TIG Influence Maximisation Challenge

TIG Labs

August, 2025

1 Impact: Practical and Scientific

Understanding how ideas, behaviors, and innovations spread through social networks is a classic challenge, studied across fields from epidemiology to economics. The problem of influence maximization, formally introduced to optimize viral marketing campaigns, tackles this challenge directly by asking: who are the most influential individuals in a network? Identifying this small set of key nodes can trigger a large-scale cascade of adoption, whether it's for a new product, a political message, or a public health initiative[1].

The computational challenge of influence maximization is amplified by the scale of modern networks, which often contain millions of users and billions of connections, rendering exhaustive search methods intractable. This has necessitated the development of scalable and theoretically-grounded algorithms that combine techniques from graph theory, probability, and combinatorial optimization [1]. Consequently, influence maximization has become a foundational problem in computational social science and network science. Its applications shape crucial strategies in everything from viral marketing and political campaigning to epidemiological modeling and the dissemination of critical public health information. Despite significant progress, the field continues to evolve, with ongoing research driven by the need for algorithms that can handle the massive scale and dynamic nature of real-world networks.

2 Problem Description and Formulation

Influence maximization is the problem of selecting a set of initial "seed" users in a social network to maximize the expected spread of influence (e.g., word-of-mouth effect) through the network [1]. Formally, we are given a directed graph $G = (V, E)$ representing a social network, where each node corresponds to a user and a directed edge $(i, j) \in E$ indicates that user i can influence user j (for example, via a "follow" relationship). A stochastic diffusion process (influence model) runs on this graph, where active users may activate their neighbors over time.

In the widely used *Independent Cascade* (IC) model, each directed edge $e = (i, j)$ is associated with an activation probability $p_e \in [0, 1]$ (commonly $p_e = 0.01$ in experiments). If user i becomes active, then for each outgoing edge (i, j), user j becomes active with probability p_e, independently of other neighbors. Once a user is activated, they remain active, and the process continues until no further activations occur.

At time $t = 0$, we may select an initial seed set $S \subseteq V$ subject to a resource constraint. In the standard formulation, this is a *cardinality constraint*:

$$|S| \leq k,$$

where k is the budget for the number of seeds (e.g., $k = 50$ or $k = 100$). The objective is to choose S that maximizes the expected number of activated users at the end of the diffusion process. Letting $f(S)$ denote the expected final spread from seed set S, the optimization problem is:

$$S^* = \arg\max_{|S| \leq k} f(S).$$

This is an NP-hard combinatorial optimization problem even under simple diffusion models such as IC or the *Linear Threshold* (LT) model [1].

Knapsack Constraint Variant

In many real-world scenarios, activating a user has an associated cost, such as the monetary incentive required to participate in a marketing campaign. Let $w_i > 0$ denote the cost (or weight) of selecting user $i \in V$, and let $W > 0$ be the total available budget. The seed set S must then satisfy the *knapsack constraint*:

$$\sum_{i \in S} w_i \leq W.$$

The optimization problem becomes:

$$S^* = \arg \max_{\sum_{i \in S} w_i \leq W} f(S).$$

This generalization is strictly harder than the cardinality case: the classic greedy algorithm for submodular maximization no longer guarantees a constant-factor approximation, and more sophisticated algorithms are required [2, 3]. The knapsack variant better reflects industrial settings where resources are limited and influencer costs are heterogeneous, providing a richer and more challenging benchmark for algorithm design. Weights can be assigned in various ways: uniformly at random, from a power-law distribution, or adversarially proportional to degree to mislead degree-based and greedy heuristics.

The total available budget is usually set as a fraction of the total weight. For example,

$$W = \frac{1}{10} \sum_i w_i.$$

The paper by Amanatidis et. al [4] considers a fixed budget of 10% or 15% of the total weight of the network, as well as testing over a varying budget ranging from 1% to 10% and up to 30%.

3 Baseline Algorithms

The most widely known approach for influence maximization is the greedy algorithm. Since $f(S)$ is monotone and submodular under both the IC and LT models, the greedy algorithm yields a $(1 - 1/e)$-approximation for the cardinality-constrained version [5, 1]. It iteratively selects the node with the highest marginal gain until the budget is exhausted.

Lazy Evaluation. To speed up greedy, a priority queue can store upper bounds on marginal gains. At each iteration, the element with the highest bound is evaluated; if its actual marginal gain still leads, it is selected, otherwise it is reinserted with the updated bound. This often reduces computation by an order of magnitude.

Simpler Heuristics. Degree-based selection (choosing top-k out-degree nodes) requires no influence simulation and is extremely fast, though typically less effective than greedy.

These baselines were proposed for the cardinality constraint version

4 Random Instance Generation

Each instance consists of a synthetic social network G together with influence-model parameters. Real-world social networks often exhibit power-law degree distributions, small-world properties, and community structure. Random graph models capable of replicating these properties—such as the R-MAT model [6]—are recommended. NVIDIA's CUDA-based `syngen_pyt` and RAPIDS `cuGraph` both support R-MAT generation and are GPU-friendly options.

R-MAT parameters (cuGraph). To instantiate R-MAT in `cuGraph`, the following parameters must be set.

- **Graph size.**
  - `scale` (s): number of vertices $n = 2^s$. Controls instance size.
  - `num_edges` (m) *or* `edge_factor` (f): total edges $m = f \cdot n$ (choose one). f sets average out-degree.
  - edge_factor for Graph 500 benchmark is $f = 16$ [7]
- **Initiator probabilities.**
  - `a`, `b`, `c` (with `d` $= 1 - a - b - c$): quadrant probabilities for the Kronecker initiator; control skew/community structure.
  - *Typical default:* $a = 0.57,\ b = 0.19,\ c = 0.19,\ d = 0.05$ used by the Graph 500 benchmark.
  - *Is this setting being visible to benchmarkers a problem?*
- **Directionality and symmetry.**
  - `directed`: `True` for directed graphs (natural for IC)
- **Vertex labeling and reproducibility.**
  - `scramble_vertex_ids`: randomize labels to eliminate quadrant locality artifacts (recommended: `True`).
  - `seed`: RNG seed for reproducibility (set per instance).
- **Quality controls.**
  - `allow_self_loops` / `allow_multi_edges`: whether to keep or drop self-loops / parallel edges (recommended: drop for IM benchmarks).
  - `renumber`: relabel vertices to a contiguous range if needed by downstream kernels.
- **Execution mode.**
  - `mg`: multi-GPU (`True`) vs single-GPU (`False`); choose `True` for largest tiers.
- **Optional edge weights.**
  - `generate_weights`: if `True`, attach weights (e.g., uniform or power-law) for weighted variants; otherwise generate an unweighted topology and set IC probabilities separately.

Instance metadata and post-processing. After generation:

1. **Assign IC parameters:** set $p_e \equiv p$ (uniform) or sample p_e i.i.d. from a chosen distribution (e.g., $\mathcal{U}(0, p_{\max})$). Store p (or distribution) as instance metadata.
2. **Knapsack variant (optional):** sample node costs w_i (e.g., uniform, power-law, or degree-proportional) and set a budget W so the optimal seed set size is typically 50–100.

Example (Python/cuGraph sketch).

```
import cugraph as cg

G = cg.generators.rmat(
    scale=22,                   # n = 2**22
    num_edges=n*16,    # edge_factor = 16
    a=0.57, b=0.19, c=0.19,    # d = 1 - a - b - c
    directed=True,
    scramble_vertex_ids=True,
    allow_self_loops=False,
    seed=42,
    mg=True                     # multi-GPU for largest tiers
)
# downstream: set IC p=0.01, run Monte Carlo evaluators, etc.
```

These settings make instance generation deterministic (via `seed`), scalable (via `scale`, `edge_factor`, `mg`), and tunable (via a, b, c, d) while keeping the influence parameters orthogonal to topology.

5 Difficulty Parameters

- the number of vertices n grows w.r.t. difficulty.
- the size of seed sets k is chosen from fixed possible values, say k = 50 or k = 100 (same as [8]) or for knapsack variant max_weight is set as a fraction of the total weight (as in [4])
- use the IC model with fixed uniform parameter p = 0.01 (same as [8]). This is because (i) it can be specified with a single scalar parameter; (ii) it simplifies Monte Carlo implementation (see the next section); and (iii) it is the most common experiment setting in the literature.
- A better_than_baseline parameter. This is the percentage the influence spread of a solution seed set, $f(S)$, is better than the spread of the baseline seed set, $f(S_0)$.

$$\text{better\_than\_baseline} = \frac{f(S) - f(S_0)}{f(S_0)}$$

S is a valid solution if the following criteria is met:

$$f(S) \geq (\text{better\_than\_baseline} + 1) \times f(S_0)$$

6 Solution Verification

Asymmetry exists between finding a solution and solution verification in influence maximization. Finding the optimal seed set is NP-hard[1].

In contrast, verifying the quality of a given seed set S is far more tractable. In the Independent Cascade (IC) model, each edge $e = (u, v)$ is associated with an activation probability p_e. Once activated, nodes remain active until the diffusion terminates. This stochastic process can be equivalently viewed as sampling a random subgraph G' of G in which each edge e is retained independently with probability p_e. For a fixed seed set S, the influence spread $f(S)$ is the expected size of the set $\Gamma_{G'}(S)$ of vertices reachable from S in G':

$$f(S) = \mathbb{E}_{G'} \left[\, |\Gamma_{G'}(S)| \, \right].$$

Since enumerating all possible G' is infeasible for large networks, we approximate $f(S)$ via Monte Carlo (MC) simulation:

$$\hat{f}(S) = \frac{1}{R} \sum_{i=1}^{R} |\Gamma_{G_i}(S)| \, ,$$

where $G_1, \ldots, G_R$ are independent samples from the IC process. Each computation $\Gamma_{G_i}(S)$ can be performed in parallel, enabling scalability to large graphs.

Monte Carlo Properties. The IC model is inherently probabilistic, and MC simulation is a natural estimator for $f(S)$:

- Exact computation requires enumerating all activation sequences, which is exponential in the number of edges.
- MC converges to the true $f(S)$ as $R \to \infty$, with variance decreasing as $\mathcal{O}(1/\sqrt{R})$.
- Larger R yields more accurate estimates but at higher computational cost; in practice, R is chosen to balance accuracy and runtime.

Practical Parameters. The original work by Kempe et al. [1] suggests $R \approx 10{,}000$ for stable evaluations on medium-sized networks. For very large graphs, smaller R may suffice if results are averaged over multiple independent verification runs. Since MC trials are parallelisable, GPU or distributed implementations can achieve high throughput, ensuring that verification remains significantly faster than the initial seed set search.

References

[1] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[2] M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[3] A. Badanidiyuru and J. Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1497–1514, 2014.

[4] Georgios Amanatidis, Federico Fusco, Philip Lazos, Stefano Leonardi, and Rebecca Reiffenhäuser. Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. *Advances in neural information processing systems*, 33:16903–16915, 2020.

[5] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Mathematical Programming*, 14(1):265–294, 1978.

[6] 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*, pages 442–446. SIAM, 2004.

[7] Graph 500 Steering Committee. Graph 500 benchmark specification. `https://graph500.org/?page_id=12`, 2017. Accessed: 2025-08-14.

[8] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 199–208, 2009.