Geodemographic Influence Maximization

Kaichen Zhang[5,1], Jingbo Zhou[1,4,*], Donglai Tao[6], Panagiotis Karras[2], Qing Li[3], Hui Xiong[7*]
[1]Business Intelligence Lab, Baidu Research, [2]Aarhus University, [3]Baidu Inc.
[4]National Engineering Laboratory of Deep Learning Technology and Applications, China
[5]Beijing University of Posts and Telecommunications, [6]Tsinghua University, [7]Rutgers University

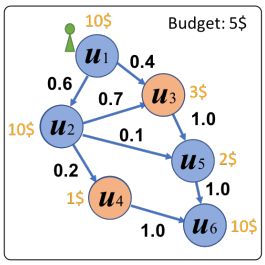
ABSTRACT

Given a set of locations in a city, on which ones should we place ads on so as to reach as many people as possible within a limited budget? Past research has addressed this question under the assumption that dense trajectory data are available to determine the reach of each ad. However, the data that are available in most industrial settings do not consist of dense, long-range trajectories; instead, they consist of statistics on people's *short-range point-to-point movements*. In this paper, we address the natural problem that arises on such data: given a distribution of population and point-to-point movement statistics over a network, find a set of locations within a budget that achieves maximum expected reach. We call this problem *geodemographic influence maximization* (GIM). We show that the problem is NP-hard, but its objective function is monotone and submodular, thus admits a greedy algorithm with a $\frac{1}{2}(1-\frac{1}{e})$ approximation ratio. Still, this algorithm is inapplicable on large-scale data for high-frequency digital signage ads. We develop an efficient deterministic algorithm, Lazy-Sower, exploiting a novel, tight double-bounding scheme of marginal influence gain as well as the *locality* proprieties of the problem; a learning-based variant, NN-Sower, utilizes randomization and deep learning to further improve efficiency, with a slight loss of quality. Our exhaustive experimental study on two real-world urban datasets demonstrates the efficacy and efficiency of our solutions compared to baselines.

ACM Reference Format:
Kaichen Zhang[5,1], Jingbo Zhou[1,4,*], Donglai Tao[6], Panagiotis Karras[2], Qing Li[3], Hui Xiong[7*]. 2020. Geodemographic Influence Maximization . In *26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403327

1 INTRODUCTION

Outdoor advertising in the form of printed posters and billboards, as well as their digital counterparts [4], is gaining appeal due to its proven effectiveness [24]; its revenue is in the order of 30 billion dollars in the US alone [27]. The target audience of such out-of-home advertising consists of people who notice printed and digital ads while engaged in daily trips. With *sparse motion pattern data* generated by positioning devices, there is an opportunity to optimize such advertising in terms of the amount of customers reached.



(a)POI network model

k	Moves	Contribution
1	u_1u_2	~~0.6~~
	$u_1\mathbf{u_3}$	0.4
2	$u_1u_2\mathbf{u_3}$	0.6*0.7=0.42
	$u_1u_2\mathbf{u_4}$	0.6*0.2=0.12
	$u_1u_2u_5$	~~0.6*0.1=0.06~~
	$u_1\mathbf{u_3}u_5$	0.4*1.0=0.40

(b)The contribution of each move when a person is to make k steps from u_1

Figure 1: An Example of GIM. When k=2 steps, orange nodes are the best solution (u_3u_4) with total influence 0.94.

Existing works on outdoor advertisement placement [26, 27] aim to maximize an influence function based on counting hits on trajectories. This approach assumes a large number of precise and long user trajectories with a high sampling rate are available, allowing to estimate the number of trajectories in a location's vicinity. However, such dense trajectory data are hard to obtain in real-world industrial settings due to reasons pertaining to user privacy and sampling fidelity [3, 7, 8]; on the other hand, it is realistic to work with *aggregate statistics* on short-range point-to-point movements.

In this paper, we address the problem of *geodemographic influence maximization* (GIM): given data on point-to-point transition probabilities and a distribution of population over a network, select a set of outdoor locations within a budget to maximize the expected reach over that population. Figure 1 illustrates a simple case of GIM. Each row in the table calculates the probability of a path of k steps between node u_1 (where a member of population stands) and any location, for $k = 1$ and $k = 2$. To maximize the aggregate probability, contributed by all selected locations, of hitting such a path under a budget of 5$, it suffices to chose locations u_3 and u_4, with total probability $0.42 + 0.12 + 0.40 = 0.94$. As paths u_1u_2 and $u_1u_2u_5$ are not affected by the solution, we strip a line over them.

While the problem is NP-hard, its objective function is monotone and submodular. We propose a greedy algorithm that picks the location of largest unit marginal influence value in each iteration, with an $\frac{1}{2}(1-\frac{1}{e})$ approximation ratio over the optimal solution. Still, its time complexity is $O(kn^2(n+m))$, where n is the number of locations, m the number of network edges, and k the maximum number of moves a person may make; that is too high for large-scale data. Nevertheless, the number of locations visited in a single outdoor trip, according to real-world data, is small; it follows that network locations influence each other *locally*. We exploit this *locality* to accelerate marginal influence calculations, only examining those parts of the network that may affect calculated values. We

*Jingbo Zhou and Hui Xiong are corresponding authors (zhoujingbo@baidu.com, hxiong@rutgers.edu). This work was done when Kaichen Zhang was an intern at Baidu Research.

KDD '20, August 23–27, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403327

also design a novel bounding scheme, *LazyTag*, which is independent of the locality of the problem; *LazyTag* maintains upper and lower bounds for the marginal influence of each network node so as to avoid redundant calculations. By virtue of these techniques, our algorithm, Lazy-Sower, reduces computational cost without sacrificing quality.

Whereas printed ads last for a long time once placed, digital signage ads are updated every few seconds. A digital advertiser may need to bid for ad locations within a short time span, hence an algorithm that suggests such locations should be highly efficient. In such a high-turnover scenario, it is worthwhile to sacrifice effectiveness for efficiency. Thereby, we develop a randomized, learning-based variant of our technique, NN-Sower. This variant trains a neural network (NN) model to choose network regions likely to yield good candidates. In each iteration, it chooses a region using the NN, randomly generates a subset of candidate nodes therein, and returns the one that maximizes unit marginal influence.

Our experiments on real-world data sets show that our model can deal with hundreds millions of mobility data records. Lazy-Sower achieves the same influence as CELF at 20% of the runtime, while NN-Sower reaches up to 91% of the influence at only 2% of the runtime, and a 55x speedup over a trivial randomized algorithm.

2 RELATED WORK

The **influence Maximization** (IM) problem calls to find k *seed nodes* in a network that maximize the spread of influence initiated therefrom by a certain diffusion model [13]. The IM problem is **NP**-hard under popular diffusion models, such as the Independent Cascade (IC) model [18, 20], yet the submodularity of the associated influence function allows for a simple greedy algorithm with an $1-{}^{1}/_{e}$ approximation ratio. Several algorithms try to speed up the influence calculation process [1, 2], while some exploit the network's modularity via a community-based strategy [5, 11, 25]. Other works extend the influence maximization problem in a manner that takes user locations into consideration. In Location-Aware Influence Maximization (LAIM) [16] the aim is to maximize the expected number of influenced users in a specific query region. Likewise, in Location-Based Influence Maximization (LBIM) [31] users transit between an online and an offline phase, while their offline decisions are affected by their locations compared to product locations. However, locations are static. Neither the LAIM nor the LBIM problem consider movement patterns as a means of exercising influence.

Overall, while the GIM problem resembles the IM problem and its variants [12, 17, 19], it differs in the *means of influence*. In IM, influence is exercised from one individual to another by means of online or offline diffusion. Contrariwise, in GIM, influence is exercised from locations to an audience by means of *movements* of audience members. Thus, solutions to the IM problem, including its location-oriented variants, are not applicable to GIM.

Other works use trajectory data to study location-oriented advertising problems. In [6], the aim is to select k trajectories to be attached with an advertisement so as to maximize the expected influence over an audience. In the Trajectory-driven Billboard Placement (TBP) problem [26], the aim is to select a set of locations, under a budget constraint, so as to maximize the number of trajectories that pass *within a distance threshold* from a chosen location. Under this objective, overlap of influence from distinct locations on the same trajectory is redundant. In a variation of TBP, the problem of Optimizing Impression Counts (OIC) [27], the influence on a trajectory grows as a logistic function of distinct *impressions* by different billboards thereupon, hence overlaps bring benefit.

We emphasize that such trajectory-oriented solutions are incommensurable to our proposal. They presume the availability of complete, dense, fine-grained, long-haul trajectories. This assumption is unrealistic in real-world industrial settings, as such long-haul trajectories are usually only available for vehicles [29, 30]. Real-world user checkpoints are, in the best case, coarse-grained, and can only be utilized as aggregate statistics [28] due to privacy concerns. We propose a problem that can be solved using available data. Instead of expecting each user's record to be an individual long-haul trajectory, we use data on short-range point-to-point movement patterns arising from any means of transportation, including vehicular, pedestrian, and public transport, to calculate the probability of movement from one point to another. Thereby, individual user data are recombined to derive combinations that are not explicitly recorded. This problem formulation is applicable on sparse, privacy-constrained data.

Symbol	Description
G	A POI graph model.
L	A given limited budget
U	A spectator's movement pattern
K	The threshold of spectators' movements
$F(S)$	The influence of a selected POI set S
$F_S(u)$	The marginal influence of u to S
C	A partitioning of G
r	The number of samples we pick in each iteration
m	The number of regions in C
H_t	The History of region selection at iteration t
$D(H_t, n)$	The n-digest of H_t, an $m \times n$ matrix

Table 1: Important Notations

3 PRELIMINARY

In this section, we define the problem and prove its **NP**-hardness; we show that the GIM objective function is submodular and monotonic, and design a greedy approximation algorithm for GIM.

3.1 Problem Definition

A *point-of-interest* (POI) network is a graph $G = (V, E, cost, coord, spec)$, where V is the vertex set with each vertex $v \in V$ representing a POI, E the edge set, *cost* a vector denoting the cost of POIs at each vertex, *coord* longitude-latitude pairs denoting the coordinates of each POI, and *spec* a distribution of *spectators*, introduced later. In the ensuing discussion, we use the terms *POI*, *location*, and *place* interchangeably. Each edge $e \in E$ is a triplet (u, v, p), denoting that a spectator moves from u to v with probability p; these probabilities, derived from real movement data, add up to 1 per node.

A *spectator* is a tuple (u, k), where vertex u denotes an initial position and k a number of moves along edges. A spectator (u, k) may move in G along any sequence of POIs $U = \{u_0, u_1, u_2, \ldots, u_k\}$, where $u_0 = u, u_i \in V, i = 1, \ldots, k$. We say that a POI set S *influences* a spectator's movement pattern U if there is a $v \in S \cap U \neq \emptyset$.

If a spectator is already at position $u \in S$, then it is definitely influenced by S. Otherwise, if it has $k = 0$ moves to do, then it is definitely not influenced by S, while if it has $k > 0$ steps to do, then it may be influenced depending on whether it moves towards some

POI in S. Thus, the probability that set S influences spectator (u, k) is defined recursively:

$$f(u,k,S) = \begin{cases} 1, & u \in S \\ 0, & k = 0 \wedge u \notin S \\ \sum_{(u,v,p)\in E} p \cdot f(v, k-1, S), & k > 0 \end{cases}$$

Spectator Distribution. Our analysis of real-world people's motion patterns indicates that most of the audience engage in a small number of moves in one trip. Let $spec(u, k)$ denote the number of spectators whose initial position is u and number of moves is exactly k, $0 \le k \le K$, where K is a given threshold. Then we define the influence of S on a population of spectators as follows.

Definition 3.1. The influence of a POI set S to all spectators is the sum of influences from S to each spectator:

$$F(S) = \sum_{u\in V} \sum_{k=0}^{K} spec(u,k) \cdot f(u,k,S)$$

We also denote the marginal influence of a location x with respect to a POI set S as $F_S(x) = F(S \cup \{x\}) - F(S)$. A location $u \in V$ incurs cost $cost(u)$ if selected, while we have a budget L. The cost of a POI set S is the total cost of all POIs in the set, $cost(S) = \sum_{u\in S} cost(u)$. Eventually, we define the GIM problem as follows.

Geodemographic Influence Maximization (GIM). Given a POI network $G = (V, E, cost, coord, spec)$ and a budget L, select a vertex subset $S \subseteq V$ with cost within L that maximizes the influence of S to all spectators. Formally:

$$\underset{S\subseteq V,\, cost(S)\le L}{\arg\max} \{F(S)\}$$

Theorem 3.2. *GIM is* **NP***-hard.*

Proof. We reduce the Knapsack problem to GIM. In Knapsack, given a budget L, we need to find a subset T of a set of tuples $I = \{(c_1, w_1), (c_2, w_2), \cdots, (c_n, w_n)\}$ that maximizes $\sum_i w_i$ while $\sum_i c_i \le L$. Given any instance of Knapsack, we map each tuple (c_i, w_i) in I to a vertex $u_i \in V$, with empty edge set E, budget L, $cost(u_i) = c_i$, and spectator distribution $spec(u_i, 0) = w_i$, 0 elsewhere. Then, a subset $S \subseteq V$ that maximizes $F(S) = \sum_{u\in V} spec(u, 0)$ with $cost(S) = \sum_{u\in S} cost(u) \le L$ corresponds to a subset T that maximizes $\sum_{(c_i,w_i)\in T} w_i$ with $\sum_{(c_i,w_i)\in T} c_i \le L$, i..e, solves Knapsack optimally. Since the mapping needs polynomial time, it follows that GIM is **NP**-hard. □

We show that $F(S)$ is monotone and submodular in the appendix.

3.2 A Basic Greedy Algorithm

Algorithm 1 presents our basic greedy algorithm, Greedy, based on the submodularity of $F(S)$. In each iteration, we add to S the vertex u that maximizes the unit marginal influence, $\frac{F_S(x)}{cost(u)}$, unless adding u violates the budget, where $F_S(x) = F(S \cup u) - F(S)$. Still, a brute-force application of this strategy will lead to an unbounded approximation ratio [10]. To avoid this disposition, we also consider the best single-vertex solution (Lines 20–21). The following theorem provides the approximation ratio of Greedy.

Theorem 3.3. *Greedy achieves an approximation of* $\frac{1}{2}(1 - 1/e)$.

Proof. $F(S)$ is monotone and submodular according to Theorems A.2 and A.4. Thus we have an approximation factor of $\frac{1}{2}(1 - 1/e)$ holds, according to [10, 14, 23]. □

In each round, Algorithm 1 computes the marginal influence of each of $O(|V|)$ vertices. The time complexity to compute $F(\cdot)$ is $O(K|V|(|V| + |E|))$, hence the time complexity of Algorithm 1 is $O(K|V|^2(|V| + |E|))$. We use array ϕ, whose size is $K|V|$, hence the space complexity of Algorithm 1 is $O(K|V|)$.

Algorithm 1: Greedy(G, L)

```
1  Function Comp_F(V*, E*, S)
2      for k = 0 to K do
3          foreach u ∈ V* do
4              if u ∈ S then φ[u,k] ← 1;
5              else if k = 0 then φ[u,k] ← 0;
6              else
7                  φ[u,k] ← 0;
8                  foreach (u,v,p) ∈ E* do
9                      φ[u,k] ← φ[u,k] + φ[v,k−1] × p ;
10     result = Σ_{u∈V} Σ_{k=0}^{K} φ[u,k] × spec(u,k);
11     return result
12 Initialize a matrix φ;
13 S ← ∅;
14 N ← V;
15 while N ≠ ∅ do
16     x* ← arg max_{x∈N} F_S(x)/cost(x);
       // F_S(x) = Comp_F(V,E,S ∪ {x}) - Comp_F(V,E,S)
17     if cost(S) + cost(x*) ≤ L then
18         S ← S ∪ {x*};
19     N ← N\{x*};
20 v* ← arg max_{v∈V, cost(v)≤L} F({v});
21 return arg max{F(S), F({v*})
```

4 OVERVIEW

Here we outline the components of our solution: (i) an improved greedy method that exploits the locality properties of the problem; (ii) a partition-based framework; and (iii) a machine learning scheme that further improves performance.

We first exploit a locality property in GIM: each location is only influenced by, and influences, certain nearby locations. We propose two ways to exploit this locality. First, we calculate *marginal influence* for a node v by visiting only appropriate neighboring locations rather than the whole vertex set. Second, in a proposed method we call *LazyTag*, we derive upper and lower bounds so as to estimate a location node's marginal influence. Upon selecting a node, we update the upper and lower bounds of influenced neighbors. We use these bounds to avoid redundant marginal influence calculations and hence reduce the computation cost. These improvements lead to a novel greedy algorithm: Lazy-Sower. Section 5 presents the details regarding the locality propriety in GIM and Lazy-Sower.

The second component of our framework, introduced in Section 6, relies on a partitioning-based heuristic that facilitates the selection of nodes from diverse network regions. We partition the network into regions. Then, in each iteration, we employ a tailored *oracle* mechanism to predict which region is likely to yield good POIs; we select a random subset of POIs in the region the oracle returns, and pick the best POI among this subset.

Section 7 presents the Neural-Network oracle (NN-Oracle) for our partition-based framework. While it is hard to predict which specific POI is best by means of a neural network, it is more feasible

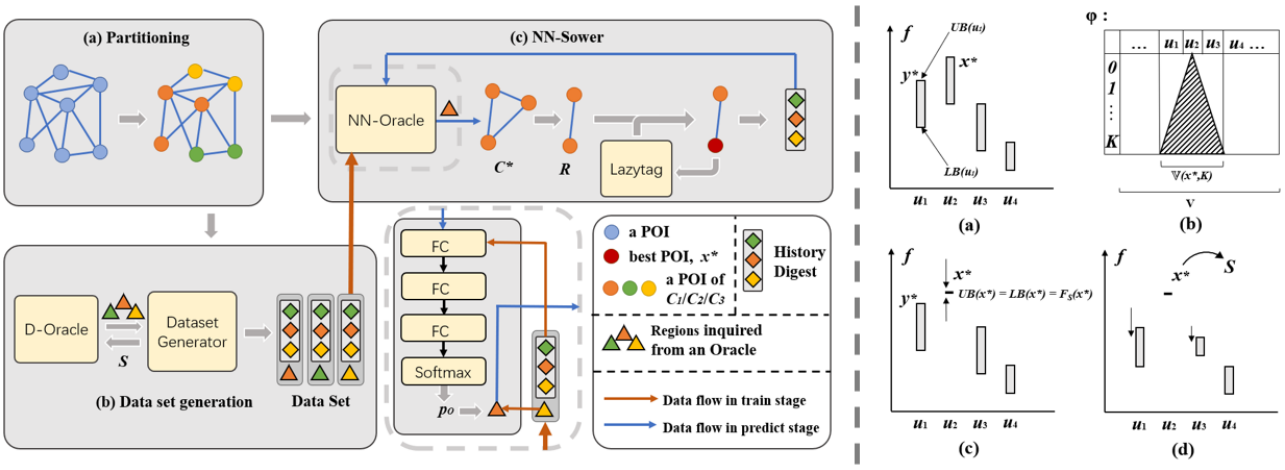


Figure 2: Left: Architecture of NN-Sower; details of NN-Oracle in the dotted box. Right: Lazy-Sower for $\mathbb{V}(u_2, K)$=$\{u_1, u_2, u_3\}$; (a) bounds of u_1, u_2, u_3, u_4 in an iteration; since $LB(x^*) < UB(y^*)$, we calculate $F_S(x^*)$; (b) calculating $F_S(x^*)$ by Marginal Influence Improvement; we only need to visit shaded area instead of whole table; (c) in the next iteration, since $LB(x^*) > UB(y^*)$, we select x^*, (d) add x^* into S and update bounds for $\mathbb{V}(x^*, K)$ by LazyTag; as u_4 is not in $\mathbb{V}(x^*, K)$, it is unnecessary to update it.

to effectively predict which region is best.We observe the historical unit marginal influence of POIs selected in a region provide good indications of the quality of that region. Thus, we collect information on selected nodes and use them as features. Our NN-Sower algorithm applies this NN-Oracle on top of the partitioning framework. Figure 2 (Left) shows the full architecture of NN-Sower.

5 LAZY-SOWER

Analyzing people's movement records, we find that people check-in at a few POIs within each trip. It follows that each network location may be influenced by, or exercise influence upon, nearby locations only. Lazy-Sower exploits this locality with two key innovations, Marginal Influence Improvement and LazyTag. Figure 2 (Right) shows an example of Lazy-Sower in operation.

5.1 Marginal Influence Improvement

The most critical component of greedy variants is the calculation of the marginal influence of a POI x on a set S, $F_S(x) = F(S \cup \{x\}) - F(S)$. Comp_F in Algorithm 1 computes $F_S(x)$ straightforwardly, incurring a high computational overhead. Here, we develop a more efficient method to compute marginal influence, exploiting the locality of the influence function. Let $g(u, k, x, S)$ denote the *probability gain*, with respect to spectator (u, k), effected by adding x to S, i.e., $g(u, k, x, S) = f(u, k, S \cup \{x\}) - f(u, k, S)$. Thus:

$$F_S(x) = \sum_{u \in V} \sum_{k=0}^{K} spec(u, k) \cdot (f(u, k, S \cup \{x\}) - f(u, k, S))$$
$$= \sum_{u \in V} \sum_{k=0}^{K} spec(u, k) \cdot g(u, k, x, S)$$

Due to the definition of f, g is recursively defined as:

$$g(u, k, x, S) = \begin{cases} 1 - f(u, k, S), & u = x \\ 0, & u \in S \vee k = 0 \\ \sum_{(u,v,p) \in E} p \cdot g(v, x, k-1, S), & k > 0 \end{cases}$$

Let $\mathbb{V}(x, K)$ denote the set of vertices that a spectator can reach POI x from them, or reach them from x ,within K steps; $\mathbb{E}(x, K)$ denotes the set of edges related to $\mathbb{V}(x, K)$. POI networks constructed from real-world data are sparse. Table 2 shows that, for $K = 5$ in Beijing data, the average of $|\mathbb{V}(x, K)|$ and $|\mathbb{E}(x, K)|$ is respectively 1/2997 and 1/2429 of $|V|$ and $|E|$. By the definition of $g(u, k, x, S)$, it follows that, if a spectator (u, k) does not reach the added POI x within k steps, the marginal probability gain, with respect to (u, k), effected by adding x to S is zero, i.e., $\forall k \in [0..K]$, if $u \notin \mathbb{V}(x, k)$, then $g(u, k, x, S) = 0$. Thus, if we have a record of $f(u, k, S)$, we only need to iterate over $\mathbb{V}(x, K)$. We thus calculate $F_S(x)$ as presented in Line 1-10 of Algorithm 2. We store $f(u, k, S)$ in $\phi[u, k]$, which we update in each iteration. The time complexity of Margin(x, S), including the update of ϕ, is $O(K(|\mathbb{V}(x, K)| + |\mathbb{E}(x, K)|))$, a significant improvement over the original $O(K(|V| + |E|))$.

5.2 LazyTag

The key idea behind the Lazy Greedy algorithm [21] as applied on a network setting [15], is to avoid redundant marginal gain recalculations, focusing only one those vertices that have a potential to be selected. The GIM problem presents a further opportunity to avoid redundant computations, thanks to its locality. We take advantage of this opportunity with the *LazyTag* method, presented in Line 30-44 of Algorithm 2. Before computing the marginal influence $F_S(x)$, we estimate an upper bound $UB(x)$ and a lower bound $LB(x)$ therefor, $LB(x) \leq F_S(x) \leq UB(x)$. In each iteration, we scan the vertex set N by decreasing upper bound per unit cost, until we arrive at an x^* such that, for all remaining $y \in N$, $\frac{LB(x^*)}{cost(x^*)} \geq \frac{UB(y)}{cost(y)}$. We

then select x^* into S, and update the upper and lower bounds for $x \in \mathbb{V}(x^*, K)$. Trivial bound values to be used in this update are $0 \le F_{S\cup\{x^*\}}(y) \le F_S(y)$. We derive tighter nontrivial bounds of $F_{S\cup\{x^*\}}(y)$ with attractive computational overhead compared to computing the marginal influence $Margin()$ for all $y \in \mathbb{V}(x^*, K)$ from scratch.

Upper bounds. To update the upper bound $UB(v)$ for $F_{S\cup\{x^*\}}(v)$, we subtract from the previous value of that bound the marginal gain contribution by all movements from x^* to v, $\hat{U} = \{x^*, u_1, \cdots, u_{k-1}, v\}$, that now contributes to $F(S \cup \{x^*\})$ but did not contribute to $F(S)$ in the previous iteration, hence $\hat{U} \cap S = \emptyset$, for each eligible length k; the subtracted value is:

$$\begin{aligned}\Delta_S(x^*, v) &= \sum_{k=0}^{K} spec(x^*, k) P(x^*, v, k, S) \\ &= \sum_{k=0}^{K} spec(x^*, k) \sum_{(v,u,p)\in E}^{u\notin S} p \cdot P(x^*, u, k-1, S)\end{aligned}$$

Where $P(x^*, u, k, S)$ is the probability of movement along k steps from x^* to u on the induced subgraph $G(V \setminus S)$ (Lines 12–17 in Algorithm 2). We calculate $\Delta_S(x^*, v)$ for all $v \in \mathbb{V}(x^*, k)$ in $O(K \cdot (|\mathbb{V}(x, K)| + |\mathbb{E}(x, K)|))$ times; Lines 18–19 subtract $\Delta_S(x^*, v)$ from $UB(v)$ for every v.

Lower bounds. To calculate a lower bound $LB(v)$ for $F_{S\cup\{x^*\}}(v)$, we *selectively* enumerate, for each eligible k, contributing movements U, with $u_k = v, U \cap (S \cup \{x^*\}) = \emptyset$. To make this enumeration selective, we introduce a threshold b, and only consider the top-b edges (u_{i-1}, u_i, p), in terms of probability p, from each node u_{i-1} along a movement. We iterate over all qualifying movements U by brute force in $O(b^K)$. In practice, we set $b = 1$, so the total time complexity of updating $LB(y)$ is $O(K \cdot |\mathbb{V}(x^*, K)|)$. Lines 20-29 in Algorithm 2 show this method.

5.3 Analysis on Lazy-Sower

Approximation ratio. As the lower bound of x^* should be larger than the upper bounds of other POIs (Lines 36–40, Algorithm 2), hence the unit marginal influence of x^* is largest, and thus Lazy-Sower picks up the POI with largest unit marginal influence in each iteration, as the basic greedy does. Hence, it achieves the same influence and approximation ratio as the basic greedy, $\frac{1}{2}(1 - 1/e)$.

Complexity. In the first iteration, Lazy-Sower computes the marginal influence for $O(|V|)$ POIs at most. Due to LazyTag, in each iteration, $O(|\mathbb{V}(x, K)|)$ POIs are put in LazyTag. Since only POIs with LazyTag are processed, there are only $O(|\mathbb{V}(x, K)|)$ POIs to compute in each subsequent iteration. Thus, the number of times computing marginal influence is $O(|V| + |V||\mathbb{V}(x, K)|) = O(|V||\mathbb{V}(x, K)|)$. The time complexity to compute marginal influence and update bounds is $O(K(|\mathbb{V}(x, K)| + |\mathbb{E}(x, K)|))$, yielding total time complexity $O(K|V||\mathbb{V}(x, K)| \cdot (|\mathbb{V}(x, K)| + |\mathbb{E}(x, K)|))$. The size of the arrays is $K|V|$, hence space complexity is $O(K|V|)$.

Comparison to CELF. CELF [15] assigns to each POI a flag and stores the previous value (marginal influence or unit marginal influence) for each POI. In each iteration, CELF sets all flags to false, and then iteratively checks the flag of the POI with largest value. If that flag is false, it updates the related value and sets the flag to true, until its finds a POI with true flag.

LazyTag differs from CELF. First, CELF utilizes trivial bounds of marginal gain (i.e., lower bound 0 and upper bound the previous marginal gain). Contrariwise, LazyTag employs much tighter bounds. CELF has to calculate marginal gain at least once, while LazyTag may, in an extreme case, directly pick the top candidate if its lower bound is larger than all upper bounds of others. Besides, CELF updates the flags of all candidates in each iteration, while LazyTag only updates some bounds. Many problems present locality properties, whereby picking one candidate only effects the marginal gain of some other candidates instead of all. In such applications, LazyTag is drastically more powerful than CELF.

Algorithm 2: Lazy-Sower(G, L)

```
1  Function Margin(x,S)
2      Initialize matrix g;
       // φ[u,k] stores current f(u,k,S)
       // g[u,k] stores current g(u,k,x,S)
3      for k = 0 to K do
4          foreach u ∈ V(x,K) do
5              if u = x then g[u,k] ← 1 − φ[u,k];
6              else if u ∈ S ∨ k = 0 then g[u,k] ← 0;
7              else
8                  foreach (u,v,p) ∈ E(x,K) do
9                      g[u,k] ← g[u,k] + p · g[v,k−1];
10     return result = Σ_{u∈V} Σ_{k=0}^{K} g[u,k] × spec(u,k)
11 Function UpdateUB(x*, S)
12     Initialize matrix π;
       // π[u,k] is used to store P(x*,u,k,S)
       // initially, π[x*,0] = 1, others are 0
13     for k = 0 to K − 1 do
14         foreach u ∈ V(x,K) \ S do
15             foreach (u,v,p) ∈ E(x,K) do
16                 if v ∈ V \ S then
17                     π[u,k+1] ← π[u,k+1] + p × π[v,k];
18     foreach v ∈ V(x*,K) \ (S ∪ {x*}) do
19         Upper[v] ← Upper[v] − Σ_{k=0}^{K} spec(x*,k) π[v,k];
20 Function UpdateLB(x*, S)
21     foreach v ∈ V(x*,K) \ S do
22         u ← v; p̂ ← 1;
23         sum ← spec(v,0);
24         for k = 1 to K do
               // only select edge with largest probability since b = 1
25             (u′,u,p) ← arg max_{u′∉S∪{x*}, (u′,u,p)∈E} p;
26             p̂ ← p̂ × p;
27             sum ← sum + spec(u′,k) × p̂;
28             u ← u′;
29         Lower[v] ← sum;
30 foreach v ∈ V do
31     Upper[v] ← Σ_{u∈V} Σ_{k=0}^{K} spec(u,k) ;
32     Lower[v] ← 0 ;
33 Initialize a matrix φ;
34 S = ∅; N = V ;
35 while N ≠ ∅ do
36     x* ← arg max_{x∈N} Upper[x]/cost(x) ;
37     y* ← arg max_{y∈N\{x*}} Upper[y]/cost(y) ;
38     if Upper[y*]/cost(y*) > Lower[x*]/cost(x*) then
39         Upper[x*] = Lower[x*] = Margin(x*, S) ;
40         continue ;
41     if cost(S) + cost(x*) ≤ L then
42         UpdateUB(x*, S); UpdateLB(x*, S);
43         S ← S ∪ {x*} ;
44         Call Comp_F(V(x*,K), E(x*,K), S) to update array φ;
45     N ← N\{x*} ;
46 v* ← arg max_{v∈V, cost(v)≤L} F({v});
47 return arg max{F(S), F({v*})}
```

6 PARTITION-BASED HEURISTIC

Here we present our partition-based framework.

Definition 6.1. (**Partitioning**) Let $C = \{C_1, C_2, \cdots, C_m\}$ denote a partitioning of the POI graph G with m regions, where $\bigcup_{k=1}^{m} C_k = V$ and $\forall i \neq j, C_i \cap C_j = \emptyset$. We call every $C_i \in C$ a region.

Figure 3 shows an example partitioning: we divide the graph into a $t \times t$ grid, so that each POI belongs to one grid cell.

Assume we had some oracle functions from a partitioning C of G to a region C^*, that returns, in each iteration, the region C^* that contains the optimal POI selection. Algorithm 3 presents this modification. In each iteration, we inquire Oracle for the best region C^*; then we sample a POI subset R from C^*. Each time we pick a POI x from C^* with probability $\frac{cost(x)}{cost(C^*)}$, and repeat r times (Line 6, `Sample()` is defined in Algorithm 5). Eventually, we add into S the POI x^* that maximizes $\frac{F_S(x)}{cost(x)}$.

Algorithm 3: Random-Partition(G, L, C, r)

```
1  S = ∅;
2  N = V;
3  Initialize the digest matrix Dig;
   // Dig is an n-digest matrix in shape of m × n.
4  while N ≠ ∅ do
5      C_i = NN-Oracle(C, Dig) ;
6      R = Sample(C*, r);
7      x* = arg max_{x∈R} F_S(x)/cost(x);
8      if cost(S) + cost(x*) ≤ L then
9          Dig[i][1..n − 1] ← Dig[i][2..n];
10         Dig[i][n] ← F_S(x*)/cost(x*);
11         S = S ∪ {x*};
12     N = N \ {x*};
13 return F(S);
```

Note that we also use the Marginal Influence Improvement and LazyTag techniques within Random-Partition. Marginal Influence Improvement enhances the calculation of $F_S(x)$, updating ϕ after picking x^*. With respect to LazyTag, we maintain bounds for all POIs, pick up x^* from the sampled subset R and y^* from $R \backslash \{x^*\}$, and compare y^* and x^* to decide whether to select x^*. Details remain as in Algorithm 2. In terms of objective value, the following guarantee applies to Random-Partition.

Theorem 6.2. *Let S be the subset selected by Random-Partition in round i, and let ξ_i be the ratio of the unit marginal gain the algorithm achieves by selecting element a_i to the* optimal *unit marginal gain. Let O be the optimal solution, and $u = \frac{1}{L}\sum_i cost(a_i)\xi_i$. Then $F(S) \geq (1 - e^{-u})F(O)$.*

Proof. Please refer to Appendix A.2 □

For an oracle to work, we need to define when a region is more preferable than another. To that end, we define the *expectation* value EC_i of each region C_i as:

$$EC_i = \sum_{x \in C_i} \frac{F_S(x)}{cost(x)} Pr[x_i^* = x] \tag{1}$$

Where $x_i^* = \arg\max_{x \in R} \frac{F_S(x)}{cost(x)}$, and R is the set sampled from C_i. A region of larger expectation is deemed to be more preferable, as it is likely to yield, after sampling, an x^* with high unit marginal influence. An ideal oracle should return the region C^* that maximizes EC_i. Yet, we need to compute EC_i. Random-Partition collects r samples, in each iteration, picking x with probability $\frac{cost(x)}{cost(C^*)}$, and selects $x_R^* = \arg\max_{x \in R} \frac{F_S(x)}{cost(x)}$. Without loss of generality, we list all the POIs in C_i in ascending order of unit marginal influence, as $v_1, v_2, ..., v_{|C_i|}$. Hence:

$$EC_i = \sum_{j=1}^{|C_i|} \frac{F_S(v_j)}{cost(v_j)} \frac{s_j^r - s_{j-1}^r}{cost(C_i)^r}$$

where $s_j = cost(v_j) + s_{j-1}$ with $s_0 = 0$. Lines 1–8 of Algorithm 4 present how we compute EC_i in detail.

Unfortunately, the computation of every EC_i incurs cost similar to Greedy, as it requires calculating the marginal influence of every vertex first. Thus, we opt for the following Oracle definition:

Definition 6.3. (**Oracle**) An Oracle O receives a partitioning C, computes a probability vector $\mathbf{p}_O = [p_{O1}, \ldots, p_{Om}]^T$, and returns a region C^* at random according to $\mathbf{p}_O$, i.e., returns C_i with probability p_{Oi}. The value of Oracle O is defined as $v_O = \mathbf{EC} \cdot \mathbf{p}_O$, where $\mathbf{EC} = [EC_1, EC_2, \cdots, EC_m]^T$.

Algorithm 4: D-Oracle(G, C, r, u)

```
   // u[i, j] stores current unit marginal influence of the j-th POI in C_i
   // Below we calculate every EC_i
1  for i = 1 to |C| do
2      s ← 0;
3      currentEC ← 0;
4      v ← sorted list of POIs in C_i;
5      for j = 1 to |C_i| do
6          currentEC ← currentEC + u[i, j] × ((s + cost(v_j))^r − s^r)/cost(C_i)^r;
7          s ← s + cost(v_j);
8      EC_i ← currentEC;
9  C* = arg max EC_i;
10 return C*
```

Algorithm 4 provides D-Oracle, an ideal oracle with a one-hot $\mathbf{p}_D$, where $p_{Di} = 1$ when EC_i is the maximum among regions, 0 otherwise. We also define Naive-Oracle as $\mathbf{p}_N = [\frac{1}{m}, \cdots, \frac{1}{m}]^T$.

7 NN-SOWER

In this section, we introduce a Neural Network Oracle (NN-Oracle), discuss its feature construction and training. Then we propose a randomized algorithm that applies NN-Oracle to Random-Partition.

7.1 Neural Network Oracle

The goal of oracle design is to maximize the $v_O = \mathbf{p}_O \cdot \mathbf{EC}$ of our Oracle O in a reasonable time. We train a neural network Oracle O to simulate D-Oracle, aiming to minimize the cross-entropy loss between $\mathbf{p}_O$ and $\mathbf{p}_D$:

$$\mathcal{L} = -\sum_{i=1}^{m} p_{Oi} \log p_{Di} \tag{2}$$

Since $\mathbf{p}_D$ is a one-hot vector, minimizing the cross entropy is very similar to a classification problem. Yet, as many regions may have similar EC values, we need not use prediction accuracy to evaluate the performance of an oracle; intead, we use the *efficiency* of an Oracle O as $\frac{v_O}{v_D}$, which fairly expresses the performance of Oracles. We opt for a Neural Network Oracle (NN-Oracle), namely a Multi-Layer Perceptron (MLP), which outputs the probability vector $\mathbf{p}_O$. NN-Sower randomly returns a region according to $\mathbf{p}_O$. The input vector $\mathbf{d}^{1\times(m\times n)}$ is a flattened and normalized *n-digest*

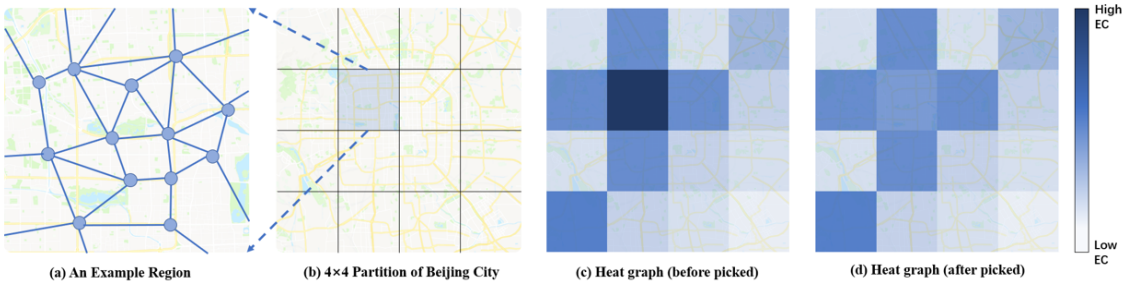


Figure 3: An example region (a) among the partition of Beijing (b) and expected values before and after picking (a).

of a *history* H, denoted by $D(H, n)$. We define these terms in the next section. Our use of an MLP illustrates the effectiveness of this concept; other networks, such as ResNet [9], can also be applied.

7.2 Feature Construction

RANDOM-PARTITION constructs S iteratively. Then, instead of merely recording the final S, we record *how S is composed step by step*, i.e., the *history* of region selection. We define the *history* as follows.

Definition 7.1. (**History**) The history H_t of region selection at iteration t is a list of records $\{(region_i, best_i)\}, 1 \le i \le t$, where $region_i$ is the region selected in the i-th iteration of RANDOM-PARTITION, and $best_i$ the unit marginal influence of x^* in that iteration.

The whole history H_t shows what S is like, and each record $(region_i, best_i)$ reflects *the current optimum in each region*, which is about G. Yet it is impractical to use the entire history; when a region C_i is picked many times, the first records from C_i are less helpful than later records, since the EC of a region falls with each POI selected. Therefore, we introduce the n-digest of a history, which keeps the last n records of each region. Formally:

Definition 7.2. (**History Digest**) The n-digest $D(H_t, n)$ of history H_t is an $m \times n$ matrix $\{d_{ij}\}$, where d_{ij} represents the unit marginal influence of x^* when picking C_i at the j-th time from the end of history. If a region C_i has been picked fewer than j times, $d_{ij} = 0$.

We record the history digest (Lines 3 and 9–10, Algorithm 3) and use it as input in NN-Oracle after min-max normalization.

7.3 NN-Sower

NN-Sower applies an NN-Oracle on top of RANDOM-PARTITION. Figure 2 shows its architecture. NN-Sower contains an offline training stage and online deployment stage. In the offline training stage, we first divide the graph into m regions and get the partitioning C. Then we call a dataset-generator several times to get the data for NN-Oracle. In each iteration, we inquire a C_i from the D-ORACLE, derive $\mathbf{p}_D$ and save $(Dig, \mathbf{p}_D)$ as a record. We train an NN-Oracle using the generated data. When we deploy NN-Sower online, we call RANDOM-PARTITION(G, L, C, r) to obtain the GIM solution.

8 EXPERIMENTAL EVALUATION

We conduct experiments on two real-world data sets in two large Chinese cities, Beijing and Chengdu. The data sets consist of two parts: **POI information** and **user movements.** We obtain data from the a third-party mobility data semantic platform, which assigns user positions to POIs, using various methods to determine that a user checked at or appeared in a POI. The data come from Beijing and Chengdu, on September 2–21, 2018. These data follow the pattern described in Section 3.1, made out of sequences of POIs per user, $U_i = \{u_{i0}, u_{i1}, u_{i2}, \ldots, u_{ik}\}$.

Based on these above data sets, we construct the Graph G where the vertex set V is a set of the POIs. We derive edge probabilities as described in Section 3.1. In particular, $\forall u, v \in V$, we have:

$$p_{u,v} = \frac{\sum_{U_i} \sum_{j=0}^{|U_i|-1} [u_{ij} = u \text{ and } u_{i(j+1)} = v]}{\sum_{U_i} \sum_{j=0}^{|U_i|-1} [u_{ij} = u]}$$

We add edge $(u, v, p_{u,v})$ to G i.f.f. $p_{u,v} \neq 0$. We get the spectator distribution *spec*, setting $spec(u, k)$ to the number of length-k movements starting from u. Table 2 provides detailed information.

As we cannot obtain the cost for all the POIs, we estimate the cost of each POI as $cost(x) = \beta \cdot sum(x)/20 + 100$, where β is a random factor ranging from 0.8 to 1.2, and $sum(x)$ the number users who have visited x in the examined time interval. This estimate is based on the economic theory that the cost of an ad on a POI consists of a variable cost and a fixed cost, the variable cost being proportional to the traffic at this POI. We also present a discussion about the deployment plan of SOWER in Appendix A.6.

	$\lvert V\rvert$	$\lvert E\rvert$	K	$\mathbb{V}(x, K)$	$\mathbb{E}(x, K)$	#Moves
Beijing	686k	2334k	5	228.9	961.9	626m
Chengdu	564k	2086k	5	262.9	1283.1	420m

Table 2: Dataset Statistics.

Parameter	Values
L	200k, 300k, **400k**, 500k, 600k
K	0, 1, 2, 3, 4, **5**, 6, 7, 8
m (Beijing)	16, 36, 64, 100, **144**, 196, 256
m (Chengdu)	16, 36, **64**, 100, 144, 196, 256
r	50, 100, **200**, 400, 800, 1600
V, E enlarged multiple	**1x**, 2x, 3x, 4x, 5x

Table 3: Parameter settings.

We compare against the following baselines:

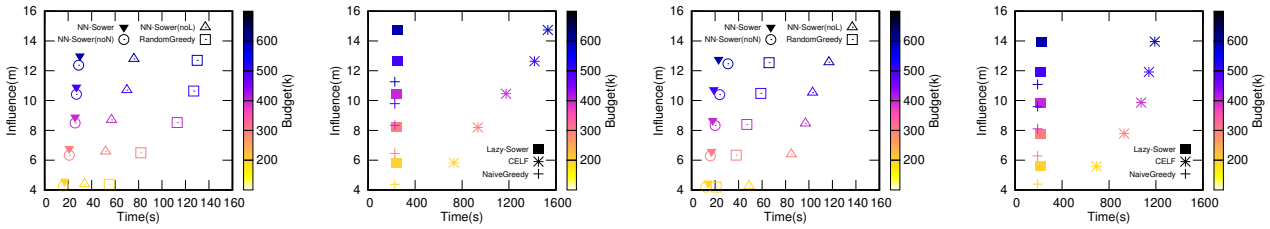
NaiveGreedy. A method that sorts POIs by unit marginal influence over an empty set, and picks POIs by the descending order.

CELF. The network-oriented version of the Lazy Greedy algorithm [15], which reuses calculations of previous iterations while picking POIs by unit marginal influence gain.

RandomGreedy. Our adaptation of the method of [22] for a budget-constrained problem. In each iteration, it takes a sample of POIs and picks the one with largest unit marginal influence.

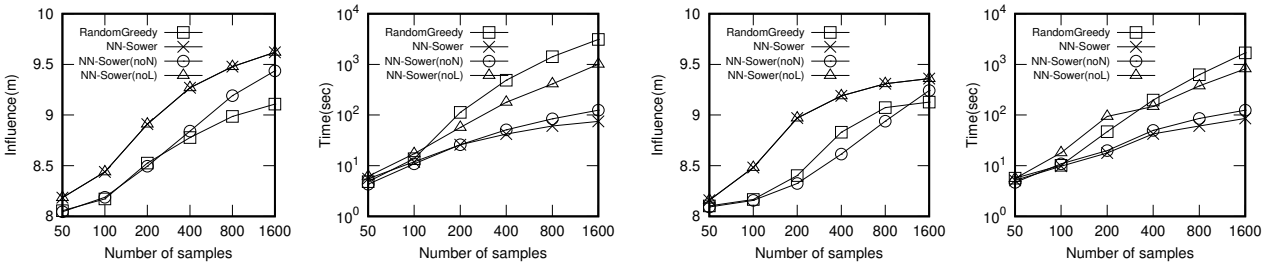
Lazy-Sower. Our deterministic greedy algorithm.

NN-Sower. Our partition-based algorithm using an NN-oracle. We also use the NN-Sower without some of its components: NN-Sower (noL) refers to NN-Sower without the LazyTag improvement;



(a) **Non-Deterministic Algs, Beijing** (b) **deterministic Algs, Beijing** (c) **Non-Deterministic Algs, Chengdu** (d) **deterministic Algs, Chengdu**

Figure 4: Effect of varying Budget L; best viewed in color — each color stands for a certain budget.



(a) **Influence, Beijing** (b) **Runtime, Beijing** (c) **Influence, Chengdu** (d) **Runtime, Chengdu**

Figure 5: Effect of varying number of samples r.

NN-Sower (noN) is NN-Sower without the NN-Oracle, i.e., the partition-based framework with Naive-Oracle.

We divide methods into Deterministic algorithms (NaiveGreedy, CELF and Lazy-Sower) and Non-deterministic algorithms (RandomGreedy, NN-Sower and its variants).

Settings. We implemented the NN-Oracle in Python and other parts in C++. Experiments run on a server with a 2.0 GHz Intel Xeon Gold 5117 CPU and 198GB memory running CentOS 6.3. When training the NN-Oracle, we use a Cirrus Logic GD 5446 GPU. Training takes advantage of the GPU. Table 3 shows the settings of all parameters with default values highlighted in bold. In our real-world data most people visit no more than five POIs in one outdoor trip; thus, we select $K = 5$ as the default value of K. We conduct each experiment 10 times and report the average result. The MLP in NN-Sower consists of three hidden ReLU layers (of sizes 512, 256, and 128) and one softmax output layer. We set the length of the History Digest to $m \times 16$, where m is the number of grid cells per city, shown in Table 3. Thus, the input layer has size $m \times 16$, and the output layer m. For each city, we call the dataset-gernerator 60 times with the bold parameters in Table 3; in about 12 hours it generates 17k records. We randomly pick 10% of records as test set, 10% as validation set, and 80% as training set. When training, we set batch size as 64, number of epochs as 50,000, and learning rate as 0.0001. Training takes 1000 seconds for each city.

8.1 Varying the Budget L

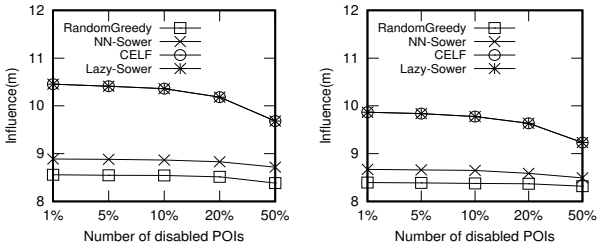
Figure 4 shows results when varying the budget L . Lazy-Sower achieves the same influence as CELF while it only takes up to 20% of the time. NN-Sower achieves 1.8% to 6.4% higher influence than RandomGreedy in about 25% of the time. Lazy-Sower achieves the highest influence, while NN-Sower obtains up to 91% of that influence in 2% of the runtime of CELF. As the budget grows from 200k to 600k, the runtime of Lazy-Sower grows by less than 5%, that of NN-Sower grows by about 70%, while those of RandomGreedy and CELF grows by up to 214% and 123%, respectively. The runtime of NaiveGreedy is stable, but its influence is always much lower than that of Lazy-Sower and their gap grows with budget. NN-Sower(noL) achieves the same influence as NN-Sower but takes more than twice of the time; NN-Sower(noN) needs nearly the same time but yields lower influence than NN-Sower. That is reasonable, since LazyTag reduces time, while the NN-Oracle gains influence.

8.2 Varying the Number of Samples r

Figure 5 shows the performance of non-deterministic algorithms with varying number of samples r. While NN-Sower and its noL version achieve the highest influence, the performance of the noN variant shows that the NN-Oracle improves influence from 1% to 5% compared to the Naive Oracle. Yet, as Figures 5(b) and (d) show, a larger number of samples r also incurs higher runtime. The runtime of NN-Sower is lower than those of other methods, and about 92% of the one of its variant without LazyTag. Surprisingly, NN-Sower with NN-Oracle is also a bit faster than NN-Sower with Naive Oracle, as its superior choices also facilitate subsequent calculations. Last, NN-Sower achieves the same influence as RandomGreedy at 1/55 of the runtime. Appendix A.4 presents an evaluation while varying the number of grid cells m.

8.3 Robustness to Disabled POIs

In real-world applications, we may not be allowed to place ads on certain POIs, as some of them may be reserved or disqualified. We evaluate the robustness of our methods by randomly selecting some POIs and rendering them ineligible. Figure 6 shows our results. Unsurprisingly, the influence of all methods drops with increasing disabled POIs. Yet the performance of RandomGreedy and NN-Sower declines slightly, while that of CELF and Lazy-Sower more perceptibly. As non-deterministic algorithms are liable to miss some good POIs anyway, they are more robust to failures [20].

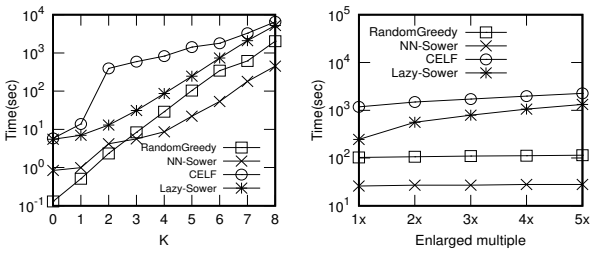


(a) **Influence(Beijing)** (b) **Influence(Chengdu)**

Figure 6: Effect of varying disabled POIs

8.4 Scalability

The threshold K determines whether a spectator can reach POI x. Figure 7(a) shows that NN-Sower needs a little more time than RandomGreedy when $K < 3$, since the NN computation takes most of the time in these cases. However, when $K > 3$, NN-Sower outperforms RandomGreedy by a factor of 3 to 6. The runtime of CELF is higher than Lazy-Sower. We also evaluate scalability in growing graph size. We considered adding vertices and edges in the graph. However, doing so properly would require a network growth model, which is beyond the scope of this work. We test the scalability by duplication: we copy the same graph several times and use the union of these graphs as the input. The duplicated POIs have the same cost, coordinates and spectator distribution as their original ones. For example, if it is $G = (V = \{u_1, u_2\}, E = \{(u_1, u_2, 1), (u_2, u_1, 1)\})$ and we copy it twice, then the new graph is $G' = (V' = \{u_1, u_2, u_1^*, u_2^*\}, E' = \{(u_1, u_2, 1), (u_2, u_1, 1), (u_1^*, u_2^*, 1), (u_2^*, u_1^*, 1)\})$. Figure 7(b) shows that both our proposals scale gracefully with graph size.



(a) **Varying** K (b) **Varying** $|V|$,$|E|$

Figure 7: Scalability on Beijing dataset.

9 CONCLUSIONS

We introduced and studied the probelm of Geodemographi Influence Maximization. We showed that this problem is NP-hard, but its objective function is monotone and submodular. Going beyond previous work in the area, we equipped our algorithm with a novel, tight double-bounding scheme that accelerates marginal influence gain calculations, exploiting the locality properties of the problem. Thereby, we built Lazy-Sower, an algorithm that achieves equal to the state-of-the-art CELF method at much lower runtime, and can handle large urban-network data. Furthermore, we devised a learning-based variant, NN-Sower, that utilizes randomization and deep learning to enhanced efficiency even further with only a slight loss of quality. Our extensive experiment evaluations on two real-world datasets verify the effectiveness, efficiency, scalability, and robustness of our methods.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (Grant No.91746301,61725205) and the Danish Council for Independent Research (Research Project 9041-00382B).

REFERENCES

[1] Christian Borgs, Michael Brautbar, Jennifer T. Chayes, and Brendan Lucier. 2014. Maximizing Social Influence in Nearly Optimal Time. In *SODA*. 946–957.
[2] Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*. 1029–1038.
[3] Yves-Alexandre De Montjoye, Laura Radaelli, Vivek Kumar Singh, et al. 2015. Unique in the shopping mall: On the reidentifiability of credit card metadata. *Science* 347, 6221 (2015), 536–539.
[4] Charles Dennis, Richard Michon, J. JoŠko Brakus, Andrew Newman, and Eleftherios Alamanos. 2012. New insights into the impact of digital signage as a retail atmospheric tool. *Journal of Consumer Behaviour* 11, 6 (2012), 454–466.
[5] Aram Galstyan, Vahe Musoyan, and Paul Cohen. 2009. Maximizing influence propagation in networks with community structure. *Physical Review E* 79, 5 (2009), 056102.
[6] Long Guo, Dongxiang Zhang, Gao Cong, Wei Wu, and Kian-Lee Tan. 2016. Influence maximization in trajectory databases. *IEEE TKDE* 29, 3 (2016), 627–641.
[7] Tianyi Hao, Jingbo Zhou, Yunsheng Cheng, Longbo Huang, and Haishan Wu. 2016. User identification in cyber-physical space: a case study on mobile query logs and trajectories. In *SIGSPATIAL*. 1–4.
[8] Tianyi Hao, Jingbo Zhou, Yunsheng Cheng, Longbo Huang, and Haishan Wu. 2020. A Unified Framework for User Identification across Online and Offline Data. *IEEE TKDE* (2020).
[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.
[10] Thibaut Horel. 2015. Notes on Greedy Algorithms for Submodular Maximization. Online.
[11] Shixun Huang, Zhifeng Bao, J Shane Culpepper, and Bang Zhang. 2019. Finding Temporal Influential Users over Evolving Social Networks. In *ICDE*. 398–409.
[12] Sergei Ivanov, Konstantinos Theocharidis, Manolis Terrovitis, and Panagiotis Karras. 2017. Content Recommendation for Viral Social Influence. In *SIGIR*.
[13] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD*. 137–146.
[14] Samir Khuller, Anna Moss, and Joseph Seffi Naor. 1999. The budgeted maximum coverage problem. *Inform. Process. Lett.* 70, 1 (1999), 39–45.
[15] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In *KDD*. 420–429.
[16] Guoliang Li, Shuo Chen, Jianhua Feng, Kian-Lee Tan, and Wen-Syan Li. 2014. Efficient location-aware influence maximization. In *SIGMOD*. 87–98.
[17] Yuchen Li, Ju Fan, George V. Ovchinnikov, and Panagiotis Karras. 2019. Maximizing Multifaceted Network Influence. In *ICDE*. 446–457.
[18] Alvis Logins and Panagiotis Karras. 2019. Content-Based Network Influence Probabilities: Extraction and Application. In *ICDM Workshops*. 69–72.
[19] Alvis Logins and Panagiotis Karras. 2019. An Experimental Study on Network Immunization. In *EDBT*. 726–729.
[20] Alvis Logins, Yuchen Li, and Panagiotis Karras. 2020. On the Robustness of Cascade Diffusion under Node Attacks. In *WWW*. 2711–2717.
[21] Michel Minoux. 1978. Accelerated greedy algorithms for maximizing submodular set functions. In *IFIP Conference on Optimization Techniques*. 234–243.
[22] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Kraus. 2015. Lazier Than Lazy Greedy. In *AAAI*. 1812–1818.
[23] G. L. Nemhauser and L. A Wolsey. 1978. Best algorithms for approximating the maximum of a submodular set function. *Math. Operat. Res.* 3, 3 (1978), 177–188.
[24] Lex van Meurs and Mandy Aristoff. 2009. Split-Second Recognition: What Makes Outdoor Advertising Work? *Journal of Advertising Research* 49, 1 (2009), 82–92.
[25] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. 2010. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *KDD*. 1039–1048.
[26] Ping Zhang, Zhifeng Bao, Yuchen Li, Guoliang Li, Yipeng Zhang, and Zhiyong Peng. 2018. Trajectory-driven Influential Billboard Placement. In *KDD*.
[27] Yipeng Zhang, Yuchen Li, Zhifeng Bao, Songsong Mo, and Ping Zhang. 2019. Optimizing Impression Counts for Outdoor Advertising. In *KDD*. 1205–1215.
[28] Jingbo Zhou, Hongbin Pei, and Haishan Wu. 2018. Early warning of human crowds based on query data from Baidu maps: Analysis based on Shanghai stampede. In *Big data support of urban planning and management*. 19–41.
[29] Jingbo Zhou, Anthony KH Tung, Wei Wu, and Wee Siong Ng. 2013. R2-D2: a system to support probabilistic path prediction in dynamic environments via "semi-lazy" learning. *VLDB* 6, 12 (2013), 1366–1369.
[30] Jingbo Zhou, Anthony KH Tung, Wei Wu, and Wee Siong Ng. 2013. A "semi-lazy" approach to probabilistic path prediction in dynamic environments. In *KDD*.
[31] Tao Zhou, Jiuxin Cao, Bo Liu, Shuai Xu, Ziqing Zhu, and Junzhou Luo. 2015. Location-based influence maximization in social networks. In *CIKM*. 1211–1220.

A APPENDIX

A.1 Properties of Influence Function

We now prove that $F(S)$ is a monotonic and submodular function.

Lemma A.1. *The function $f(u,k,S)$ is submodular, i.e., for any $u \in V$, k, $X \subseteq Y \subseteq V$, and $x \in V \backslash Y$, it is*

$$f(u,k,X \cup \{x\}) - f(u,k,X) \geq f(u,k,Y \cup \{x\}) - f(u,k,Y)$$

Proof. We use induction on k. When $k = 0$, by definition, the differences on both sides are either 0 or 1. Then, if the difference on the left side is 1, the inequality holds. Otherwise, if the difference on the left side is 0, then it should hold that $u \neq x$, hence either $u \in Y$ or $u \in V \backslash Y \backslash \{x\}$. If $u \in Y$, then $f(u,0,Y \cup \{x\}) = f(u,0,U,Y) = 1$, while if $u \in V \backslash Y \backslash \{x\}$, then $f(u,0,Y \cup \{x\}) = f(u,0,Y) = 0$; in both cases, the inequality holds. Now, let us assume that the inequality holds for k and consider the case of $k+1$:

$$\begin{aligned} & f(u,k+1,X \cup \{x\}) - f(u,k+1,X) \\ &= \sum_{(u,v,p) \in E} p[f(v,k,X \cup \{x\}) - f(v,k,X)] \\ &\geq \sum_{(u,v,p) \in E} p[f(v,k,Y \cup \{x\}) - f(v,k,Y)] \\ &= f(u,k+1,Y \cup \{x\}) - f(u,k+1,Y) \end{aligned}$$

Where the inequality in the third line holds due to our assumption. Since both the base case and the inductive step hold, the lemma is proven. □

Theorem A.2. *$F(S)$ is submodular.*

Proof. For every X, Y⊆ V with X⊆ Y and every x $\in V \backslash Y$, it is

$$\begin{aligned} & F(X \cup \{x\}) - F(X) \\ &= \sum_{u \in V} \sum_{k} spec(u,k)(f(u,k,X \cup \{x\}) - f(u,k,X)) \\ &\geq \sum_{u \in V} \sum_{k} spec(u,k)(f(u,k,Y \cup \{x\}) - f(u,k,Y)) \\ &= F(Y \cup \{x\}) - F(Y) \end{aligned}$$

The inequality in the third line holds due to Lemma A.1. So F is a submodular function. □

Next, we will prove that $F(S)$ is monotonic.

Lemma A.3. *The function $f(u,k,S)$ is monotone, i.e., for every $u \in V$, k, and $X \subseteq Y \subseteq V$, it is*

$$f(u,k,X) \leq f(u,k,Y)$$

Proof. We perform induction on k. The base case of $k = 0$ holds by definition. Let us assume that the lemma holds for k and consider the case of $k+1$:

$$\begin{aligned} f(u,k+1,X) &= \sum_{(u,v,p) \in E} p \cdot f(v,k,X) \\ &\leq \sum_{(u,v,p) \in E} p \cdot f(v,k,Y) = f(v,k+1,Y) \end{aligned}$$

Then the lemma follows inductively. □

Theorem A.4. *$F(S)$ is monotonic.*

Proof. $\forall X \subseteq Y \subseteq V$, it is

$$\begin{aligned} F(X) &= \sum_{u \in V} \sum_{k} spec(u,k) f(u,k,X) \\ &\leq \sum_{u \in V} \sum_{k} spec(u,k) f(u,k,Y) = F(Y) \end{aligned}$$

The inequality holds due to Lemma A.3. □

A.2 Proof of Theorem 6.2

Proof. Let $S_i = \{a_1, a_2, \cdots, a_i\}$, note that:

$$\begin{aligned} \frac{F(S_i) - F(S_{i-1})}{cost(a_i)} &= \xi_i \max_{o \in V \backslash S_{i-1}} \frac{F_{S_{i-1}}(o)}{cost(o)} \\ &\geq \xi_i \sum_{o \in O \backslash S_{i-1}} \frac{F_{S_{i-1}}(o)}{cost(o)} \frac{cost(o)}{cost(O \backslash S_{i-1})} \\ &\geq \frac{\xi_i}{L} \sum_{o \in O \backslash S_{i-1}} F_{S_{i-1}}(o) \\ &\geq \frac{\xi_i}{L} (F(O) - F(S_{i-1})) \end{aligned}$$

Here, the third inequality can be deduced from the submodularity of F stated in Theorem A.2 and A.4.

Rearranging, we get:

$$F(S_i) \geq \frac{cost(a_i)\xi_i}{L} (F(O) - F(S_{i-1})) + F(S_{i-1})$$

Next, we use induction to show that

$$F(S_i) \geq \left(1 - \prod_{j=1}^{i} \left(1 - \frac{cost(a_j)\xi_j}{L}\right)\right) F(O)$$

For the base case of $i = 1$ along with $S_0 = \emptyset$, we have:

$$\begin{aligned} F(S_1) &\geq \frac{cost(a_1)\xi_1}{L} (F(O) - F(S_0)) + F(S_0) \\ &= \frac{cost(a_1)\xi_1}{L} F(O) \\ &= \left(1 - \left(1 - \frac{cost(a_1)\xi_1}{L}\right)\right) F(O) \end{aligned}$$

For any round i, the following holds according to the inductive hypothesis:

$$\begin{aligned} & F(S_i) \geq \frac{cost(a_i)\xi_i}{L} (F(O) - F(S_{i-1})) + F(S_{i-1}) \\ &= \frac{cost(a_i)\xi_i}{L} F(O) + \left(1 - \frac{cost(a_i)\xi_i}{L}\right) F(S_{i-1}) \\ &\geq \frac{cost(a_i)\xi_i}{L} F(O) \\ &+ \left(1 - \frac{cost(a_i)\xi_i}{L}\right)\left(1 - \prod_{j=1}^{i-1} \left(1 - \frac{cost(a_j)\xi_j}{L}\right)\right) F(O) \\ &= \left(1 - \prod_{j=1}^{i} \left(1 - \frac{cost(a_j)\xi_j}{L}\right)\right) F(O) \end{aligned}$$

Further applying $1 - x \geq e^{-x}$, the inequality implies:

$$F(S_i) \geq \left(1 - \prod_{j=1}^{i} e^{-\frac{cost(a_j)\xi_j}{L}}\right) F(O)$$
$$= \left(1 - e^{-\sum_{j=1}^{i} \frac{cost(a_j)\xi_j}{L}}\right) F(O)$$

As for S, we have:

$$F(S) = F(S_{|S|}) \geq (1 - e^{-u}) F(O)$$

□

A.3 Data Generation

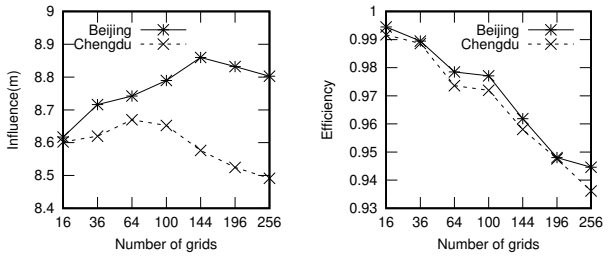
In the offline training stage, we generate data as follows. In each iteration, we inquire a C_i from the D-Oracle and derive $\mathbf{p}_D$. Same as in Random-Partition, we sample a POI subset R from C_i, and select the POI x^* that has the maximum $\frac{F_S(x^*)}{cost(x^*)}$. Then we save $(Dig, \mathbf{p}_D)$ as a piece of data and add the record $(C_i, \frac{F_S(x^*)}{cost(x^*)})$ into the digest Dig. After that, we maintain $u[i, j]$ by updating POIs near x^* immediately after picking x^* (Lines 28–29), since D-Oracle requires the exact unit marginal influence for every POI. So its time complexity is $O(K|V||\mathbb{V}(x, K)|\cdot(|\mathbb{V}(x, K)|+|\mathbb{E}(x, K)|))$ and its space complexity is $O(K|V|)$. Algorithm 5 shows the details.

Algorithm 5: Dataset-Generator(G, L, C, r)

```
1  Function Sample(N, r)
2      Initialize an array ord;
       // N = u_1, u_2, ..., u_n
       // ord[0] = 0, and ∀i ∈ [1..n], ord[i] = ord[i-1] + cost(u)
3      R ← ∅;
4      for i = 1 to r do
5          pos ← rand() × ord[n];
6          find t ∈ [1..n] s.t. ord[t-1] ≤ pos < ord[t];
7          R ← R ∪ {u_t};
8      return R
9  Initialize the digest matrix Dig;
10 Initialize matrix u;
11 data ← ∅;
12 S ← ∅;
13 N ← V;
14 for i = 1 to |C| do
15     for j = 1 to |C_i| do
           // Let y be the j-th POI of C_i
16         u[i, j] ← F_S(y) / cost(y)
17 while N ≠ ∅ do
18     C_i ← D-Oracle(G, C, r, u);
19     p_D ← a one-hot vector where p_D[i] = 1;
20     R ← Sample(C*, r);
21     x* ← arg max_{x∈R} F_S(x) / cost(x);
22     if cost(S) + cost(x*) ≤ L then
23         data ← data ∪ {(Dig, p_D)};
24         Dig[i][1..n-1] ← Dig[i][2..n];
25         Dig[i][n] ← F_S(x*) / cost(x*);
26         S ← S ∪ {x*};
           // Next we update u. Let x* be the d-th POI of C_i.
27         u[i, d] = 0 ;
28         foreach y ∈ V(x*, K) \ S do
               // Let y be the t-th POI of C_j
29             u[j, t] ← F_S(y) / cost(y);
30     N ← N\{x*};
31 return data
```

A.4 Varying Number of Grid Cells m

As the number of grid cells m determines the input and output size of NN-Sower, we fix its value for all experiments. Figure 8 shows the effect of m on quality and runtime. We observe that: (1) *Efficiency* decreases as m grows: the more grid cells there are to choose from, the harder it is to choose the best. (2) still, as m grows, the attained influence v_O rises first, and then falls; that is because $v_O = v_D \cdot Efficiency$, where v_D increases with the growth of m even while *Efficiency* falls; for example, if each POIs is in a separate grid cell, then v_D equals the largest unit marginal influence value among all POIs. We opt for $m = 144$ for Beijing and $m = 64$ for Chengdu; as Beijing is larger, it requires more grid cells to prevent having too many POIs per cell.



(a) **Influence** (b) **Efficiency**

Figure 8: Effect of varying number of grids cells m.

A.5 Usability of NN-Sower

We point out that NN-Sower performs well even under different settings of budget L and number of samples r. In all the evaluation results of Figure 5 and 4, the Neural Network model is trained with $L = 400$K and $r = 200$. However, NN-Sower performs well even with other settings of L and r. This outcome illustrates the usability of NN-Sower; it suffices to train the model once, and then let it deal with diverse L and r.

A.6 Deployment Plan

SOWER is developed for Baidu Juping[1], an ecosystem of intelligent marketing and offline media digitalization in the AI era established by Baidu since 2018. Given the evidence provided, our algorithm will be deployed on the backend to automatically generate a package of POIs for delivering advertisement. In the first step, the Lazy-Sower will be deployed for solving the Guarantee Delivery (GD) advertisement. Later, thanks its high efficiency, NN-Sower will be deployed for real-time biding (RTB) advertisement.

Lazy-Sower can be deployed and executed immediately because it's a deterministic algorithm. While NN-Sower can be re-trained infrequently, such as a month or a quarter, because it can handle various situations well once trained according to Section 8.3 and Appendix A.5. Thus we can use only one server to support all training tasks for hundreds of cities in China.

[1]https://juping.baidu.com/