality of the seeds computed is quantified using the expected number of activations at the end of the diffusion process. Our experiments measured the average number of activations obtained from five simulations. For the partitioning-based approaches, the number of partitions was varied from four to 64 for the smaller inputs, and up to 256 for the three medium-sized inputs.

Fig. 5 presents the results of this comparative study. The results are presented as a percentage gain with respect to the `GHC` baseline of [25]. The results show that the quality of influence achieved by `IMpart` is highly comparable, if not better (green cells) than the `GHC` baseline [25]—with over 8% improvement in some cases. In cases where `IMpart` implementations degrade quality (red cells) relative to the `GHC` baseline, the loss is mostly negligible, with only a few cases leading to about 2% (for `IMpart-grappolo`) or 5% (for `IMpart-metis`) or 3% (for `IMpart-gratis`). In comparison, the other two community detection based IM methods—CoFIM and Hajdu *et al.*—consistently show significant loss in quality (up to 21%) compared to the `GHC` baseline. These results demonstrate the qualitative superiority of `IMpart`. Among the `IMpart` variants, we observe that all three implementations yield comparable quality, with `IMpart-gratis` marginally outperforming the other two.

2) Effect of seed set size on quality: Next, we study the impact of the number of seeds on quality. We varied k from 10 to 800, and ran `IMpart-metis` and `IMpart-gratis`, keeping the number of partitions at 64. With four methods running on ten values of k and 14 inputs, we evaluate a total of 140 executions *per* method. We also ran the two other community detection methods for comparison. Fig. 1 summarizes the results in the form of a performance chart. This chart denotes the fraction of inputs (y-axis) over the 140 executions for each method, for which the performance of a method deviates from the best performing method (x-axis) at that level. The results show that `IMpart-metis` and `IMpart-gratis` are clearly better than the other two tools by a significant margin, with `IMpart-gratis` outperforming all other methods in well over 75% of the input cases.

			IMpart-metis				IMpart-gratis					
Input	HC E[σ(S)]	IMpart-grappolo	Metis-4	Metis-16	Metis-64	Metis-256	Gratis-4	Gratis-16	Gratis-64	Gratis-256	CoFIM	Hadju et. al
AstroPh	15,591	0.31%	0.87%	0.40%	-0.30%		0.61%	0.37%	0.19%		-1.69%	-1.30%
facebook	16,058	4.14%	5.43%	4.46%	2.96%		5.61%	4.73%	4.13%		-0.86%	-1.91%
CondMat	15,685	0.19%	1.17%	0.33%	-1.23%		1.59%	-0.27%	-0.38%		-4.65%	-2.58%
HepTh	22,667	3.75%	4.07%	3.87%	3.16%		4.01%	3.62%	3.55%		1.47%	0.29%
EU	14,732	5.65%	8.28%	6.07%	2.84%		8.22%	6.19%	4.41%		-7.51%	-4.50%
HepPh	19,353	3.09%	3.61%	2.52%	1.00%		3.72%	2.12%	0.86%		-20.42%	-3.90%
Enron	24,817	0.66%	1.28%	0.71%	-3.50%		1.32%	0.94%	0.67%		-1.55%	-0.85%
github	31,666	0.07%	0.13%	-0.07%	-0.02%		0.14%	0.00%	0.11%		-0.16%	0.04%
RO	20,142	-2.31%	-0.57%	-2.42%	-5.41%		-0.39%	-1.55%	-3.23%		-21.61%	-7.86%
HU	31,553	0.23%	1.41%	0.71%	-2.84%		2.08%	-0.31%	-0.12%		-14.79%	-5.93%
HR	46,584	1.09%	1.36%	0.74%	0.39%		1.29%	1.02%	0.88%		-4.43%	-1.13%
Epinions	31,684	-0.74%	-0.14%	-0.86%	-0.80%	-1.20%	-0.22%	-0.54%	-0.76%	-0.70%	-1.12%	-0.59%
Slashdot	56,649	-0.15%	-0.08%	-0.26%	-0.28%	-0.35%	-0.03%	-0.11%	-0.27%	-0.35%	-0.38%	0.00%
dblp	156,207	0.23%	2.60%	1.11%	-0.80%	-1.46%	1.95%	0.82%	-0.02%	0.10%	-7.28%	-1.33%

Fig. 5: Qualitative evaluation of IMpart implementations compared to the state-of-the-art parallel GHC implementation (column 2), and two other community detection based methods (last two columns). All values are shown as a percentage net improvement over the baseline. Metis-m denotes the number of partitions m used, and similarly for IMpart-gratis. All experiments used $k = 100$.

In cases where it is not the best, it is still only within 1.1× away from the best performing method.

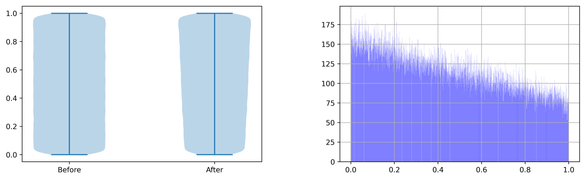


Fig. 6: *Edgecut analysis of DBLP when partitioned by METIS* : The left plot shows the distribution of weights over the edges before vs. after removal of edges post-partitioning. The plot on the right shows the histogram of the edges dropped over the probability values.

3) Edge-cut analysis: As IMpart discards inter-partition edges, it is effectively a way to gain performance by potentially trading off quality (by losing edge information) compared to the GHC baseline. Ideally, partitioning should remove edges that are less important for influence spread. To test this hypothesis, we examined the edge weight distribution before and after partitioning for the DBLP input. The results shown in Fig. 6 (left) validate this hypothesis with much of the larger weight distribution retained after edge removals. Moreover, the histogram of the dropped edges shows that more edges were removed from the lower probability spectrum, which explains the negligible loss in quality by IMpart in Fig. 5.

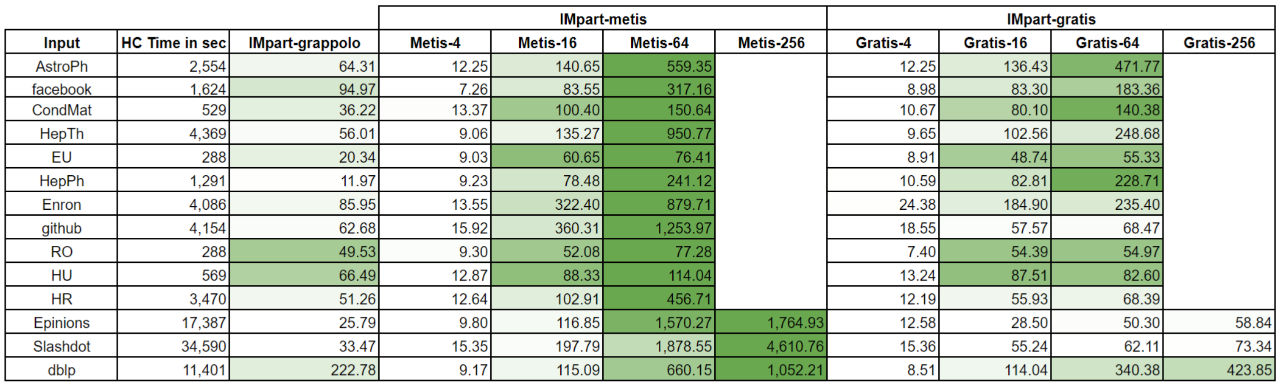
C. Performance evaluation

Next, we analyze performance of IMpart implementations. First we evaluate it on a 128-core shared memory platform. Fig. 7 shows the speedups achieved by IMpart over the state-of-the-art parallel GHC baseline [25]. All experiments were performed with $k = 100$.

The results show anywhere between one to three orders of magnitude performance improvement over the GHC baseline. Among the variants, IMpart-metis delivers the best speedups, followed by IMpart-gratis and IMpart-grappolo. These results demonstrate significant acceleration of time-to-solution. For instance, on Slashdot (73K nodes, 905K edges), our partitioning-based shared memory implementation yields 4,610× speedup, reducing the runtime from 9h 36m to 7 seconds on 128 threads. We also observe that the speedup generally increases with increasing number of partitions for both IMpart-metis and IMpart-gratis. We note that the cost of preprocessing time (partitioning, community detection) was low compared to the total time. For instance, to partition DBLP into 256 partitions, IMpart-metis and IMpart-gratis took 0.8 and 0.4 seconds respectively while the rest of the algorithm took 10 and 26 seconds respectively.

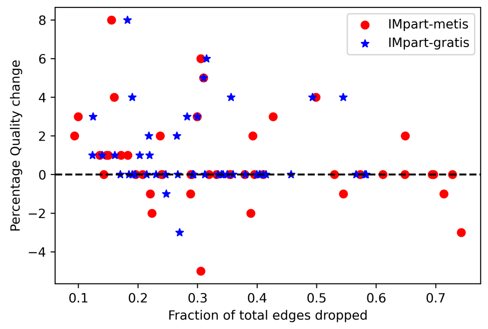
The above results also highlight a performance-quality trade-off between IMpart-metis and IMpart-gratis. While IMpart-gratis is better in quality (Fig. 5), IMpart-metis is better in performance (Fig. 7). To understand why this happens, we examined edge-cuts. In Fig. 8, we show the correlation between the fraction of edges dropped versus: a) the percent change in quality, and b) the speedup, for both implementations. Intuitively, with a larger edge-cut, speedups should improve (as there is less work during sampling). This is what we see in Fig. 8b, with IMpart-metis pruning more edges and hence delivering better performance. However, by removing edges we also run the risk of potentially degrading solution quality. While this is to some extent observable in Fig. 8a, it can be seen that the loss is slightly more for IMpart-metis.

To further understand the performance-quality trade-off, we examined examples (based on Fig. 7) where IMpart-metis provides significantly more speedup than IMpart-gratis for $m = 64$ and cases where the speedups were comparable. We observed that IMpart-metis tends to partition an input graph into roughly uniform sizes; while IMpart-gratis sometimes generates uneven partition workloads due to skewness in community sizes like in Epinions and Slashdot. As a result the seed selection algorithm on the larger partitions become bottlenecks. Whereas for examples like HepPh and DBLP, the IMpart-gratis partitions are more balanced and as a result show comparable speedups when compared against IMpart-metis.

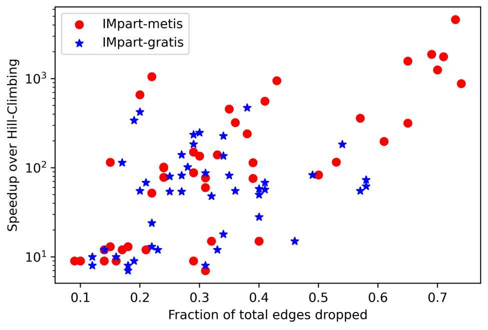


			IMpart-metis				IMpart-gratis			
Input	HC Time in sec	IMpart-grappolo	Metis-4	Metis-16	Metis-64	Metis-256	Gratis-4	Gratis-16	Gratis-64	Gratis-256
AstroPh	2,554	64.31	12.25	140.65	559.35		12.25	136.43	471.77	
facebook	1,624	94.97	7.26	83.55	317.16		8.98	83.30	183.36	
CondMat	529	36.22	13.37	100.40	150.64		10.67	80.10	140.38	
HepTh	4,369	56.01	9.06	135.27	950.77		9.65	102.56	248.68	
EU	288	20.34	9.03	60.65	76.41		8.91	48.74	55.33	
HepPh	1,291	11.97	9.23	78.48	241.12		10.59	82.81	228.71	
Enron	4,086	85.95	13.55	322.40	879.71		24.38	184.90	235.40	
github	4,154	62.68	15.92	360.31	1,253.97		18.55	57.57	68.47	
RO	288	49.53	9.30	52.08	77.28		7.40	54.39	54.97	
HU	569	66.49	12.87	88.33	114.04		13.24	87.51	82.60	
HR	3,470	51.26	12.64	102.91	456.71		12.19	55.93	68.39	
Epinions	17,387	25.79	9.80	116.85	1,570.27	1,764.93	12.58	28.50	50.30	58.84
Slashdot	34,590	33.47	15.35	197.79	1,878.55	4,610.76	15.36	55.24	62.11	73.34
dblp	11,401	222.78	9.17	115.09	660.15	1,052.21	8.51	114.04	340.38	423.85

Fig. 7: Performance speedups achieved by the `IMpart` implementations over the state-of-the-art parallel `GHC` baseline [25] on a shared memory machine with 128 cores.



(a) Edge-cut vs Quality



(b) Edge-cut vs Speedup

Fig. 8: *Effect of edge-cuts on quality (part a) and performance speedup (part b).* Each point corresponds to an execution with a unique [method, input, no. partitions] combination.

D. `IMpart-metis`: Distributed executions

We present scaling results with up to 8 nodes by fixing k as 100. The weak scaling results are shown in Table II. Results show near-perfect weak scaling where with doubling of processes and doubling of partitions (graph size), parallel runtime is maintained.

We also investigated strong scaling for `IMpart-metis`. Table III shows a strong scaling study for Orkut-group ($|V| = 8.7M, |E| = 327M$) partitioned into 4096 pieces by `METIS`. We observe linear scaling as we vary the number of cores from 32 to 128. Subsequently, no performance improvement is observed because of reduced work per process.

TABLE II: Weak scaling on GTgraph inputs.

Input	#parts	#Processes (#cores)	Exec. time (s)
SynGraph1	512	1 (32)	38.98
SynGraph2	1024	2 (64)	37.44
SynGraph3	2048	4 (128)	32.68
SynGraph4	4096	8 (256)	35.77

TABLE III: Strong scaling on Orkut-group dataset.

Input	#Processes (#cores)	Execution time (s)
Orkut_group	1 (32)	327.29
#Vertices = 8.7M	2 (64)	186.19
#Edges = 327M	4 (128)	91.69
#partitions = 4096	8 (256)	130.99
(`METIS`)	16 (512)	100.44

VI. Conclusion

We introduced a partitioning-based approach (`IMpart`) to accelerate and scale influence maximization (IM) on shared and distributed memory systems. We demonstrated significant speedups, reducing runtime from 10 hours to 7 seconds without noticeable loss in the quality of solution. While our approximation bound of $1/m$ for m partitions is a loose bound, it provides insight for careful partitioning of the graph to minimize the impact of information loss from cut edges. Our empirical results corroborate this observation and demonstrate superior performance over state-of-the-art methods.

In our future work, we intend to design a scalable distributed framework for IM that is not limited by the loss of information for performance improvements via theoretically sound sparsification and sketching techniques. We will also study the impact of partitioning enforced from requirements such as fairness, information privacy and memory-scaling or as a preprocessing step to induce a vertex ordering that improves memory latency of the distributed IM framework. Given its importance, we believe that our work will advance not only algorithmic development but also wider adoption of IM in diverse applications and data science pipelines.

Acknowledgement

The research was in part supported by NSF awards CCF 1919122 and OAC 1910213.

References

[1] P. Domingos, "Mining social networks for viral marketing," *IEEE Intelligent Systems*, vol. 20, no. 1, pp. 80–82, 2005.

[2] M. Minutoli, M. Halappanavar, A. Kalyanaraman, A. Sathanur, R. Mcclure, and J. McDermott, "Fast and scalable implementations of influence maximization algorithms," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, 2019, pp. 1–12.

[3] P. Domingos and M. Richardson, "Mining the network value of customers," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 2001, pp. 57–66.

[4] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 137–146.

[5] X. Li, X. Cheng, S. Su, and C. Sun, "Community-based seeds selection algorithm for location aware influence maximization," *Neurocomputing*, vol. 275, pp. 1601–1613, 2018.

[6] S. Kundu, C. Murthy, and S. K. Pal, "A new centrality measure for influence maximization in social networks," in *International conference on pattern recognition and machine intelligence*, Springer, 2011, pp. 242–247.

[7] S. K. Pal, S. Kundu, and C. Murthy, "Centrality measures, upper bound, and influence maximization in large scale directed social networks," *Fundamenta Informaticae*, vol. 130, no. 3, pp. 317–342, 2014.

[8] Y. Wang, G. Cong, G. Song, and K. Xie, "Community-based greedy algorithm for mining top-k influential nodes in mobile social networks," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10, Washington, DC, USA: Association for Computing Machinery, 2010, pp. 1039–1048. DOI: 10.1145/1835804.1835935.

[9] Y.-C. Chen, W.-Y. Zhu, W.-C. Peng, W.-C. Lee, and S.-Y. Lee, "CIM: Community-based influence maximization in social networks," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 5, no. 2, pp. 1–31, 2014.

[10] A. Bozorgi, H. Haghighi, M. S. Zahedi, and M. Rezvani, "INCIM: A community-based algorithm for influence maximization problem under the linear threshold model," *Information Processing & Management*, vol. 52, no. 6, pp. 1188–1199, 2016.

[11] M. Halappanavar, A. V. Sathanur, and A. K. Nandi, "Accelerating the mining of influential nodes in complex networks through community detection," in *Proceedings of the ACM International Conference on Computing Frontiers*, ser. CF '16, Como, Italy: Association for Computing Machinery, 2016, pp. 64–71, ISBN: 9781450341288. DOI: 10.1145/2903150.2903181. [Online]. Available: https://doi.org/10.1145/2903150.2903181.

[12] J. Shang, S. Zhou, X. Li, L. Liu, and H. Wu, "CoFIM: A community-based framework for influence maximization on large-scale networks," *Knowledge-Based Systems*, vol. 117, pp. 88–100, 2017.

[13] L. Hajdu, M. Krész, and A. Bóta, "Evaluating the role of community detection in improving influence maximization heuristics," *Social Network Analysis and Mining*, vol. 11, no. 1, pp. 1–11, 2021.

[14] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed computing*, vol. 48, no. 1, pp. 96–129, 1998.

[15] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[16] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

[17] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 420–429.

[18] A. Goyal, W. Lu, and L. V. Lakshmanan, "CELF++: Optimizing the greedy algorithm for influence maximization in social networks," in *Proceedings of the 20th international conference companion on World wide web*, 2011, pp. 47–48.

[19] G. Göktürk and K. Kaya, "Boosting parallel influence-maximization kernels for undirected networks with fusing and vectorization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1001–1013, 2020.

[20] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, SIAM, 2014, pp. 946–957.

[21] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1539–1554.

[22] M. Minutoli, M. Drocco, M. Halappanavar, A. Tumeo, and A. Kalyanaraman, "CuRipples: Influence maximization on multi-gpu systems," in *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–11.

[23] S. Shahrouz, S. Salehkaleybar, and M. Hashemi, "Gim: Gpu accelerated ris-based influence maximization algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2386–2399, 2021.

[24] J. Tang, X. Tang, A. Lim, K. Han, C. Li, and J. Yuan, "Revisiting modified greedy algorithm for monotone submodular maximization with a knapsack constraint," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 1, pp. 1–22, 2021.

[25] M. Minutoli, P. Sambaturu, M. Halappanavar, A. Tumeo, A. Kalyananaraman, and A. Vullikanti, "PREEMPT: Scalable epidemic interventions using submodular optimization on multi-gpu systems," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 2020, pp. 1–15.

[26] Y. Liu, E. K. Chong, A. Pezeshki, and Z. Zhang, "Submodular optimization problems and greedy strategies: A survey," *Discrete Event Dynamic Systems*, vol. 30, no. 3, pp. 381–412, 2020.

[27] J. Xie, B. K. Szymanski, and X. Liu, "SLPA: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process," in *2011 ieee 11th international conference on data mining workshops*, IEEE, 2011, pp. 344–349.

[28] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026 113, 2004.

[29] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, IEEE, 2006, 10–pp.

[30] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, "High-quality hypergraph partitioning," *ACM J. Exp. Algorithmics*, Mar. 2022. DOI: 10.1145/

3529090. [Online]. Available: https://doi.org/10.1145/3529090.

[31] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2625–2638, 2017.

[32] "Partitioning tool for hypergraphs (patoh)," in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA: Springer US, 2011, pp. 1487–1487, ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_2197. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_2197.

[33] H. Lu, M. Halappanavar, and A. Kalyanaraman, "Parallel heuristics for scalable community detection," *Parallel Computing*, vol. 47, pp. 19–37, 2015.

[34] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*, http://snap.stanford.edu/data, Jun. 2014.

[35] D. A. Bader and K. Madduri, "GTgraph: A synthetic graph generator suite," *Atlanta, GA, February*, vol. 38, 2006.

[36] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *Proceedings of the 2004 SIAM International Conference on Data Mining*, SIAM, 2004, pp. 442–446.