Geodemographic Influence Maximization

Kaichen Zhang
Beijing University of Posts and Telecommunications

Jingbo Zhou
Baidu Research

Donglai Tao
Tsinghua University

Panagiotis Karras
Aarhus University

Hui Xiong
Rutgers University
Baidu Research

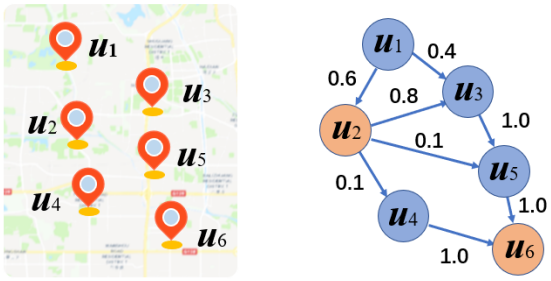
ABSTRACT

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 the reach of an ad is determined by a predefined set of *trajectories*. However, in most real-world situations, a set of trajectories is not available. Instead, *mobility data* are expressed in the form of statistics on point-to-point movements of people. In this paper, we address the natural problem that arises in this setting: given a distribution of population and point-to-point movement statistics over a network, select a set of locations within a budget so as to maximize its 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 admitting a greedy algorithm with a $\frac{1}{2}(1-\frac{1}{e})$ approximation ratio. Still, the time complexity of this algorithm hinders its application on large-scale data. We propose an efficient solution, utilizing a novel, tight double-bounding scheme of marginal influence gain as well as the *locality* proprieties of the problem. We propose a deterministic algorithm, Lazy-Sower, which achieves the same result as the state-of-the-art CELF algorithm, with a significant efficiency advantage thanks to its exploitation of the bounding scheme and locality, and a learning-based variant, NN-Sower, which utilizes randomization and deep learning to further improve efficiency, with a slight loss of quality. An exhaustive experimental study on two real-world urban datasets demonstrates the efficacy and efficiency of our solutions compared to baselines.

PVLDB Reference Format:
. A Sample Proceedings of the VLDB Endowment Paper in LaTeX Format. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Outdoor posters and billboards provide a means of advertising that has the potential to be both highly effective and also appealing [17], while its revenue is in the order of 30 billion dollars in the US alone [20]. The target audience of such out-of-home advertising consists of people who notice posters and billboards while engaged in daily trips. With data on such trips generated by positioning devices, there is an opportunity to optimize the effectiveness of such advertising in terms of the amount of customers reached.



(a) Real-world POI distribution (b) POI network model

Figure 1: An Example of POI network model. (a) A distribution of real-world POIs. (b) POI network model, where blue nodes are POIs without ads, and orange nodes are those with ads.

Existing works about outdoor advertisement placement [19, 20] aim to maximize an influence function based on counting hits on trajectories. This approach assumes the availability of a large number of precise individualized user trajectories with a high sampling rate, so as to accurately estimate the number of trajectories in a location's vicinity. However, it is hard to obtain individual users' trajectory data in real-world applications. Yet it is easier to obtain statistics on point-to-point transitions from check-in records.

In this paper, we address the problem of *geodemographic influence maximization* (GIM): select a set of outdoor locations within a budget to maximize the expected reach over a distribution of population, using aggregate data on point-to-point transition probabilities. Several challenges arise from the GIM problem. First, we have to consider the influence on the same user from different locations. For example, if most people come to location A from B, if we put an advertisement at B, it is unnecessary to put one at A. We need to build a transition graph to model such factors. Such a network will have a high density of locations in urban areas. While the problem is NP-hard, its objective function is monotone and submodular. Thus, we propose a greedy algorithm that picks the location of largest unit marginal influ-

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

ence 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, i.e., too high for large-scale data.

Analyzing people's real-life movement records, we find that people visit a few places in one outdoor trip. Thus, influence from one network location to another is *local*. We exploit this locality property to improve the calculation of marginal influence, only examining those parts of the networks that may affect calculated values. As this calculation is in the core of the greedy algorithm, its improvement has a significant effect on efficiency. We also design a novel bounding scheme, *LazyTag*, which is an innovation in its own right, 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 called Lazy-Sower, achieves the same result as the state-of-the-art CELF algorithm at much lower time complexity and computational cost in practice. We improve efficiency further with a learning-based variant, NN-Sower, that partitions the network in regions and trains a neural network (NN) model to choose a region likely to yield good candidates. Then, in each iteration, it chooses a region using the NN, randomly generates a subset of candidate nodes therein, and returns the node that maximizes unit marginal influence.

We conduct experiments on real-world data sets. The results 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

Here we review areas of related work that have a bearing on our problem.

2.1 Submodular Function Maximization

A set function is *submodular* if it presents *diminishing returns*: adding an element X to a set S brings about a larger increase than adding X to a superset of S. Formally, if $F(X \cup \{x\}) - F(X) \geq F(Y \cup \{x\}) - F(Y)$ for every $X \subseteq Y \subseteq V$ and every $x \in V \backslash Y$ where V is the universal set of F, then F is a submodular function. Submodularity has implications on problems such as maximum cut [6], set cover [4], and rank function of matroids [3], which pertain to the optimization of set functions under cardinality, budget, or matroid constraints [11]. A *standard greedy* algorithm [16] achieves an approximation ratio of $1-1/e$ under cardinality constraints and $\frac{1}{2}(1-1/e)$ under budget constraints in $O(n^2)$ time, under the assumption that calculating the function takes $O(1)$. An *accelerated* variant [14] of standard greedy, *lazy greedy* exploits the submodularity property further to reduce complexity, and has been often reinvented [12]. More recently, a *stochastic* variant of lazy greedy, *stochastic greedy* [15] achieves an approximation ratio of $(1-1/e-\epsilon)$, in expectation, under cardinality constraints, in $O(n \log \frac{1}{\epsilon})$ time. The GIM objective is a submodular function. Still, applying generic submodular function maximization techniques on GIM yields a rudimentary performance. We achieve high efficiency by exploiting a novel bounding scheme and specific features of the problem.

2.2 Influence Maximization in Social Networks

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 [9]. The IM problem is **NP**-hard under popular diffusion models, such as the Independent Cascade (IC) model, yet the submodularity of the associated influence function allows for a simple greedy algorithm with a $1-1/e$ approximation ratio. Several algorithms try to speed up the influence calculation progress [1, 2], while some exploit the network's modularity via a community-based strategy [5, 18].

Some works extend the influence maximization problem in a manner that takes user locations into consideration. In Location-Aware Influence Maximization (LAIM) [13] the aim is to maximize the expected number of influenced users in a specific query region. Likewise, in Location-Based Influence Maximization (LBIM) [21] 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; influence in LBIM is not determined by movement patterns. Neither the LAIM nor the LBIM problem consider movement patterns as a means of exercising influence in the manner that we do.

Overall, while the GIM problem resembles the IM problem and its variants, 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 habitual *movements* of audience members. Thus, solutions to the IM problem, including its location-oriented variants, are not applicable to GIM. A secondary difference is that we define GIM under a budget constraint L, as the cost of locations can vary widely, while IM is defined under a cardinality constraint k, assuming uniform cost.

2.3 Trajectory-driven Influence Maximization

Some other works use trajectory data to study location-oriented advertising problems. In [7], 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 [19] the aim is to select a set of locations, under a budget constraint, so as to maximize the number of trajectories in a database 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) [20], considers that the influence on a trajectory grows as a logistic function of distinct *impressions* by different billboards thereupon, hence overlaps bring benefit.

Such trajectory-oriented solutions rely on the availability of complete, dense vehicle trajectories, which they process on the fly; this approach constrains scalability to large data sets; under data sparsity, they may wrongly consider that one is not influenced by a billboard with a uniform distance threshold. Contrariwise, in GIM, we require neither a restrictive distance threshold nor dense checkpoints. Instead, we utilize data about people's habitual movement patterns, and analyze them offline in advance; thus, GIM solutions are applicable on data arising from any means of transportation including vehicular, pedestrian, and public transport, where checkpoints may be limited and coarse-grained, as users may not keep their tracking device always on.

3. PRELIMINARY

In this section, we define the problem and prove its **NP**-hardness. Next, we show that the GIM objective target function is submodular, and calculate its value.

3.1 Problem Definition

A point-of-interest (POI) network is a *probabilistic* graph $G = (V, E, cost, spec)$, where V is the vertex set, E is the edge set, *cost* a vector denoting the cost of each POI, and *spec* a distribution of *spectators*. Each vertex $v \in V$ represents a *point of interest* (POI); 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 an audience member moves from u to v with probability p.

A *spectator* is a tuple (u, k), where vertex u denotes an initial position and integer k denotes 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:

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

Spectator Distribution. Our analysis of real-world people's motion patterns indicates that the number of moves is smaller than a certain threshold for most of the audience. Let $spec(u, k)$ denote the number of spectators whose initial position is u and number of moves is at most k, $0 \leq k \leq K$, where K is an given threshold. Then we define the influence of S on a population of spectators as follows.

Definition 1. The influence of a POI set S to all spectators $v \in V$ 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.

PROBLEM 1. ***Geodemographic Influence Maximization (GIM)***. *Given a POI network $G = (V, E, cost, spec)$ 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:*

$$\arg\max_{S \subseteq V,\, cost(S) \leq L} \{F(S)\}$$

THEOREM 1. *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 $\sum_i c_i \leq B$. 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 $cost(S) = \sum_{u \in S} cost(u) \leq L$ corresponds to a subset T the maximizes $\sum_{(c_i, w_i) \in T} w_i$ with $\sum_{(c_i, w_i) \in T} c_i \leq L$, i..e, solves KNAPSACK optimally. Since the mapping needs polynomial time, it follows that GIM is **NP**-hard. □

3.2 Properties of Influence Function

We now prove that $F(S)$ is a monotonic and submodular function, and propose a way to calculate its value.

LEMMA 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 proved. □

THEOREM 2. *$F(S)$ is submodular.*

PROOF. For every X, Y⊆ V with X⊆ Y and every x ∈ $V \backslash Y$, 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 1. So F is a submodular function. □

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

LEMMA 2. *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 3. *$F(S)$ 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 2. □

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*})
```

3.3 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 [8]. 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 4. GREEDY *achieves an approximation factor of* $\frac{1}{2}(1-1/e)$.

PROOF. $F(S)$ is monotone and submodular according to Theorems 2 and 3. Thus we have an approximation factor of $\frac{1}{2}(1-1/e)$ holds, according to [16, 10, 8]. □

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 $O(K|V|^2(|V|+|E|))$.

Symbol	Description
G	A POI graph model. To be specific, $G = (V, E, cost, spec)$
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

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.

4.1 Lazy-Sower and Locality

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.

4.2 Partitioning-based Heuristic

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. We train a neural network as our oracle; different oracle mechanisms can be used in principle.

4.3 NN-Sower

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 to effectively predict which region is best.

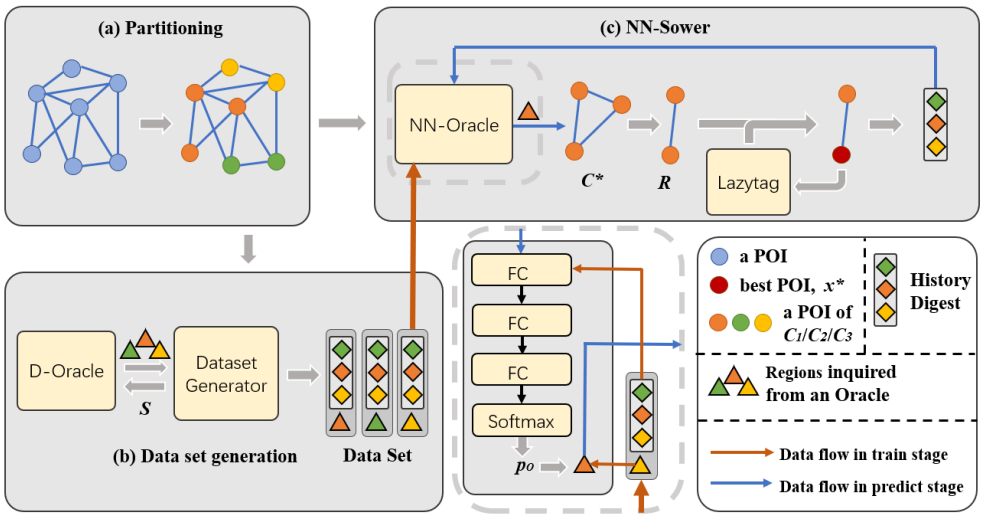


Figure 2: Architecture of our framework. Details of NN-Oracle is presented in the dotted box.

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 shows the full architecture of NN-Sower.

5. LAZY-SOWER

In this section, we outline the locality property of GIM and introduce Lazy-Sower (also shown in Algorithm 2) that exploits this locality to introduce two key innovations, Marginal Influence Improvement and LazyTag, thereby lowering time complexity without loss of quality.

5.1 Locality property

Analyzing people's movement records, we find that people check-in at a few POIs within each day trip. It follows that each network location may be influenced by, or exercise influence upon nearby locations only. In our real-world data most people visit no more than five POIs in one outdoor trip; thus, we select $K = 5$. We develop techniques that exploit this property.

5.2 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:

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

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

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

Let $\mathbb{V}(x, K)$ and $\mathbb{E}(x, K)$ denote the set of vertices and edges, respectively, from which a spectator can reach POI x within K steps; for $K = 5$, the average of $|\mathbb{V}(x, K)|$ and $|\mathbb{E}(x, K)|$ is far less than $|V|$ 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 $\mathbb{V}(x, K)$, i.e., over vertices u such that x is within the reach of u. 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 $O(K(|V| + |E|))$.

5.3 LazyTag

The key idea behind the Lazy Greedy algorithm [14] as applied on a network setting [12], 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 31-47 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 sampled vertex set R by decreasing upper bound per unit cost, until we arrive at an x^* such that, for all remaining $y \in R$, $\frac{LB(x^*)}{cost(x^*)} \geq \frac{UB(y)}{cost(y)}$. 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

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;
23         p̂ ← 1;
24         sum ← spec(v,0);
25         for k = 1 to K do
               // we only select the edge with largest
                  probability since b = 1
26             (u′,u,p) ← arg max^{u′∉S∪{x*}}_{(u′,u,p)∈E} p;
27             p̂ ← p̂ × p;
28             sum ← sum + spec(u′,k) × p̂;
29             u ← u′;
30         Lower[v] ← sum;
31 foreach v ∈ V do
32     Upper[v] ← Σ_{u∈V} Σ_{k=0}^{K} spec(u,k) ;
33     Lower[v] ← 0 ;
34 Initialize a matrix φ;
35 S = ∅ ;
36 N = V ;
37 while N ≠ ∅ do
38     x* ← arg max_{x∈N} Upper[x]/cost(x) ;
39     y* ← arg max_{y∈N\{x*}} Upper[y]/cost(y) ;
40     if Upper[y*]/cost(y*) > Lower[x*]/cost(x*) then
41         Upper[x*] = Lower[x*] = Margin(x*, S) ;
42         continue ;
43     if cost(S) + cost(x*) ≤ L then
44         UpdateUB(x*, S);
45         UpdateLB(x*, S);
46         S ← S ∪ {x*} ;
47         Call Comp_F(𝕍(x*,K), 𝔼(x*,K),S) to update array φ;
48     N ← N\{x*} ;
49 v* ← arg max_{v∈V,cost(v)≤L} F({v});
50 return arg max{F(S), F({v*})
```

update are $0 \leq F_{S\cup\{x^*\}}(y) \leq F_S(y)$; we proceed to define tighter variants in the next paragraph.

We derive tight nontrivial bounds of $F_{S\cup\{x^*\}}(y)$ with attractive computational overhead compared to computing the marginal influence $Margin(y, S \cup \{x^*\})$ for $y \in \mathbb{V}(x^*, K)$ 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 $O(K*(|\mathbb{V}(x,K)|+|\mathbb{E}(x,K)|))$ time; 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 $O(K \cdot |\mathbb{V}(x^*, K)|)$. Line 20-30 in Algorithm 2 show this method.

5.4 Analysis on Lazy-Sower

Approximation ratio. As the lower bound of x^* should be larger than the upper bounds of other POIs (Line 38–42, 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, $\frac{1}{2}(1 - 1/e)$.

Time 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 $O(K^2|V||\mathbb{V}(x, K)| \cdot (|\mathbb{V}(x, K)| + |\mathbb{E}(x, K)|))$.

Comparison to CELF. CELF [12] 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 in several ways. First, CELF only utilizes trivial bounds of marginal gain (i.e., lower bound 0 and upper bound the previous marginal gain). Contrariwise, LazyTag employs much tighter bounds that confer an advantage. CELF has to calculate marginal gain at least once, while LazyTag may, in an extreme case, even 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 more drasticaly more powerful than CELF. As we will see, in our experiments with real-world datasets, Lazy-Sower is much faster than CELF.

Algorithm 3: RANDOM-PARTITION(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  S = ∅;
10 N = V;
11 Initialize the digest matrix Dig;
   // Dig is an n-digest matrix in shape of m × n.
12 while N ≠ ∅ do
13     C_i = NN-ORACLE(C, Dig) ;
14     R = Sample(C*, r);
15     x* = arg max_{x∈R} F_S(x)/cost(x);
16     if cost(S) + cost(x*) ≤ L then
17         Dig[i][1..n-1] ← Dig[i][2..n];
18         Dig[i][n] ← F_S(x*)/cost(x*);
19         S = S ∪ {x*};
20     N = N\{x*};
21 return F(S);
```

6. PARTITION-BASED HEURISTIC

Here, we present our partition-based method framework, along with a naive oracle that uniformly picks a region from the partitioning at random. We formulate the oracle design and discuss the partition strategy.

6.1 Partitioning Framework

Definition 2. (**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.

Assume we had some NN-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, which we call RANDOM-PARTITION. 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 (Lines 1–8). Eventually, we add into S the POI x^* that maximizes $\frac{Margin(x)}{cost(x)}$. Algorithm 4 presents a trivial such oracle, NAIVE-ORACLE, which returns a region uniformly at random.

Algorithm 4: NAIVE-ORACLE(C)

```
1 C* = a region picked from C uniformly at random;
2 return C*
```

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 5. *Let S be the subset selected by* RANDOM-PARTITON *in round i, and let ξ_i be the ratio of the marginal gain the algorithm achieves by selecting element a_i to the* optimal *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. Let $S_i = \{a_1, a_2, \cdots, a_i\}$, note that:

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

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

Rearranging, we get:

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

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:

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

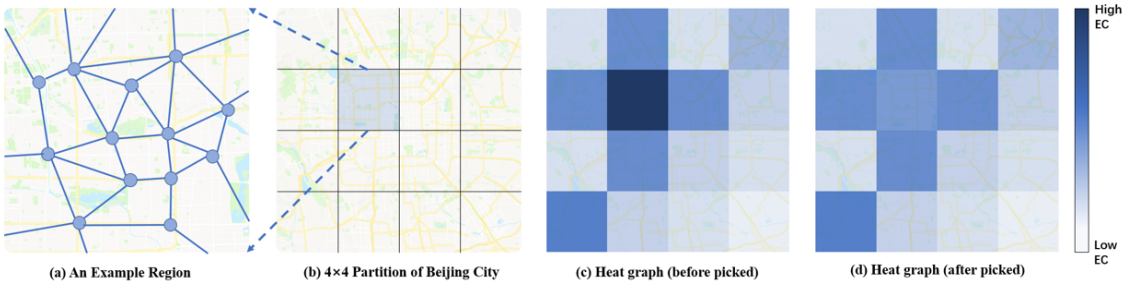


Figure 3: An example partitioning; (a) is an example region among the partition of Beijing presented in (b); (c) & (d) show the expectation values before and after picking the region of (a).

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:

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

□

6.2 Oracle Design

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:

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

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, a 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-PARTITIONED 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 5 present how we compute EC_i in detail.

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

Definition 3. An ORACLE O function 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}; O may also receive other input parameters so as to determine $\mathbf{p}_O$.

Algorithm 5: 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*
```

In NAIVE-ORACLE, $\mathbf{p}_N = [\frac{1}{m}, \cdots, \frac{1}{m}]^T$; An ideal oracle, D-ORACLE, given in Algorithm 5, has a one-hot $\mathbf{p}_D$, where:

$$p_i = \begin{cases} 1 & EC_i \text{ is the maximum among all regions} \\ 0 & otherwise \end{cases}$$

We define the value of ORACLE A as $v_A = \mathbf{EC} \cdot \mathbf{p}_A$, where $\mathbf{EC} = [EC_1, EC_2, \cdots, EC_m]^T$. ORACLE A is better than ORACLE B if $v_A > v_B$, since we are likely to pick an x^* with higher unit marginal influence using A than using B. We aim to strike a balance between time cost and oracle value.

6.3 Partitioning Strategy

Let us discuss the partitioning strategy. We observe that:

1. A partitioning should increase performance by applying D-ORACLE. Since $\mathbf{v}_D \geq \mathbf{v}_O$ for any Oracle O, when $\mathbf{v}_D$ is big enough, other Oracles may have a good performance.

2. The selection of a vertex from region C^* should incur a small decrease in the EC_i value of adjacent regions. This property ensures EC_i easy to predict.

A trivial partitioning strategy assigns only one POI per region: $\forall i, |C_i| = 1$. This strategy maximizes the value of D-ORACLE, yet fails in terms of rule (2). To balance rule (1) and (2), we apply an empirical GRID strategy: we divide the graph into a $t \times t$ grid according the geographic position of each vertex, and then to convert each grid cell into each region. In our daily life, the response of downtown ads is often higher than rural ones. Thus, GRID may satisfy rule (1) in certain cases. Regarding rule (2), the locality of the problem ensures that the effect of each choice is limited to a small part of the graph of $\mathbb{V}(x, K)$.

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 in oracle design is to maximize the v_O of our ORACLE O in a reasonable time:

$$v_O = \mathbf{p}_O \cdot \mathbf{EC} = \sum_{i=1}^{m} p_{O\,i} EC_i \tag{2}$$

While it is time-consuming to call D-ORACLE online, it is feasible to use D-ORACLE as ground truth offline. We train a neural network Oracle O to simulate D-ORACLE. Formally, we minimize the cross-entropy loss between $\mathbf{p}_O$ and $\mathbf{p}_D$:

$$\mathcal{L} = -\sum_{i=1}^{m} p_{O\,i} \log p_{D\,i} \tag{3}$$

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. As Figure 2 shows, 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* of a *history* H, $D(H, n)$; we define these terms in the following.

Algorithm 6: NN-ORACLE(C, d)

```
1 p_O ← MLP(d);
2 C* ← a region randomly picked from C according to the
    probability vector p_O;
3 return C*
```

7.2 Feature construction

As D-ORACLE uses S and G to select a region ($u[i, j]$ is extracted from S and G), it is reasonable to include the information about S and G in the input vector, using a bitmap to store S or using the distribution of cost and spectators in G. Yet S and G may be big, rendering the input vector very long and the cost of prediction prohibitive. Besides, the calculation of marginal influence and EC is quite complex, hence the neural network cannot converge to a good result using trivial features. We seek to a more robust feature construction method. We observe that 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. Formally, we define the *history* as the follows.

Definition 4. (**History**) The history H_t of region selection at iteration t is a list of records $\{(region_i, best_i)\}$, where $region_i$ represents the region selected in the i-th iteration of RANDOM-PARTITION, and $best_i$ represents the unit marginal influence of x^* in that iteration. We define the *begin* and *end* of H_t as the *head* $((region_1, best_1))$ and *tail* $((region_t, best_t))$ of a list.

A history reflects properties of G and S: the whole history H_t shows what S is like, and each record $(region_i, best_i)$ reflects *the current best of 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 5. (**History Digest**) The n-digest $D(H_t, n)$ of a 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, $d_{ij} = 0$.

The history digest is more concise than the raw input S. We use it as the input of our NN-Oracle after min-max normalization. Lines 11 and 17–18 in Algorithm 3 record the history digest.

7.3 Data Generation

In the train stage, we generate data as follows. In each iteration, we inquire a C_i from the D-ORACLE, record EC, and derive $\mathbf{p}_D$. We add the record $(C_i, \frac{margin(x^*,S)}{cost(x^*)})$ into the digest Dig. Thus, we treat Dig as input and $\mathbf{p}_D$ as output. Algorithm 7 shows the details. Since D-ORACLE requires the exact unit marginal influence for every POI, we maintain $u[i, j]$ by updating POIs near x^* immediately after picking x^* (Lines 19–20).

Algorithm 7: DATASET-GENERATOR(G, L, C, r)

```
1  Initialize the digest matrix Dig;
2  Initialize matrix u;
3  data ← ∅;
4  S ← ∅;
5  N ← V;
6  for i = 1 to |C| do
7      for j = 1 to |C_i| do
           // Let y be the j-th POI of C_i
8          u[i, j] ← F_S(y) / cost(y)
9  while N ≠ ∅ do
       // record EC when calling D-Oracle
10     C_i ← D-Oracle(G, C, r, u);
11     R ← Sample (C*, r);
12     x* ← arg max_{x∈R} F_S(x) / cost(x);
13     if cost(S) + cost(x*) ≤ L then
14         Dig[i][1..n − 1] ← Dig[i][2..n];
15         Dig[i][n] ← F_S(x*) / cost(x*);
16         data ← data ∪ {(Dig, EC)};
17         S ← S ∪ {x*};
           // Next we update u. Let x* be the d-th POI of C_i.
18         u[i, d] = 0 ;
19         foreach y ∈ 𝕍(x*, K) \ S do
               // Let y be the t-th POI of C_j
20             u[j, t] ← F_S(y) / cost(y);
21     N ← N\{x*};
22 return data
```

7.4 NN-Sower

NN-Sower applies an NN-Oracle on top of RANDOM-PARTITION. Figure 2 shows its architecture. In a training stage, we divide the graph into m regions and get the partitioning C, call a DATASET-GENERATOR several times to get the data

for NN-Oracle, and train an NN-Oracle using the generated data. Then 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:

1. **POI information.** For any POI, we collect its name and coordinates.
2. **User movements.** We obtain user movements in Beijing and Chengdu from September 2–21, 2018. These data follow the pattern described in Section 3.1, as sequences of POIs $U = \{u_0, u_1, u_2, \ldots, u_k\}$. The data are obtained from Baidu mobility data semantic platform, which assigns user positions to POIs, using various methods to determine that a user checked or appeared in at a POI.

Based on these above data sets, we construct the Graph G where the vertex set V is a set of the POIs. The edges of G are constructed as described in Section 3.1. To ensure that each POI has at least one outgoing edge, we add an edge $(u, u, 1)$ if POI u lacks outgoing edges. We get the spectator distribution *spec*, setting $spec(u, k)$ to the number of length-k movements starting from u. The threshold K is the max value of *spec*. 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 ads on a POI consists of a variable cost and a fixed cost, the variable cost being proportional to the traffic at this POI.

8.1 Compared Methods

To our knowledge, this is the first work on the GIM problem. Thus, we compare the Lazy-Sower and NN-Sower against the following baselines:

- **Naive-Greedy.** A method that sorts POIs by unit marginal influence over an empty set, and picks POIs by the descending order until it runs out of budget.
- **CELF.** The network-oriented version of the Lazy Greedy algorithm [12], which reuses calculations of previous iterations to accelerate marginal influence computations.
- **RandomGreedy.** Our adaptation of the method of [15] for a budget-constrained problem. In each iteration, it takes a sample of POIs, calculates the unit marginal influence of POIs in the sample, and picks the POI with the largest value therein.
- **Lazy-Sower.** Our deterministic greedy algorithm.
- **NN-Sower.** Our partition-based algorithm using an NN-oracle. We also use the NN-Oracle without some of its components: NN-Sower (noL) refers to NN-Sower without the LazyTag improvement; NN-Sower (noN) is NN-Sower without the NN-Oracle, i.e., the partition-based framework with Naive-Oracle.

We divide methods into two groups: Deterministic algorithms (Naive-Greedy, CELF, RandomGreedy) and Non-deterministic algorithms (RandomGreedy, NN-Sower and its variants). Since BasicGreedy is too slow to deal with our datasets, while other works have shown that CELF achieves the same object in much less time, we do not include Basic-Greedy into our experiments.

	$\|V\|$	$\|E\|$	K	#User movements
Beijing	686,385	2,334,271	5	626,607,685
Chengdu	564,388	2,086,387	5	420,255,787

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

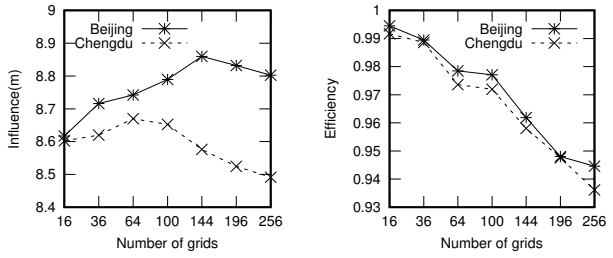
Table 3: Parameter settings.

8.2 Experimental Settings

We implemented the NN-Oracle in Python and the rest of the codes in C++. Experiments run on a server with a 2.0 GHz Intel Xeon Gold 5117 CPU and 198GB memory running CentOS/6.3 OS. When training the NN-Oracle, we use a Cirrus Logic GD 5446 GPU.

Table 3 shows the settings of all parameters with default values highlighted in bold. In all experiments, we vary one parameter while keeping the rest at default values. We conduct each experiment 10 times and report the average result.

The multi-layer perceptron (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 the test set, 10% as the validation set, and 80% as the training set. When training, we use batch size 64, 50,000 epochs, and a learning rate of 0.0001. Training takes 1000 seconds for each city.



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

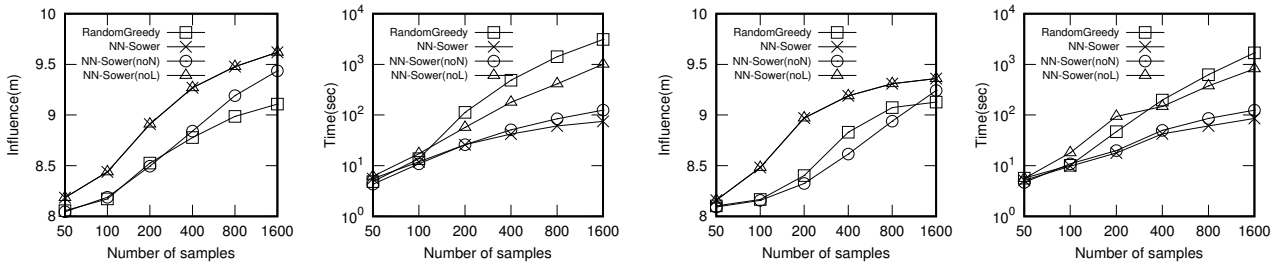
Figure 4: Effect of varying number of grids cells m.

Method	Budget									
	200k		300k		400k		500k		600k	
	Inf (m)	Time (s)	Inf (m)	Time (s)	Inf (m)	Time (s)	Inf (m)	Time (s)	Inf (m)	Time (s)
RandomGreedy	4.38	55	6.49	82	8.52	113	10.64	127	12.70	130
NN-Sower	**4.59**	**17**	**6.78**	**21**	**8.90**	**26**	**10.9**	**27**	**13.00**	**30**
NN-Sower(noL)	4.59	34	6.78	52	8.90	57	10.9	70	13.00	76
NN-Sower(noN)	4.26	16	6.33	21	8.49	26	10.42	27	12.37	29
Lazy-Sower	**5.83**	**240**	**8.20**	**244**	**10.46**	**245**	**12.64**	**247**	**14.74**	**250**
CELF	5.83	730	8.20	935	10.46	1175	12.64	1418	14.74	1631
NaiveGreedy	4.38	227	6.45	227	8.33	227	9.79	227	11.26	227

Table 4: Effect of varying budget L on Beijing dataset.

Method	Budget									
	200k		300k		400k		500k		600k	
	Inf (m)	Time (s)	Inf (m)	Time (s)	Inf (m)	Time (s)	Inf (m)	Time (s)	Inf (m)	Time (s)
RandomGreedy	4.19	21	6.34	38	8.40	47	10.48	59	12.54	66
NN-Sower	**4.46**	**14**	**6.59**	**17**	**8.66**	**18**	**10.73**	**19**	**12.77**	**23**
NN-Sower(noL)	4.46	49	6.59	85	8.66	97	10.73	103	12.77	117
NN-Sower(noN)	4.18	12	6.28	16	8.32	20	10.40	24	12.47	31
Lazy-Sower	**5.59**	**212**	**7.78**	**215**	**9.87**	**219**	**11.92**	**220**	**13.96**	**222**
CELF	5.59	692	7.78	929	9.87	1074	11.92	1140	13.96	1190
NaiveGreedy	4.39	192	6.28	192	8.10	192	9.59	192	11.08	192

Table 5: Effect of varying budget L on Chengdu dataset.



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

Figure 5: Effect of varying number of samples r.

8.3 Number of Grid Cells m

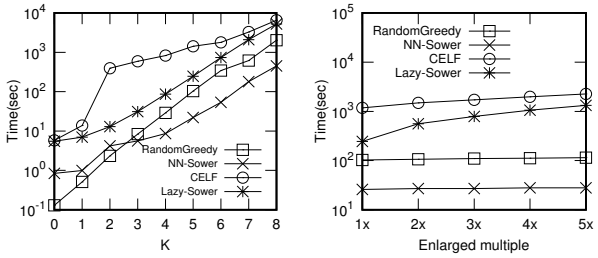
As the number of grid cells m determines the input and output size of NN-Sower, we fix its value for all experiments. Figure 4 shows the effect of m on quality and runtime. We observe that: (1) *Efficiency* decreases as m grows: the more grid cells there are to choose from, the harder it is to choose 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.

8.4 Varying the Budget L

Tables 4 and 5 show the results of all methods when varying the budget L ("Inf" stands for "Influence"). 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.

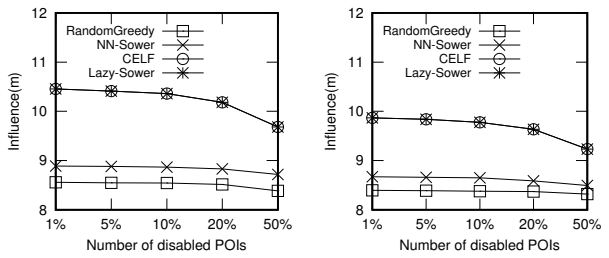
8.5 Varying the Number of Samples r

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,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.



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

Figure 6: Scalability on Beijing dataset.



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

Figure 7: Effect of varying disabled POIs

8.6 Scalability in K

The threshold K determines whether a spectator can reach POI x, thus the number of vertices and edges, $|\mathbb{V}(x, K)|$ and $|\mathbb{E}(x, K)|$. Figure 6(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, which lacks the computational advantage of Lazy-Sower, is higher.

8.7 Scalability in Graph Size

We now design an experiment to evaluate the efficiency of our model with varying graph size. Since our solutions can process whole-city data sets with low runtime, it would be uninteresting to evaluate scalability on smaller graphs. We considered adding fake vertices and edges in the graph. However, doing so properly would require a network growth model, which is beyond the scope of this work. Random additions of vertices may change $|\mathbb{V}(x, K)|+|\mathbb{E}(x, K)|$, hence it would be hard to determine how graph size affects runtime, all other factors being equal. Thus, 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 $G = (V = \{p_1, p_2\}, E = \{(p_1, p_2, 1), (p_2, p_1, 1)\})$ and we copy it twice, then the new graph is $G' = (V' = \{p_1, p_2, p_1^*, p_2^*\}, E' = \{(p_1, p_2, 1), (p_2, p_1, 1), (p_1^*, p_2^*, 1), (p_2^*, p_1^*, 1)\})$. This method changes graph size while preserving other characteristics.

As we can see in Figure 6(b), the runtime of non-deterministic algorithms (NN-Sower and RandomGreedy) is almost stable with growing graph size, as their runtime mainly depends on r and K. On the other hand, deterministic algorithms (Lazy-Sower and CELF) have runtime growing with graph size, as they need to process all POIs. Among non-deterministic algorithms, NN-Sower takes much less time than RandomGreedy; among deterministic algorithms, Lazy-Sower takes much less time than CELF. Thus, both our proposals scale gracefully with graph size for algorithms of their kind.

8.8 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 Tables 4 and 5, and Figure 5, 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.

8.9 Robustness to Disabled POIs

In real-world applications, it is always possible that we may not be allowed to place ads on certain POIs, as some of them may be reserved, disqualified for ad placements, or already occupied by other ads. In this section we evaluate the robustness of our methods in such cirucmstances, by randomly selecting some POIs and rendering them ineligible. Figure 7 sows our results. Unsurprisingly, the influence of all methods drops with increasing number of disabled POIs. However, those of RandomGreedy and NN-Sower undergo relatively minor change, while the performance of CELF and Lazy-Sower declines more perceptibly. One possible explanation of this phenomenon is that non-deterministic algorithms are liable to miss some POIs in any case, hence they are more robust to disablements, while deterministic algorithms take all POIs in consideration, thus are more sensitive to failures.

9. CONCLUSIONS

We introduced and studied the Geodemographic Influence Maximization problem: given a distribution of population and point-to-point movement statistics over a network, select a set of locations within a budget constraint so as to maximize its expected reach on people. We have proven that this problem is NP-hard, but its objective function is monotone and submodular. Thus, we designed a greedy approximation algorithm. We went beyond previous work in the area by equipping our algorithm with a novel, tight double-bounding scheme that accelerates its marginal influence gain calculations, while we also exploited 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.

10. REFERENCES

[1] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier. Maximizing Social Influence in Nearly Optimal Time. *arXiv e-prints*, page arXiv:1212.0884, Dec 2012.

[2] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, pages 1029–1038. ACM, 2010.

[3] J. Edmonds. Matroids, submodular functions and certain polyhedra. *Combinatorial Structures and Their Applications*, page 69–87, 1970.

[4] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM 45*, pages 634–652, 1998.

[5] A. Galstyan, V. Musoyan, and P. Cohen. Maximizing influence propagation in networks with community structure. *Physical Review E*, 79(5):056102, 2009.

[6] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM 42*, page 1115–1145, 1995.

[7] L. Guo, D. Zhang, G. Cong, W. Wu, and K.-L. Tan. Influence maximization in trajectory databases. *IEEE Transactions on Knowledge and Data Engineering*, 29(3):627–641, 2016.

[8] T. Horel. Notes on greedy algorithms for submodular maximization. Online, 2015.

[9] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146. ACM, 2003.

[10] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[11] A. Krause and D. Golovin. *Submodular Function Maximization*, page 71–104. Cambridge University Press, 2014.

[12] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429. ACM, 2007.

[13] G. Li, S. Chen, J. Feng, K.-L. Tan, and W.-S. Li. Efficient location-aware influence maximization. In *SIGMOD*, pages 87–98. ACM, 2014.

[14] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In J. Stoer, editor, *Optimization Techniques*, pages 234–243, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[15] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrak, and A. Kraus. Lazier than lazy greedy. In *AAAI*, pages 1812–1818, 2015.

[16] G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[17] L. van Meurs and M. Aristoff. Split-second recognition: What makes outdoor advertising work? *Journal of Advertising Research*, 49(1):82–92, 2009.

[18] Y. Wang, G. Cong, G. Song, and K. Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *KDD*, pages 1039–1048. ACM, 2010.

[19] P. Zhang, Z. Bao, Y. Li, G. Li, Y. Zhang, and Z. Peng. Trajectory-driven influential billboard placement. In *KDD*, pages 2748–2757, 2018.

[20] Y. Zhang, Y. Li, Z. Bao, S. Mo, and P. Zhang. Optimizing impression counts for outdoor advertising. In *KDD*, pages 1205–1215, 2019.

[21] T. Zhou, J. Cao, B. Liu, S. Xu, Z. Zhu, and J. Luo. Location-based influence maximization in social networks. In *CIKM*, pages 1211–1220. ACM, 2015.