Towards an Efficient Weighted Random Walk Domination

Songsong Mo[1], Zhifeng Bao[2], Ping Zhang[3], Zhiyong Peng[1]
[1]Wuhan University, [2]RMIT University, [3]HUAWEI
songsong945@whu.edu.cn,zhieng.bao@rmit.edu.au,zhangping62@huawei.com,peng@whu.edu.cn

ABSTRACT

In this paper, we propose and study a new problem called the weighted random walk domination. Given a weighted graph $G(V, E)$ and a budget B of the weighted random walk, it aims to find a k-size set S, which can minimize the total costs of the remaining nodes to access S through the weighted random walk, which is bounded by B. This problem is critical to a range of real-world applications, such as advertising in social networks and telecommunication base station selection in wireless sensor networks. We first present a dynamic programming based greedy method (DpSel) as a baseline. DpSel is time-consuming when $|V|$ is huge. Thus, to overcome this drawback, we propose a matrix-based greedy method (MatrixSel), which can reduce the computation cost greatly. To further accelerate MatrixSel, we propose a BoundSel approach to reduce the number of the gain computations in each candidate selection by proactively estimating the upper bound of the marginal gain of the candidate node. Notably, all methods can achieve an approximation ratio of $(1 - 1/e)$. Experiments on real datasets have been conducted to verify the efficiency, effectiveness, memory consumption and scalability of our methods.

PVLDB Reference Format:
Songsong Mo, Zhifeng Bao, Ping Zhang, Zhiyong Peng. Towards an Efficient Weighted Random Walk Domination. PVLDB, 14(4): 560 - 572, 2021.
doi:10.14778/3436905.3436915

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/songsong945/RandomWalkDomination.

1 INTRODUCTION

Graph data constitutes a set of entities (nodes) and interconnections (edges) between them, both of which may have attributes associated with them. In many real-life applications, a weight is usually associated with each edge, indicating the strength of the interconnections. The semantic meaning of the edge weights vary from one scenario to another. For example, in influence maximization [17, 32], weighted pagerank [37] and personalized pagerank [25], the edge weight indicates the probability that a node chooses or impacts its neighbors; in road networks [36] and wireless sensor networks [35, 38], the edge weight represents the cost of moving a node to its neighbor. In this paper, we propose and study the Weighted Random Walk Domination (WRWD) problem.

Proceedings of the VLDB Endowment, Vol. 14, No. 4 ISSN 2150-8097.
doi:10.14778/3436905.3436915

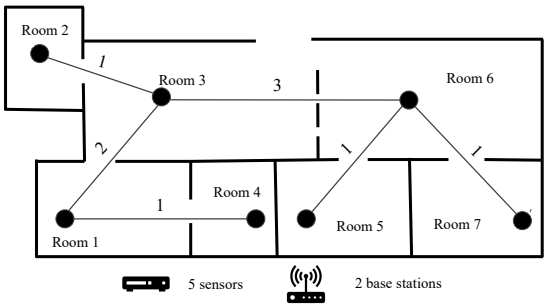


Figure 1: A motivating example.

Given an edge-weighted graph $G(V, E, wt)$ and a budget B of the weighted random walk, WRWD aims to find a k-size subset $S \in V$, which can minimize the total costs of the nodes in V to access S through the weighted random walk. The budget B is used to bound the cost (length) of the weighted random walk. WRWD can benefit many downstream applications, just to name a few: (1) In the field of information retrieval in social networks, the browsing process of users on social networks (also called social browsing) can be well-simulated by random walks [23, 27, 40]. With the development of the Internet, social browsing has become the main way for users to get information on the Internet [20, 21, 28]. Based on the social browsing, how to select k users' home pages in the network so that other users can easily browse these pages has great practical significance for social media advertising. For example, a cosmetics company wants to promote its new lipstick. It can choose k users for free trial and advertising on Facebook. If other users can easily browse to the lipstick trial, the impact of the new lipstick will increase significantly. (2) In the field of Wireless Sensor Networks (WSNs), random walk is widely used for routing protocols [16, 24, 33, 41], and one fundamental issue is to provide coverage and connectivity [15]. For example, as shown in Figure 1, five sensors and two base stations need to be placed in seven rooms to detect the temperature. The base station can send the temperature data to computer directly. However, the sensor can only send temperature data to its neighbors randomly, and this will consume a certain transmission cost. To this end, it needs to choose a node set as base stations to minimize the energy cost from the remaining nodes to those selected.

Recall that the meaning of edge weights vary from one to another. Consequently, without loss of generality, we introduce two models, the *probability-aware random walk model* $\mathcal{PR}$ and the *cost-aware random walk model* $\mathcal{CR}$, to cater for most (if not all) scenarios where WRWD can be applied. The efficiency of solving WRWD is critical, because in reality, graphs are constantly evolving and the edge weight that indicates the degree of interconnection between nodes

*Songsong Mo, Zhifeng Bao and Ping Zhang have contributed equally. Ping Zhang and Zhiyong Peng are the corresponding authors. This work was done when Songsong Mo and Ping Zhang were visiting students at RMIT University.

will change over time [19, 22]. Thus, the seed set needs to be up-to-date as well. Our theoretical analysis shows that the objective function of WRWD is submodular. Due to the fact that the problem of maximizing a submodular function is generally NP-hard [26], we turn to develop a range of approximate algorithms with theoretical guarantees. We first extend a Dynamic Programming (DP) based greedy method (DpSel), which can provide an approximation ratio of (1-1/e), to solve WRWD. However, it cannot be applied to large graphs since DP is too slow to calculate marginal gains. To address this issue, we use a matrix to compute the marginal gain and propose a matrix-based greedy selection (MatrixSel), which is n ($n = |V|$) times faster than DpSel. To accelerate MatrixSel further, we devise a bound-based greedy selection (BoundSel) approach to further reduce the number of the gain computations in each candidate selection by proactively estimating the upper bound of the marginal gain of the candidate node.

In summary, we make the following contributions.

- We propose the weighted random walk domination problem WRWD and introduce two weighted random walk models, the probability-aware model $\mathcal{PR}$ and the cost-aware model $\mathcal{CR}$, to cater for different scenario needs. We also prove that the objective function of WRWD is monotone and submodular. (Section 3).
- To solve WRWD over the probability-aware model $\mathcal{PR}$, we first present a basic greedy method (DpSel) as a baseline, achieving an approximation ratio of $(1 - 1/e)$ (Section 4). Furthermore, we propose a matrix-based greedy method (MatrixSel) and a bound-based greedy method (BoundSel), which significantly reduce the practical cost of DpSel while achieving the same approximation ratio (Section 5).
- By constructing the counterpart graph of G, we adapt DpSel, MatrixSel and BoundSel to solve WRWD over the cost-aware random walk model $\mathcal{CR}$ (Section 6).
- We conduct extensive experiments on several real-world datasets. The results validate the effectiveness, time and space efficiency, and scalability of our methods (Section 7).

2 RELATED WORK

One closely related work is the Random-walk Domination over unweighted graphs [23], which studies only a specific case of our problem. In particular, it assumes that the weight of all edges in G equals to 'one', and the nodes can dominate their neighbors by a naive random walk (Section 3.1) within a given threshold L. However, we cannot ignore the weight information of the graph. For example, we cannot ignore the different energy cost between different nodes in WSNs, which can be adjusted in many different ways [35, 38, 39]. Compared with [23], our WRWD generalizes this problem by taking the weights between edges into consideration where nodes can dominate their neighbors by a weighted random walk (Section 3.1) within a given total budget B.

Influence Maximization (IM) is another problem that is related to our study. Given a social network G, IM aims to find a k-size subset of all nodes in a social network that could maximize the spread of influence, where influence is propagated in the network according to a stochastic cascade model, such as the Independent Cascade (IC) model or the Linear Threshold (LT) model. Under both models, IM has been proven as NP-hard. Kempe et al. [17] show that a greedy algorithm can return an approximate result within the factor of $(1-1/e)$. Then the key challenge lies in how to calculate the influence of the sets efficiently and thus, a plethora of algorithms [9, 10, 13, 30, 31, 34] have been proposed to achieve speedups. Although IM and WRWD share a similar ultimate goal of maximizing influence, the influence models adopted are dramatically different. In IM, the influence models usually assume that nodes in a network can only be passively influenced by their neighbors. In contrast, the random walk model in our problem assumes that nodes in a network can also contact other nodes actively by means of browsing behaviors in social networks or routing in WSNs, etc. Moreover, the IC and LT influence models do not consider the propagation cost from one node to another, even though it is demonstrably important (e.g., the propagation costs in WSNs in Figure 1) and cannot be ignored in our WRWD.

Our problem is also related to the dominating set problem in graphs. Both problems aim to select a subset in the graph to dominate the remaining nodes. The dominating set problem has been well studied in [14] and various extensions of this problem [12, 18, 35] have been proposed to meet different scenarios. However, a core difference is on the definition of domination that leads to different computational challenges. In the dominating set problem, the nodes deterministically dominate their immediate neighbors or neighbors with a distance less than a threshold. Hence it can traverse the graph once to get the dominating set of each node. Then it only needs to update the dominating set of nodes that have an intersection with the dominating set of the selected nodes in each round. However, in WRWD the nodes can dominate their neighbors by a weighted random walk within a given budget B, which lays a burden of updating the probability in each round.

3 PROBLEM FORMULATION

In this paper, we assume that an edge-weighted connected graph is represented by $G = (V, E, wt)$, where V is the set of nodes, E is the set of edges, and wt is the weighting function that maps each edge $e \in E$ to a positive number $wt(e)$, i.e., $wt(e) > 0$. We first formally define three random walk models. Then, we formally define the WRWD problem and analyze the submodularity of the objective function of WRWD. To facilitate our presentation, all frequently used notation are listed in Table 1. These symbols divided by the dotted line first appear in Section 1, 3 and 4 respectively.

3.1 Random Walk Models

In this section, we introduce a naive random walk model and two weighted random walk models as preliminaries, which will be used in illustrating our problem and solutions.

Naive Random Walk Model ($\mathcal{NR}$). Here, let $wt(e) = 1$ for all $e \in E$, then $G = (V, E, wt)$ is an unweighted graph. Given a node $u \in V$, a naive random walk [29] starting from u picks a neighbor of u uniformly at random and moves to this neighbor, and then follows this way recursively. This process can be viewed as a Markov Chain, and each element p_{uv} in the transition matrix $\mathbf{P}$ is given by:

$$p_{uv} = \begin{cases} 1/d_u & u \text{ is a neighbour of } v \\ 0 & \text{otherwise} \end{cases}$$

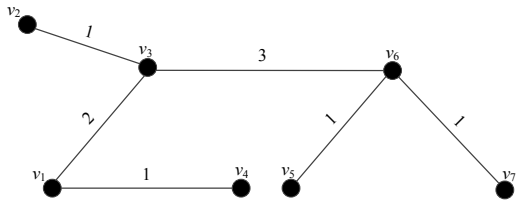


Figure 2: An instance of a cost graph G.

Table 1: Notations for problem formulation and solutions

Symbol	Description
$G(V, E, wt)$	An edge-weighted graph with $n = \|V\|$ nodes, $m = \|E\|$ edges and wt is the weighting function
B	The budget constraint of random walks
$S, \overline{S}$	S is a subset of V and $\overline{S} = V \backslash S$
$\mathbf{P}$	The transition matrix of G
C_{uS}	The expected cost for u to hit S
w_{uv}	A hitting path from u to v
w_{uS}	A random walk starting from u and first hitting a given set S
$c(w_{uv})$	The transition cost from u to v.
$\mathcal{G}(S)$	The object function of our problem
$\varphi_v(S)$	The marginal gain of v to S
$\mathcal{W}_u$	The set of all possible random walk start from u
$\mathbf{Q}_{\overline{S}}$	The non-trap matrix for G, i.e., $\mathbf{Q}_{\overline{S}} = \mathbf{P}$, if $S = \phi$
$\mathbf{Q}^t_{\overline{S}}$	The non-trap matrix of $\mathbf{P}^t$
$\mathbf{N}_{\overline{S}}$	The fundamental matrix of $\mathbf{Q}_{\overline{S}}$

where p_{uv} is the transition probability from u to v, and d_u is the out-degree of u.

Given an arbitrary random walk $w_u = (n_0, n_1, ...)$, where $(...)$ represents a sequence of nodes, starting from $n_0 = u$ and a node $v \in V$, we say that u hits v at step k, if w_u first visits v after k walk steps. We define w_{uv} as a hitting path of u to v, if $w_{uv} = (n_0, n_1, ..., n_k)$, $u = n_0$, and $v = n_k$, for $v \neq n_0, n_1 ..., n_{k-1}$. Let $\mathcal{W}_{uv}$ denote the set of all possible w_{uv}, the possibility of w_{uv} being generated is computed by $\mathcal{P}(w_{uv}) = \prod_{i=0}^{k-1} 1/d_i$. Intuitively, the expected number of steps C_{uv} for u to hit v can be written as $C_{uv} = \sum_{w_{uv} \in \mathcal{W}_{uv}} |w_{uv}| \mathcal{P}(w_{uv})$, where $|w_{uv}|$ is the length of w_{uv}. Note that we define $|w_{uv}| = 0$ if u equals v and the expected number of steps means the expected cost in the unweighted graph.

This equation can be easily generalized to a node-set version. Let w_{uS} be a random walk starting from u and first hitting a node $v \in S$, such that, $w_{uS} = (u, v_1, ..., v_k)$ for $u, v_1 ..., v_{k-1} \notin S$ and $v_k \in S$. C_{uS} can be defined as:

$$C_{uS} = \sum_{w_{uS} \in \mathcal{W}_{uS}} |w_{uS}| \mathcal{P}(w_{uS}) \tag{1}$$

Note that $\mathcal{W}_{uS}$ denotes the set of all possible w_{uS}, but it is not equal to $\bigcup_{v \in S} \mathcal{W}_{uv}$. In Figure 2, let $u = v_1, v = v_6, S = \{v_3, v_6\}$ and $w_{uv} = (v_1, v_3, v_6)$. Although $w_{uv} \in \mathcal{W}_{uv}$, w_{uv} is not contained by $\mathcal{W}_{uS}$ since it has already hit S (node v_3) before hitting v_6.

As mentioned in Section 1, in most (if not all) applications, the edge weights are not equal but indicate different semantic meanings. Thus, we introduce the following two weighted random walk models.

Probability-aware Random Walk Model ($\mathcal{PR}$). In this model, the edge weight represents the probability that a node chooses or affects its neighbors, which is useful in influence maximization [17, 32], weighted pagerank [37] and personalized pagerank [25], to name a few. Compared to the naive random walk model, the transition probability from u to v, p_{uv}, will be rewritten as follows:

$$p_{uv} = \begin{cases} wt(e_{uv}) / \sum_{n \in Ne(u)} wt(e_{un}) & u \text{ is a neighbor of } v \\ 0 & \text{otherwise} \end{cases}$$

where $Ne(u)$ is the neighbor set of u.

Similar to $\mathcal{NR}$, we have the following definitions. The possibility of w_{uv} being generated is computed by $\mathcal{P}(w_{uv}) = \prod_{i=0}^{k-1} p_{n_i n_{i+1}}$. The expected number of steps C_{uv} for u to hit v can be written as $C_{uv} = \sum_{w_{uv} \in \mathcal{W}_{uv}} |w_{uv}| \mathcal{P}(w_{uv})$. The expected cost C_{uS} for u to hit S can be computed as:

$$C_{uS} = \sum_{w_{uS} \in \mathcal{W}_{uS}} |w_{uS}| \mathcal{P}(w_{uS}) \tag{2}$$

Cost-aware Random Walk Model ($\mathcal{CR}$). In this model, the edge weight indicates the cost of moving a node to its neighbor, which is useful for road networks [36] and wireless sensor networks [35, 38]. Therefore, we have: $\mathcal{P}(w_{uv}) = \prod_{i=0}^{k-1} 1/d_i$. As mentioned before, w_{uv} is a hitting path of u to v, if $w_{uv} = (n_0, n_1, ..., n_k)$, $u = n_0$ and $v = n_k$, for $v \neq n_0, n_1 ..., n_{k-1}$. Let $c(w_{uv}) = \sum_{i=0}^{k-1} wt(e_{n_i n_{i+1}})$ denote the total cost of w_{uv}.

Let C_{uS} be the expected hitting cost of w_{uv}. Similar to Equation 1, it can be computed as:

$$C_{uS} = \sum_{w_{uS} \in \mathcal{W}_{uS}} c(w_{uS}) \mathcal{P}(w_{uS}) \tag{3}$$

In this paper, we use a constant budget B to bound the length or total cost of random walk, and hence C^B_{uS} is used to denote the value of C_{uS} bounded by budget B.

Example 1. *Figure 2 shows a simple example. Let $B = 4$, $u = v_1$, $S = \{v_6\}$ and $w_{uS} = (v_1, v_3, v_6)$. For $\mathcal{PR}$, we have $\mathcal{P}(w_{uS}) = 1 * (2/3) * (3/6) = 1/3$ and $|w_{uS}| = 3$. For $\mathcal{CR}$, we have $\mathcal{P}(w_{uS}) = 1 * (1/2) * (1/3) = 1/6$ and $c(w_{uS}) = 4$ because w_{uS} cannot visit S within $B = 4$.*

3.2 Problem Definition and Analysis

To this end, our goal is to minimize the total costs of the remaining nodes to access S through the weighted random walk. Formally, the problem is formulated as $min \sum_{u \in V \backslash S} C^B_{uS}$ such that $|S| \leq k$. It is easy to verify that the above optimization problem is equivalent to the following one.

Definition 1. ***(Weighted Random Walk Domination (WRWD)).*** *Given a weighted graph $G(V, E)$ and a budget $B > 0$, WRWD aims to find a k-size set $S \subseteq V$ that can maximize the budget B to be saved:*

$$\mathcal{G}(S) = \sum_{u \in V} (B - C^B_{uS}) \tag{4}$$

Note that we use P-WRWD and C-WRWD to denote the WRWD over $\mathcal{PR}$ and $\mathcal{CR}$, respectively. Next, we prove that the objective function of WRWD is monotone and submodular.

Theorem 1. *The objective function $\mathcal{G}(S)$ of WRWD is monotone and submodular.*

Proof. We do not prove the monotonicity of $\mathcal{G}(S)$ since it is very straightforward. It remains to show that $\mathcal{G}(S)$ is submodular based on two weighted random walk models.

Let $S \subseteq T \subset V$ and v be a node in $V \backslash T$. According to [26], $\mathcal{G}(S)$ is submodular if it satisfies: $\mathcal{G}(S \cup v) - \mathcal{G}(S) \geq \mathcal{G}(T \cup v) - \mathcal{G}(T)$. To facilitate the proof, we define $S_v = S \cup v$ and $\varphi_v(S) = \mathcal{G}(S \cup v) - \mathcal{G}(S)$. Let $\mathcal{W}_u$ denote the set of all possible random walks start from node u.

$\underline{\mathcal{PR}}$: Intuitively, C^B_{uS} can be rewritten as a sum of the length functions $|.|$, i.e. $C^B_{uS} = \sum_{w_u \in \mathcal{W}_u} |w_{uS}| \mathcal{P}(w_u)$, where we set $|w_{uS}| = B$ if w_u can not access S within B. Then, we have:

$$\begin{aligned}\varphi_v(S) &= \sum\nolimits_{u \in V}(B - C^B_{uS_v}) - \sum\nolimits_{u \in V}(B - C^B_{uS}) \\ &= \sum\nolimits_{u \in V} C^B_{uS} - \sum\nolimits_{u \in V} C^B_{uS_v} \\ &= \sum\nolimits_{u \in V} \sum\nolimits_{w_u \in \mathcal{W}_u} |w_{uS}| \mathcal{P}(w_u) \\ &\quad - \sum\nolimits_{u \in V} \sum\nolimits_{w_u \in \mathcal{W}_u} |w_{uS_v}| \mathcal{P}(w_u) \\ &= \sum\nolimits_{u \in V} \sum\nolimits_{w_u \in \mathcal{W}_u} \mathcal{P}(w_u)(|w_{uS}| - |w_{uS_v}|)\end{aligned}$$

Hence, we have:

$$\begin{aligned}\varphi_v(S) - \varphi_v(T) &= \sum\nolimits_{u \in V} \sum\nolimits_{w_u \in \mathcal{W}_u} \mathcal{P}(w_u)(|w_{uS}| - |w_{uS_v}|) \\ &\quad - \sum\nolimits_{u \in V} \sum\nolimits_{w_u \in \mathcal{W}_u} \mathcal{P}(w_u)(|w_{uT}| - |w_{uT_v}|) \\ &= \sum\nolimits_{u \in V} \sum\nolimits_{w_u \in \mathcal{W}_u} (\mathcal{P}(w_u) \\ &\quad * ((|w_{uS}| - |w_{uS_v}|) - (|w_{uT}| - |w_{uT_v}|)))\end{aligned} \tag{5}$$

To show the submodularity of $\mathcal{G}(S)$, we first prove Inequality 6.

$$((|w_{uS}| - |w_{uS_v}|) - (|w_{uT}| - |w_{uT_v}|)) \geq 0 \tag{6}$$

Since $S \subseteq T$, we have $|w_{uT}| \leq |w_{uS}|$. Then, we discuss two cases about the length function $|.|$: (1) suppose that w_u visits v before T, we have $|w_{uv}| < |w_{uT}| \leq |w_{uS}|$ and $|w_{uT_v}| = |w_{uS_v}| = |w_{uv}|$. Thus, $|w_{uS_v}| - |w_{uS}| \leq |w_{uT_v}| - |w_{uT}| < 0$; (2) the remaining case is that w_u visits v after T. At this case, we have $|w_{uS_v}| - |w_{uS}| \leq |w_{uT_v}| - |w_{uT}| = 0$. These discussions show the correctness of Inequality 6.

Based on Equation 5 and Inequality 6, we have $\varphi_v(S) - \varphi_v(T) \geq 0$ and thus $\mathcal{G}(S)$ is a submodular function based on $\mathcal{PR}$.

$\underline{\mathcal{CR}}$: Similar to $\mathcal{PR}$, we can rewrite C^B_{uS} as $\sum_{w_u \in \mathcal{W}_u} c(w_{uS}) \mathcal{P}(w_u)$, where $c(w_{uS}) = B$ if w_u cannot access S within B. Then, we have:

$$\varphi_v(S) = \sum\nolimits_{u \in V} \sum\nolimits_{w_u \in \mathcal{W}_u} \mathcal{P}(w_u)(c(w_{uS}) - c(w_{uS_v}))$$

Similar to Equation 5, we have:

$$\begin{aligned}\varphi_v(S) - \varphi_v(T) = \sum\nolimits_{u \in V} \sum\nolimits_{w_u \in \mathcal{W}_u} (\mathcal{P}(w_u) \\ * ((c(w_{uS}) - c(w_{uS_v})) - (c(w_{uT}) - c(w_{uT_v}))))\end{aligned} \tag{7}$$

To show the submodularity of $\mathcal{G}(S)$, we first prove Inequality 8.

$$(c(w_{uS}) - c(w_{uS_v})) - (c(w_{uT}) - c(w_{uT_v})) \geq 0 \tag{8}$$

Since $S \subseteq T$, we have $c(w_{uT}) \leq c(w_{uS})$. Then, we also discuss two cases about the cost function $c(.)$: (1) suppose that w_u visits v before T, we have $c(w_{uv}) < c(w_{uT}) \leq c(w_{uS})$ and $c(w_{uT_v}) = c(w_{uS_v}) = c(w_{uv})$. Thus, $c(w_{uS_v}) - c(w_{uS}) \leq c(w_{uT_v}) - c(w_{uT}) < 0$; (2) the remaining case is that w_u visits v after T. At this case, we have $c(w_{uS_v}) - c(w_{uS}) \leq c(w_{uT_v}) - c(w_{uT}) = 0$. These discussions show the correctness of Inequality 8.

Based on Equation 7 and Inequality 8, we have $\varphi_v(S) - \varphi_v(T) \geq 0$ and thus $\mathcal{G}(S)$ is a submodular function based on $\mathcal{CR}$. □

Algorithm 1: DpSel (G, B, k)

1.1 **Input:** a graph G, a budget B and k
1.2 **Output:** a node set S
1.3 Initialize $S \leftarrow \phi$
1.4 **for** $i = 1$ to k **do**
1.5 Select a node $v = \arg\max_{v \in V \backslash S}(\mathcal{G}(S \cup v) - \mathcal{G}(S))$
1.6 $S \leftarrow S \cup v$
1.7 **return** S

4 OUR FRAMEWORK

Throughout Sections 4-5, we describe our solutions for P-WRWD, i.e. the WRWD problem over the probability-aware random walk model, and later in Section 6 we discuss how to extend our solutions to address C-WRWD.

In particular, we first present a basic greedy baseline, which is extended from the greedy method designed for an unweighted graph to P-WRWD in [23]. However, the bottleneck of our problem is how to efficiently compute the marginal gain of the node (see complexity analysis in Section 4.1). To overcome this challenge, we present some efficient methods to compute the marginal gain of the node (see Section 4.2 and Section 5), which effectively solve P-WRWD. Moreover, compared to the sampling-based methods proposed in [23] for the unweighted random walk domination problem, our matrix-based method MatrixSel based on our proposed trapped model can achieve a tight approximation ratio of $(1 - 1/e)$. What's more, as we will see in Section 6, our approaches can work with both the probability-aware random walk model and the cost-aware random walk model to support different application needs in general.

4.1 A Greedy Baseline

It is not possible to compute C^B_{uS} directly from Equation 1 because the size of $\mathcal{W}_{uS}$ should be exponential. Fortunately, Li et al. [23] show that C^B_{uS} based on $\mathcal{NR}$ can be rewritten in a recursive manner as follows.

Theorem 2. *Given a random walk starting from u. The expected number of steps C^{B}_{uS} can be recursively computed as*

$$C^{\mathrm{B}}_{uS} = \begin{cases} 0 & u \in S \\ 1 & u \notin S, B = 1 \\ 1 + \sum\limits_{v \in V} p_{uv} C^{\mathrm{B}-1}_{vS} & u \notin S, B > 1 \end{cases} \tag{9}$$

It is worth highlighting that the second equality in Equation 9 holds because of the bounded constraint, without which the recursion would never stop even when B is smaller than zero. This constraint is an improvement over the method used in [23]. Similar to $\mathcal{NR}$, C^B_{uS} based on $\mathcal{PR}$ also can be computed by Equation 9.

Algorithm 1 shows the pseudo-code of the greedy method (DpSel). In each iteration, it repeatedly selects a node $v \in V \backslash S$ with the largest marginal gain such that $v = \arg\max_{v \in V \backslash S} \{\mathcal{G}(S \cup \{v\}) - \mathcal{G}(S)\}$, for the current set S. For P-WRWD, the computation of $\mathcal{G}(S)$ relies on dynamic programming according to Equation 9. Finally, it returns S as the solution when the cardinality of S is equal to k.

Approximation Ratio. This baseline is guaranteed to achieve $(1 - 1/e)$-approximation, as proved by Nemhauser et al. [26].

Complexity Analysis. The time complexity of DpSel is dominated by the time complexity of computing the marginal gain (line 1.5). To simplify the explanation, we define $n = |V|$ and $m = |E|$. The time complexity of DP is $O(Bm)$ ($n < m$), and thus computing the marginal gain takes $O(Bnm)$ time. Since the greedy method has to find the node with the marginal gain in each iteration, the time complexity of Algorithm 1 for P-WRWD is $O(kBn^2m)$.

4.2 A Matrix-Based Solution

For DpSel, the main bottleneck is to repeatedly compute C^B_{uS} in G for each $u \in V \backslash S$. To reduce the cost, we give a trapped model to capture the process of a random walk to S, and propose a matrix-based greedy algorithm, MatrixSel (see Algorithm 2), to solve P-WRWD.

4.2.1 A Trapped Model.

Definition 2. *A node $u \in V$ is called a trap if any random walk stays at u once it visits u.*

Given G and a k-size node set S in G, let us consider a random walk w_u that can visit S. Clearly, any walk steps after S is visited will have no effect on C^B_{uS}. Therefore, we can ignore these steps and treat S as a trap set. In this case, it is worth noting that C^B_{uS} is actually equivalent to the expected steps of w_u being trapped at S on G. Intuitively, we can turn each node $v \in S$ as a trap by setting $p_{vu} = 0$ for $u \in V$. As a result, the transition matrix $\mathbf{P}$ of this process can be represented by a $n \times n$ matrix as below:

$$\begin{vmatrix} \mathbf{Q}_{\overline{S}} & \mathbf{R} \\ \mathbf{0} & \mathbf{0}_S \end{vmatrix}$$

where $\mathbf{Q}_{\overline{S}}$ and $\mathbf{R}$ correspond to rows of unselected candidates in $\overline{S} = V \backslash S$; $\mathbf{0}$ and $\mathbf{0}_S$ are both zero matrices. Note that, $\mathbf{Q}_{\overline{S}}$ is equal to $\mathbf{P}$ when $S = \phi$. In this paper, $\mathbf{Q}_{\overline{S}}$ and $\mathbf{0}_S$ are called the *non-trap matrix* and the *trap matrix* respectively.

Let $\mathbf{P}^t$ denote the t^{th} power of the transition matrix $\mathbf{P}$ and $\mathbf{Q}^t_{\overline{S}}$ denote the *non-trap matrix* of $\mathbf{P}^t$. It is well known that p^t_{ij} (the $(i,j)^{th}$ entry of $\mathbf{P}^t$) gives the probability of w_{v_i} visiting v_j at step t. Let $X^{(t)}_{ij}$ ($0 < t \leq B$) be a random variable which is equal to 1 if w_{v_i} reaches v_j at the step t before it is trapped, or 0 otherwise. Then, the expectation of $X^{(t)}_{ij}$ can be computed as:

$$E[X^{(t)}_{ij}] = q^t_{ij} \tag{10}$$

where q^t_{ij} denotes the $(i,j)^{th}$ entry of $\mathbf{Q}^t_{\overline{S}}$.

Based on $\mathbf{Q}_{\overline{S}}$, we introduce the concept of *fundamental matrix* as follows.

Definition 3. *A matrix $\mathbf{N}_{\overline{S}}$ is the fundamental matrix of $\mathbf{Q}_{\overline{S}}$, if* $\mathbf{N}_{\overline{S}} = \mathbf{Q}^1_{\overline{S}} + \mathbf{Q}^2_{\overline{S}} + \ldots + \mathbf{Q}^{\mathrm{B}}_{\overline{S}}$.

Lemma 1. *Given a budget B, $\mathbf{N}_{\overline{S}}[i][j]$ gives the expected number of times (or frequency) that a random walk visits v_j, if it starts at v_i ($v_i, v_j \in \overline{S}$).*

Proof. The expected frequencies of the random walk reaching v_j in the first B steps is equal to the sum of frequencies at the t^{th} step, for $t = 1, 2 \ldots B$. Therefore, we have:

$$E(X^{(1)}_{ij} + X^{(2)}_{ij} + \ldots + X^{(B)}_{ij}) = \sum_{t=1}^{B} q^{(t)}_{ij}$$

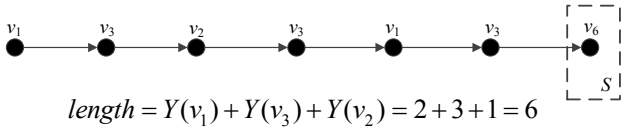


Figure 3: A random walk from v_1 trapped at S, $S = \{v_6\}$.

$$\underbrace{\begin{pmatrix} p_{11} & p_{13} & \cdots & \cdots & p_{1n} \\ p_{31} & p_{33} & \cdots & \cdots & p_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{n1} & p_{n3} & \cdots & \cdots & p_{nn} \end{pmatrix}}_{\mathbf{Q}_{\overline{S'}}} = \underbrace{\begin{pmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} \\ p_{31} & p_{32} & p_{33} & \cdots & p_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{n1} & p_{n2} & p_{n3} & \cdots & p_{nn} \end{pmatrix}}_{\mathbf{Q}_{\overline{S}}}$$

Figure 4: The relationship between $\mathbf{Q}_{\overline{S'}}$ and $\mathbf{Q}_{\overline{S}}$. Here, $S' = S \cup \{v_2\}$.

Since $\mathbf{N}_{\overline{S}} = \mathbf{Q}^1_{\overline{S}} + \mathbf{Q}^2_{\overline{S}} + \ldots + \mathbf{Q}^B_{\overline{S}}$, we get $\sum_{t=1}^{B} q^{(t)}_{ij} = \mathbf{N}_{\overline{S}}[i][j]$. Thus, the lemma is proved. □

4.2.2 A Matrix-based Greedy Method.

Observation 1. *Given a random walk w_{uS}, the length of w_{uS} equals to $\sum_{v \in \overline{S}} Y(v)$, where $Y(v)$ denotes the frequency of w_{uS} reaching v.*

This section introduces the matrix-based greedy method and the following observation bridges $\mathbf{N}_{\overline{S}}$ to our solution.

Figure 3 gives an example to explain this observation. In this figure, the length of the given random walk is 6 which equals the sum of the frequencies of $v_1, v_2, v_3, v_4, v_5, v_7$ in Figure 2. Here, since $Y(v_4)$, $Y(v_5)$ and $Y(v_7)$ are equal to zero, we omit them in the equation for brevity.

Given G, Lemma 1 shows that $\mathbf{N}_{\overline{S}}[i][j]$ is the expected frequencies of w_{v_i} visiting v_j in G. By writing this statement in matrix form, we obtain the following lemma immediately.

Lemma 2. *Given S and the fundamental matrix $\mathbf{N}_{\overline{S}}$. Let $\Phi_{\overline{S}}$ be a vector whose u^{th} ($u \notin S$) entry equals C^{B}_{uS}. Then, we have $\Phi_{\overline{S}} = \mathbf{N}_{\overline{S}} I$, where I is an identity vector.*

According to Equation 4, $\mathcal{G}(S)$ can be obtained by summing up the rows of the fundamental matrix $\mathbf{N}_{\overline{S}}$. Based on Lemma 2, we introduce a matrix-based method, which is shown in Algorithm 2. Initially, $S = \phi$. In each iteration, let $S' = \{S \cup v_j\}$ (line 2.7). MatrixSel picks the node v_j that can maximize $\mathcal{G}(S')$ according to Lemma 2 (lines 2.5 to 2.13). In particular, the algorithm first calls FastNI to compute $\Phi_{\overline{S'}}$ (line 2.8). Then, it computes the marginal gain by the sum of all elements in $\Phi_{\overline{S'}}$ (line 2.9). Here, *mar* records the gain of $\mathcal{G}(S')$. If v_j maximizes $\mathcal{G}(S')$, we add v_j into S for the next round.

Computing $\mathbf{N}_{\overline{S}}\mathrm{I}$. To obtain $\mathbf{N}_{\overline{S}}\mathrm{I}$, a naive solution is to calculate $\mathbf{N}_{\overline{S}} = \{\mathbf{Q}^1_{\overline{S}} + \mathbf{Q}^2_{\overline{S}} + \mathbf{Q}^3_{\overline{S}} + \ldots + \mathbf{Q}^B_{\overline{S}}\}$ directly. However, it is prohibitively expensive to directly compute $\mathbf{Q}^t_{\overline{S}}$ for $t = 1, 2, \ldots, B$, since one matrix multiplication needs at least $O(n^{2.37})$ time [11]. Therefore, we rewrite $\mathbf{N}_{\overline{S}}\mathrm{I}$ as $(\mathbf{Q}^1_{\overline{S}}\mathrm{I} + \mathbf{Q}^2_{\overline{S}}\mathrm{I} + \mathbf{Q}^3_{\overline{S}}\mathrm{I} + \ldots + \mathbf{Q}^B_{\overline{S}}\mathrm{I})$. Note that, $\mathbf{Q}^t_{\overline{S}}\mathrm{I}$ is a vector which can be represented by a product of a matrix and a vector, i.e., $\mathbf{Q}_{\overline{S}}(\mathbf{Q}^{t-1}_{\overline{S}}\mathrm{I})$. This implies that we can recursively calculate $\mathbf{Q}^t_{\overline{S}}\mathrm{I}$ and obtain $\mathbf{N}_{\overline{S}}\mathrm{I}$ without computing $\mathbf{N}_{\overline{S}}$ directly. Clearly, computing a matrix-vector product is much faster than executing a matrix-matrix multiplication, and thus this rewriting can greatly reduce the computation cost. Moreover, G is generally sparse in real world. To avoid dealing with the large number of zeroes, we

Algorithm 2: MatrixSel (G, B, k)

2.1 **Input:** a graph $G(V, E)$, a budget B and k
2.2 **Output:** a set S
2.3 $S \leftarrow \phi$; $curGain \leftarrow 0$
2.4 **for** $i = 1$ *to* k **do**
2.5 $\quad opt \leftarrow 0$; $u \leftarrow \phi$
2.6 $\quad$ **foreach** *node* $v_j \in V \backslash S$ **do**
2.7 $\quad\quad S' \leftarrow S \cup v_j$
2.8 $\quad\quad \Phi_{\overline{S'}} \leftarrow$ FastNI(G, B, S')
2.9 $\quad\quad mar \leftarrow \sum_{q \in V \backslash S'} (B - \Phi_{\overline{S'}}[q]) - curGain$
2.10 $\quad\quad$ **if** $opt < mar$ **then**
2.11 $\quad\quad\quad opt \leftarrow mar$
2.12 $\quad\quad\quad u \leftarrow v_j$
2.13 $\quad S \leftarrow S \cup u$
2.14 $\quad curGain \leftarrow curGain + opt$
2.15 **return** S

Algorithm 3: FastNI (G, B, S)

3.1 **Input:** a graph G, a budget B and a set S
3.2 **Output:** a n-dimensional vector Φ
3.3 Initialize vector Φ and γ' with 0.
3.4 **for** $t = 1$ *to* B **do**
3.5 $\quad$ **if** $t = 1$ **then**
3.6 $\quad\quad$ **foreach** *node* $v_i \in \overline{S}$ **do**
3.7 $\quad\quad\quad \gamma'[i] = 1$
3.8 $\quad$ **else**
3.9 $\quad\quad$ **foreach** *node* $v_i \in \overline{S}$ **do**
3.10 $\quad\quad\quad \gamma[i] \leftarrow \sum_{v_j \in Ne(v_i)} q_{ij} \gamma'[j]$
3.11 $\quad \Phi \leftarrow \Phi + \gamma$
3.12 $\quad \gamma' \leftarrow \gamma$
3.13 **return** Φ

store G as an adjacency list and introduce Algorithm 3 to speed up the computation of $\mathbf{N}_{\overline{S}}\mathbf{I}$.

In this Algorithm, γ and γ' are two temporary n-dimensional vectors used to record the values of $\mathbf{Q}_{\overline{S}}^{t}\mathbf{I}$ and $\mathbf{Q}_{\overline{S}}^{(t-1)}\mathbf{I}$ respectively. At $t = 1$, Algorithm 3 initializes γ' ($\gamma' = \mathbf{Q}^1\mathbf{I}$). When $t > 1$, it traverses the neighbors of each node in $\overline{S}$ and compute γ based on γ' (lines 3.8 to 3.9). Here $Ne(v_i)$ is the neighbor set of v_i. Finally, by summing up $\gamma = \mathbf{Q}_{\overline{S}}^{t}\mathbf{I}$ for $t = 1, 2, \ldots B$, it returns Φ as the result.

Approximation Ratio. According to Lemma 2, $\mathcal{G}(S') = \sum_{q \in V \backslash S'} (B - \Phi_{\overline{S'}}[q])$. In Algorithm 2, $curGain = \mathcal{G}(S)$ and mar is the marginal gain of the node v_j. In each greedy iteration, Algorithm 2 chooses the node with the maximum marginal gain to set S (lines 2.5 to 2.13). The whole process is consistent with the greedy strategy and the objective function $\mathcal{G}(S)$ of WRWD is monotone and submodular. Then MatrixSel is guaranteed to achieve $(1 - 1/e)$-approximation.

Time Complexity. FastNI computes $\mathbf{N}_{\overline{S}}\mathbf{I}$ in B iterations. For each iteration, it needs to traverse the neighbors of each node in $\overline{S}$ one time. Based on the adjacency list, finding the neighbor sets totally takes $O(m)$ time and thus the time complexity of FastNI is $O(Bm)$. Moreover, Algorithm 2 needs to compute $\mathcal{G}(v_j, S)$ for each candidate $v_j \in \overline{S}$ in order to select a node into S. Therefore, MatrixSel needs to invoke Algorithm 3 $O(n)$ time in each selection. By repeating this process k times to obtain S, the total complexity of Algorithm 2 for P-WRWD is $O(kBnm)$.

Space Complexity. It costs $O(n + 2m)$ space for MatrixSel and FastNI to save the undirected graph G by an adjacency list. Otherwise, several n-dimension vectors are used in MatrixSel. In total, the space complexity of Algorithm 2 for P-WRWD is $O(n + 2m)$.

5 AN EFFICIENT BOUND-BASED SOLUTION

As mentioned above, each iteration of MatrixSel requires to invoke FastNI for $O(n)$ times to select a candidate with the largest marginal gain into S. However, we note that the candidate set contains many insignificant nodes which will not be selected as seeds. To avoid the unnecessary computations, this section introduces a bound-based method BoundSel to prune the insignificant candidates. By making use of the submodular property we have proved earlier (in Theorem 1), we first employ a cost-effective lazy forward algorithm. The idea behind it is that the marginal gain of a node in the current iteration should not be more than that in previous iterations, and thus the number of the gain computations can be greatly reduced. However, this optimization cannot prune any candidate in the first round of selection, which results in n times of the gain computations. Here n denotes the number of nodes. To overcome this issue, we introduce a bound estimation method to further reduce the number of the gain computations in the first iteration.

Algorithm 4 presents the pseudo-code of BoundSel. Given S and $v \in \overline{S}$, $\mathcal{G}(v, S)$ denotes the marginal gain of v to S. To select the optimal candidate in each iteration, we first calculate the upper bound of $\mathcal{G}(v, S)$ for each $v \in \overline{S}$, denoted by $\widehat{\mathcal{G}}(v, S)$, and we use the vector α to save them (Line 4.4). To facilitate selecting seeds, we maintain a sorted list for each $v \in \overline{S}$ with the upper bound $\widehat{\mathcal{G}}(v, S)$ in descending order (Line 4.5). Then, we select the first node v_1 and compute $\mathcal{G}(v_1, S)$ by invoking Algorithm 3. Intuitively, $\mathcal{G}(v_1, S)$ can be treated as a pruning bound Pb to prune insignificant candidates. Next, we go through the node $v_j \in V/S$. If $\alpha_j \leq Pb$, we add the current node into S and end the traversal. Otherwise, we invoke Algorithm 3 to compute α_j, and update the current node and Pb if $\alpha_j > Pb$ (Lines 4.11 to 4.23).

To facilitate the understanding, we show the workflow of BoundSel in Figure 5. If $Pb \geq \widehat{\mathcal{G}}(v_2, S)$, then v_2 can be pruned safely since the inequation guarantees that v_1 is better than v_2. Otherwise, we invoke Algorithm 3 to compute $\mathcal{G}(v_2, S)$. If v_2 is better than v_1, we update Pb by $\mathcal{G}(v_2, S)$. By repeatedly adjusting Pb, we should be able to skip a large number of insignificant candidates and dramatically reduce the invocations of FastNI.

Upper Bound Estimation. Now we have a distribution $Z_v = \{Z_v^1, Z_v^2, ..., Z_v^B\}$ for v, and Z_v^t as the number of nodes that first reach v at the step t. In this phase, since $S = \emptyset$, we have $\sum_{t=1}^{B} Z_v^t = n - 1$. Intuitively, $\mathcal{G}(v, S) = \mathcal{G}(v, \emptyset) = nB - \sum_{t=1}^{B} tZ_v^t$. Hence, we estimate $\widehat{\mathcal{G}}(v, S)$ by establishing the lower bound of $\sum_{t=1}^{B} tZ_v^t$. To estimate the lower bound of $\sum_{t=1}^{B} tZ_v^t$, we first introduce Lemma 3.

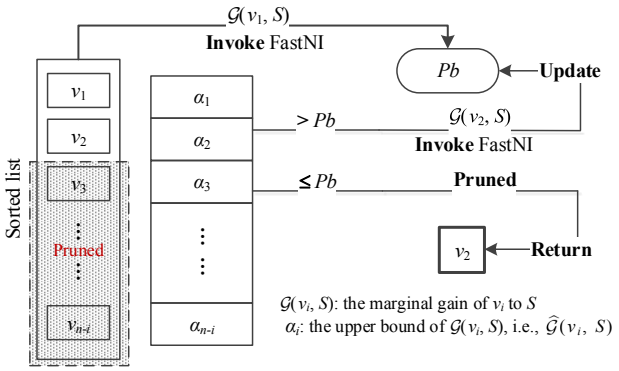


Figure 5: An example for the workflow of BoundSel.

Algorithm 4: BoundSel (G, B, k)

```
4.1  Input: a graph G(V, E), B and k
4.2  Output: a set S
4.3  S ← φ; curGain ← 0
4.4  α ←EstimateBound(G, B)
4.5  Sort V order by α Desc
4.6  for i = 1 to k do
4.7      S' ← S ∪ v_1
4.8      Φ_{S'} ← FastNI(G, B, S')
4.9      α_1 ← Σ_{q∈V\S'} (B − Φ_{S'}[q]) − curGain
4.10     current ← v_1; Pb ← α_1
4.11     foreach node v_j ∈ V\S do
4.12         if α_j ≤ Pb then
4.13             S ← S ∪ current
4.14             curGain ← curGain + Pb
4.15             Break
4.16         else
4.17             S' ← S ∪ v_j
4.18             Φ_{S'} ← FastNI(G, B, S')
4.19             α_j ← Σ_{q∈V\S'} (B − Φ_{S'}[q]) − curGain
4.20             if α_j > Pb then
4.21                 current ← v_j; Pb ← α_j
4.22             else
4.23                 continue
4.24     Resort V
4.25 return S
```

Lemma 3. *Let F_v^t denote the expected number of nodes in $\overline{S}$ that can reach v at the step t ($t \leq B$). For any integer $T \in$(1,B], we have:*

$$\sum_{t=1}^{B} tZ_v^t \geq \sum_{t=1}^{T-1} tF_v^t + T(n-1-\sum_{t=1}^{T-1} F_v^t) \tag{11}$$

Proof. We discuss the two cases of a random walk w reaching v at the step t: (1) this is the first time w reaches v; (2) w reached v at the previous step. According to the definition, Z_v^t only contains the first case, but F_v^t contains the two cases. Therefore, we have $F_v^t \geq Z_v^t$. To show the correctness of Equation 11, we have:

$$\begin{aligned}&\sum_{t=1}^{B} tZ_v^t - (\sum_{t=1}^{T-1} tF_v^t + T(n-1-\sum_{t=1}^{T-1} F_v^t)) \\ &= \sum_{t=1}^{B} tZ_v^t - T(n-1) - \sum_{t=1}^{T-1}(t-T)F_v^t\end{aligned} \tag{12}$$

Note that the next equation holds as $\sum_{t=1}^{B} Z_v^t = n-1$. By applying this equation to Equation 12, we have:

$$\begin{aligned}&\sum_{t=1}^{B} tZ_v^t - T(n-1) - \sum_{t=1}^{T-1}(t-T)F_v^t \\ &= \sum_{t=1}^{B}(t-T)Z_v^t - \sum_{t=1}^{T-1}(t-T)F_v^t \\ &= \sum_{t=T}^{B}(t-T)Z_v^i - \sum_{t=1}^{T-1}(t-T)(F_v^t - Z_v^t)\end{aligned} \tag{13}$$

Intuitively, $\sum_{t=T}^{B}(t-T)Z_v^t \geq 0$ and $\sum_{t=1}^{T-1}(t-T)(F_v^t - Z_v^t) \leq 0$. Therefore, we have:

$$\sum_{t=1}^{B} tZ_v^t - (\sum_{t=1}^{T-1} tF_v^t + T(n-1-\sum_{t=1}^{T-1} F_v^t)) \geq 0.$$

This completes the proof. □

Based on Lemma 3, $\widehat{\mathcal{G}}(v, S)$ can be estimated according to the following lemma :

Lemma 4. *Given S and a node $v \in \overline{S}$, the upper bound $\widehat{\mathcal{G}}(v, S)$ can be estimated by $(B-\rho)n + \rho + \sum_{t=1}^{\rho-1}(\rho-t)F_v^t$, where ρ is the minimal integer that satisfies $\sum_{t=1}^{\rho} F_v^t \geq n-1$.*

Proof. $\sum_{t=1}^{T-1} tF_v^t + T(n-1-\sum_{t=1}^{T-1} F_v^t)$ increases as T increases until $\sum_{t=1}^{T} F_v^t > n-1$. Therefore, the right side of Equation 11 reaches the maximum at $T = \rho$. By applying this to Equation 11 and $\mathcal{G}(v, S) = nB - \sum_{t=1}^{C} tZ_v^t$, we have:

$$\begin{aligned}\mathcal{G}(v, S) &\leq nB - (\sum_{t=1}^{\rho-1} tF_v^t + \rho(n-1-\sum_{t=1}^{\rho-1} F_v^t)) \\ &= (B-\rho)n + \rho + \sum_{t=1}^{\rho-1}(\rho-t)F_v^t\end{aligned} \tag{14}$$

This completes the proof. □

By Lemma 4, the key to estimate $\widehat{\mathcal{G}}(v, S)$ is how to compute the vector F^t effectively. According to the definition of $\mathbf{Q}_{\overline{S}}^t$, we have $F^t = I\mathbf{Q}_{\overline{S}}^t$. Hence, we can compute $I\mathbf{Q}_{\overline{S}}^t$ in a similar way to compute $\mathbf{Q}_{\overline{S}}^t I$. In particular, we replace line 3.10 in Algorithm 3 with $F_i^t \leftarrow \sum_{v_j \in Ne(v_i)} q_{ji} F_j^{t-1}$ to compute $I\mathbf{Q}_{\overline{S}}^t$. Here $Ne(v_i)$ is the neighbor set of v_i. The pseudo-code for estimating $\widehat{\mathcal{G}}(v, S)$ is presented in Algorithm 5. It computes $I\mathbf{Q}_{\overline{S}}^t$ as mentioned above (Lines 5.5 to 5.9). Next, it checks every node. For node v, if $\sum_{i=1}^{t} F_v^i \geq n-1$ and $\alpha_v = 0$, it sets $\alpha_v = (B-t)n + t + \sum_{i=1}^{t-1}(t-i)F_v^i$ (Lines 5.10 to 5.13). Here, it checks $\alpha_v = 0$ to make sure it only computes α_v once. Finally, it returns α as the result.

Approximation Ratio. BoundSel estimates the upper bound of the marginal gains of the candidate nodes. It prunes the remaining nodes safely when the marginal gain of the current node is not less than the maximum value of the upper bound of the remaining candidate nodes. Then, BoundSel can choose the node with the largest marginal gain in each iteration. Therefore, this pruning strategy does not affect the effectiveness of the method and BoundSel still achieves a $(1-1/e)$ approximation ratio.

Time Complexity. The time complexity of BoundSel is the same as that of MatrixSel at the worst case (as evidenced in Section 4.2.2), but the pruning strategy reduces the running time greatly (as evidenced in Section 7.3).

Space Complexity. BoundSel, EstimateBound and FastNI cost $O(n+2m)$ space to save the undirected graph G by an adjacency list and use several n-dimension vectors to record other temporary

Algorithm 5: EstimateBound(G, B)

5.1 **Input:** a graph G and B
5.2 **Output:** a vector α
5.3 Initialize vector α
5.4 **for** $t = 1$ *to* B **do**
5.5 **if** $t = 1$ **then**
5.6 $F^1 \leftarrow \{1, 1, \ldots, 1\}$
5.7 **else**
5.8 **foreach** *node* $v_i \in V$ **do**
5.9 $F_i^t \leftarrow \sum_{v_j \in Ne(v_i)} q_{ji} F_j^{t-1}$
5.10 **foreach** *node* $v \in V$ **do**
5.11 **if** $\sum_{i=1}^{t} F_v^i \geq n - 1$ **then**
5.12 **if** $\alpha_v = 0$ **then**
5.13 $\alpha_v = (B - t)n + t + \sum_{i=1}^{t-1}(t - i)F_v^i$
5.14 **return** α

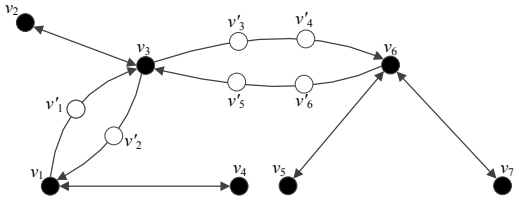


Figure 6: G' of G (Figure 2). v'_1 to v'_6 are redundant nodes which are added by step 2.

results. Therefore, the total space complexity of Algorithm 4 for solving the P-WRWD is $O(n + 2m)$.

6 OUR SOLUTIONS TO C-WRWD

In C-WRWD, we cannot rewrite C_{uS}^B in a recursive manner directly as Equation 9. Then, we show the relationship of C_{uS}^B based on $\mathcal{NR}$ and C_{uS}^B based on $\mathcal{CR}$. Toward this end, we introduce a new concept, *counterpart graph*, to illustrate the connection easily. In particular, a counterpart graph G' can be transformed from G according to Definition 4.

Definition 4. *Given a graph G, we denote the counterpart graph of G as G', and it is constructed in two steps:*
1. initialize G' as an empty graph by deleting all edges in G;
2. if there is an edge e_{ij} in G, we connect v_i and v_j by two opposite directed simple paths, and each path consists of $(wt(e_{ij}) - 1)$ supplemental nodes.

Figure 6 gives an example of G', which is transformed from G in Figure 2. In this example, we use $v'_1, v'_2, ..., v'_6$ to denote the supplemental nodes. From this example, we have the following observation.

Observation 2. *The expected hitting step from u to S (C_{uS}^B based on $\mathcal{NR}$) in G' is equal to the expected hitting cost (C_{uS}^B based on $\mathcal{CR}$) in G.*

This observation indicates that we can obtain C_{uS}^B based on $\mathcal{CR}$ in G by computing C_{uS}^B based on $\mathcal{NR}$ in G'.

6.1 DpSel for C-WRWD

Algorithm 1 shows the pseudo-code of the greedy method for P-WRWD. In each iteration, it repeatedly selects a node $v \in V \backslash S$ with the largest marginal gain, such that $v = \arg\max_{v \in V \backslash S} \{\mathcal{G}(S \cup \{v\}) - \mathcal{G}(S)\}$, to the current set S. When we apply DpSel to C-WRWD, $\mathcal{G}(S)$ can be computed by dynamic programming according to Observation 2 and Equation 9.

Approximation Ratio. This analysis is similar to the analysis of approximation ratio in Section 4.1. We omit it here for brevity.

Complexity Analysis. To simplify the explanation, we define $n = |V|$, $m = |E|$ and $\delta = 1/m \sum_{e \in E} wt(e)$. For $\mathcal{CR}$, observation 2 shows that we can utilize Equation 9 to compute C_{uS}^B by constructing G'. According to Definition 4, G' has $n + (\delta - 1)m$ nodes and δm edges. Similar to $\mathcal{PR}$, the time complexity of DP is $O(\delta Bm)$ and computing the marginal gain takes $O(\delta Bnm)$ time, because we only need to compute C_{uS}^B for $u \in V$. Therefore, the time complexity of Algorithm 1 for C-WRWD is $O(\delta kBn^2 m)$.

6.2 MatrixSel for C-WRWD

Given G', Lemma 1 shows that $N_{\overline{S}}[i][j]$ is the expected frequencies of w_{v_i} visiting v_j in G'. By using Observation 1, we see that C_{uS}^B in G' equals the sum of the i^{th} row of $N_{\overline{S}}$, which actually is C_{uS}^B in G. Similar to the way of computing C_{uS}^B over $\mathcal{PR}$, we can call FastNI to compute C_{uS}^B over $\mathcal{NR}$ in G'. Therefore, we can apply MatrixSel to C-WRWD by replacing $\Phi_{\overline{S'}} \leftarrow$ FastNI(G, B, S') with $\Phi_{\overline{S'}} \leftarrow$ FastNI(G', B, S') (MatrixSel line 2.8).

Approximation Ratio. This analysis is similar to the analysis of approximation ratio in Section 4.2.2. We omit it here for brevity.

Time Complexity. Similar to the analysis of time complexity in Section 4.2.2, FastNI takes $O(\delta Bm)$ time to C_{uS}^B in G'. In addition, MatrixSel needs to invoke FastNI $O(n)$ times in each selection and repeats k round selections. Therefore, the total complexity of MatrixSel for C-WRWD is $O(\delta kBnm)$.

Space Complexity. Since we transfer the input graph G to G' and G' has $O(n + 2(\delta - 1)m)$ nodes and $O(2\delta m)$ edges, MatrixSel and FastNI cost $O(n + 4\delta m)$ space to save the graph G'. Thus, in total, the space complexity of Algorithm 2 for solving the C-WRWD is $O(n + 4\delta m)$.

6.3 BoundSel for C-WRWD

Similarly, we can use the above method to estimate the lower bound of $\sum_{t=1}^{B} tZ_v^t$ in G', which equals the lower bound of $\sum_{t=1}^{B} tZ_v^t$ in G. Therefore, we can get the upper bound of the marginal gain of node v. In particular, we replace line 4.4 by $\alpha \leftarrow$ **EstimateBound**(G', B), and line 4.9 and 4.19 by $\Phi_{\overline{S'}} \leftarrow$ FastNI(G, B, S'). By doing so, we can apply BoundSel to C-WRWD.

Approximation Ratio. This analysis is similar to the analysis of approximation ratio in Section 5. We omit it here for brevity.

Time Complexity. Similar to the analysis of the time complexity in Section 5, the complexity of BoundSel for C-WRWD is the same as that of MatrixSel for C-WRWD at the worst case, but the pruning strategy reduces the running time greatly (as evidenced in Section 7.3).

Space Complexity. Since G' has $O(n + 2(\delta - 1)m)$ nodes and $O(2\delta m)$ edges, BoundSel, EstimateBound and FastNI cost $O(n+4\delta m)$ space to save the graph G'. Therefore, the total space complexity of Algorithm 4 for solving the C-WRWD is $O(n + 4\delta m)$.

Table 2: Summary of the datasets

Dataset	n	m	maxD	avgD	weighted
WSN	54	2458	54	46	Yes
CELE	297	2345	134	14.5	Yes
Adolescent	2.5k	13k	27	5.2	Yes
CaGrQc	5.2K	28.9K	81	5.5	No
Advogato	6.5K	51k	804	7.8	Yes
CaHepPh	12k	237k	491	19.7	No
CondMat	16k	48k	107	5.9	Yes
Slashdot	77k	937k	2537	12.1	No
YouTube	1135k	2988k	28754	5.3	No

Table 3: Parameter setting

Parameters	Value
k	20, 40, **60**, 80, 100
B	2, 4, **6**, 8, 10

7 EXPERIMENTS

In this section, we present experimental results on effectiveness, efficiency, memory consumption and scalability of our proposed methods.

7.1 Experimental Settings

Datasets. We use nine real-world datasets: WSN [1], CELE [2], Adolescent [3], CaGrQc [4], Advogato [5][1], CaHepPh [6], CondMat [2], Slashdot [7] and YouTube [8]. Their statistics are shown in Table 2. WSN is a wireless sensor network. For $\mathcal{PR}$, we use the probability of a message from a sender successfully reaching a receiver as the weight; for $\mathcal{CR}$, we use the Euclidean distance between two sensors as the transfer cost. CELE is a directed, weighted network representing the neural network of C. Elegans. Data. Adolescent is a friendship network. Advogato is the trust network of a social network site named advogato. CaGrQc and CaHepPh are paper collaboration networks. CondMat is a weighted network of coauthorships between scientists posting preprints on the Condensed Matter E-Print Archive. Slashdot is a technology-related news website known for its specific user community. YouTube is a social network of the video-sharing web site YouTube. In our datasets, WSN, CELE, Adolescent, Advogato and CondMat are edge-weighted graph. It is worth noting that the edge weight of Advogato is not an integer. We map it to an integer by enlarging the weight by 5 times. For the remaining non-integer weights, we round it up to an integer. The remaining published graph data sets are unweighted, but since our method is not dependent on the semantics of the weights or their magnitude, we assign randomly generated weights (real numbers in the range 1 to 10) to the edges of the remaining datasets.

Parameters. Table 3 shows the settings of all parameters, and the default one is highlighted in bold. Here we follow the parameter settings in [23]. In all experiments, we vary one parameter while the rest are kept default, unless specified otherwise.

Performance Measurement. For each method we evaluate the runtime and the average gain of nodes (Gain = $\mathcal{G}(S)/n$). Each experiment is repeated ten times, and the average result is reported.

[1]The original URLs of Adolescent and Advogato do not work. Then, we host the datasets on GitHub.

Methods for Comparison. We compare five methods as below. TopK: It selects the top-k nodes in term of degree as the seeds. DpSel: It is a dynamic programming based greedy method which is shown in Algorithm 1. MatrixSel: It is a matrix-based greedy method shown in Algorithm 2. BoundSel: It is a bound-based greedy method shown in Algorithm 4. SamSel: It is a sampling-based method proposed in [23]. We extend this method to answer P-WRWD and C-WRWD. Here, node u chooses its neighbor v with probability p_{uv} when we generate the $\mathcal{PR}$ sampling set. For $\mathcal{CR}$, we generate sampling set in G' to compute $\mathcal{G}(S)$. To generate the sampling set for SamSel, we adopt the exact same setting from [23], i.e., 500 Monte Carlo simulations (random walks) are repeatedly run for each node in G. Note that this value is not sufficient to guarantee the approximation ratio according to the theoretical analysis in [23]. PageRankSel: We use the weighted pagerank [37] to rank the node in G and choose the top-k nodes in terms of the pagerank value. Since the weighted pagerank considers the weight as probability, we only compare this algorithm with the algorithm used for the case of $\mathcal{PR}$ (Probability-aware Random Walk Model).

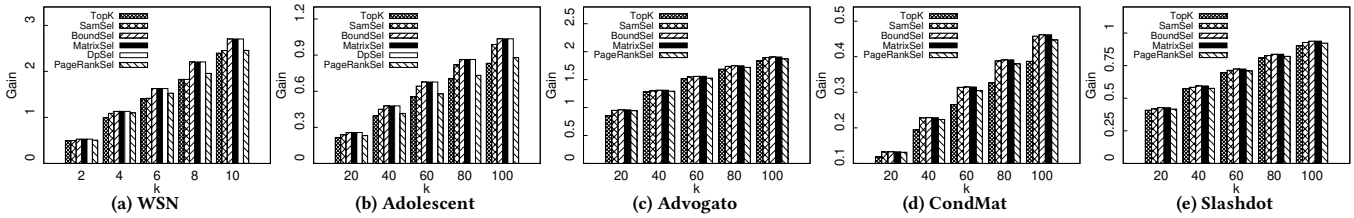
It is worth noting that DpSel is too slow to converge in 20 hours even for a very small dataset (Adolescent) when $k = 100$ and $B = 6$, we thereby have to omit it in most of the efficiency and effectiveness tests. Meanwhile, if the running time of one method is more than 10^5 seconds (i.e. more than one day), we also have to omit it.

Setup. All codes are implemented in Java, and experiments are conducted on a server with 2.3 GHz Intel Xeon 24 Core CPU and 256GB memory running Debian/4.0 OS.

7.2 Effectiveness Test

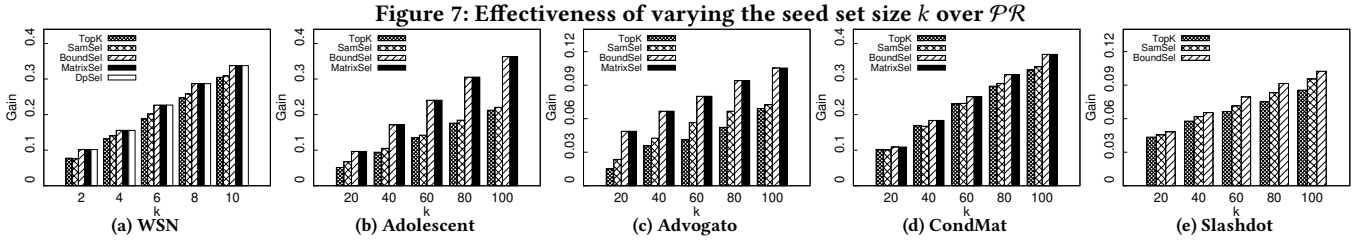
Varying the Seed Set Size k. The effectiveness of all algorithms over $\mathcal{PR}$ and $\mathcal{CR}$ on four datasets by varying k is shown in Figure 7 and Figure 8, respectively. Note that for WSN, we vary k from 2 to 10 because it has only 54 nodes (see Table 2). We have the following observations: (1) TopK has the worst performance. DpSel, BoundSel and MatrixSel achieve the same gain. When $k = 20$, the improvement of BoundSel and MatrixSel over TopK exceeds 220% on the Advogato dataset over $\mathcal{CR}$. (2) With the growth of k, the advantage of BoundSel and MatrixSel over TopK increases from 19% to 24% when k varies from 20 to 100 on the Adolescent dataset over $\mathcal{PR}$. This is because TopK ignores the overlaps of gain between the selected seeds, which degrades its effectiveness. However, BoundSel and MatrixSel select nodes based on marginal gain, which largely avoids the overlap of gain. (3) The performance of TopK is very different on different datasets. It implies that the network structure is an important variable for TopK. On the contrary, BoundSel and MatrixSel consistently perform better on all datasets. (4) As shown in Figure 7a and Figure 8a, we find DpSel, BoundSel and MatrixSel consistently beat all baselines in the real wireless dataset WSN.

Varying the Budget B. Figure 9 and Figure 10 show the result of varying B from 2 to 10 over $\mathcal{PR}$ and $\mathcal{CR}$, respectively. We find that: (1) Similar to Figure 7 and Figure 8, BoundSel and MatrixSel are consistently better than all baselines. (2) By carefully looking at the figures, it is surprising to see that the performance of TopK grows faster than that of the other methods. It is because when B is small, TopK ignores the overlaps of gain. With the growth of B, the overlaps of gain become unavoidable.



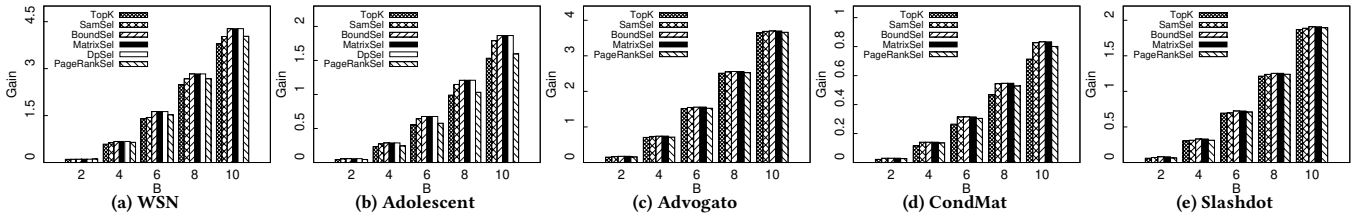
(a) WSN (b) Adolescent (c) Advogato (d) CondMat (e) Slashdot

Figure 7: Effectiveness of varying the seed set size k over $\mathcal{PR}$



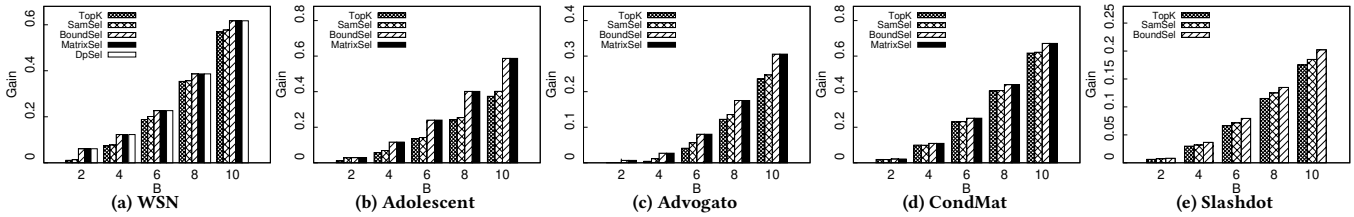
(a) WSN (b) Adolescent (c) Advogato (d) CondMat (e) Slashdot

Figure 8: Effectiveness of varying the seed set size k over $\mathcal{CR}$



(a) WSN (b) Adolescent (c) Advogato (d) CondMat (e) Slashdot

Figure 9: Effectiveness of varying the budget B over $\mathcal{PR}$



(a) WSN (b) Adolescent (c) Advogato (d) CondMat (e) Slashdot

Figure 10: Effectiveness of varying the budget B over $\mathcal{CR}$

7.3 Efficiency Test

Varying the Seed Set Size k. Figure 11 and Figure 12 present the efficiency results over $\mathcal{PR}$ and $\mathcal{CR}$, respectively, when k varies from 2 to 10 for WSN and k varies from 20 to 100 for the remaining datasets. We have the following observations: (1) The runtime of BoundSel and MatrixSel increase linearly with respect to k, which is consistent with the complexity analysis in Section 4.2.2 and 6.2. (2) BoundSel consistently beats MatrixSel and SamSel on all datasets. (3) The gap between BoundSel and MatrixSel decreases as k grows, because with the growth of k, the gap between the marginal gain of the remaining nodes becomes smaller and the pruning strategy actually performs worse. (4) From Figure 11b, we can see that the runtime of BoundSel and MatrixSel is almost identical. Meanwhile, they are about three orders of magnitude faster than DpSel, which is consistent with our time complexity analysis. (5) TopK and PagerankSel have high efficiency simply because they adopt a very simple strategy in finding a seed set, as described in our experiment setup.

Varying the Budget B. Figure 13 and Figure 14 show the runtime of all algorithms on four datasets by varying B over $\mathcal{PR}$ and $\mathcal{CR}$, respectively. We can see that the advantage of BoundSel over MatrixSel is decreases as B increases from 2 to 10 on all datasets. This is because the upper bound gets looser as B increases.

Varying the Average Edge Weight δ over $\mathcal{CR}$. In this experiment, we evaluate the efficiency of our methods over $\mathcal{CR}$ by varying the average edge weight δ on the CaGrQc and CaHepPh dataset. Here we assign randomly generated edge weights of the CaGrQc and CaHepPh dataset in different ranges to vary δ from 5 to 25. From Figure 15, we find that the runtime of BoundSel and MatrixSel increase linearly with respect to δ, which is consistent with the complexity analysis in Section 6.

Varying the Number of Edges m. In this experiment, to evaluate the running time of the proposed methods when the number of edges m is varying, we start from a small weighted graph, CELE, and randomly connect two nodes if there is no edge between them, until the number of edges reaches a given total number of edges m. Note that we assign randomly generated weights (real numbers within the range of [1,10]) to the new edges. Figure 16a and Figure 16b present the efficiency results over the Probability-aware Random Walk Model ($\mathcal{PR}$) and the Cost-aware Random Walk Model ($\mathcal{CR}$), respectively, when m varies from 5,000 to 25,000. We have the following observations: (1) The running time of BoundSel, MatrixSel and DpSel increase linearly with respect to m, which is consistent with the time complexity analysis. (2) The running time of SamSel remains almost constant when m grows, because the time complexity of SamSel is independent of m.

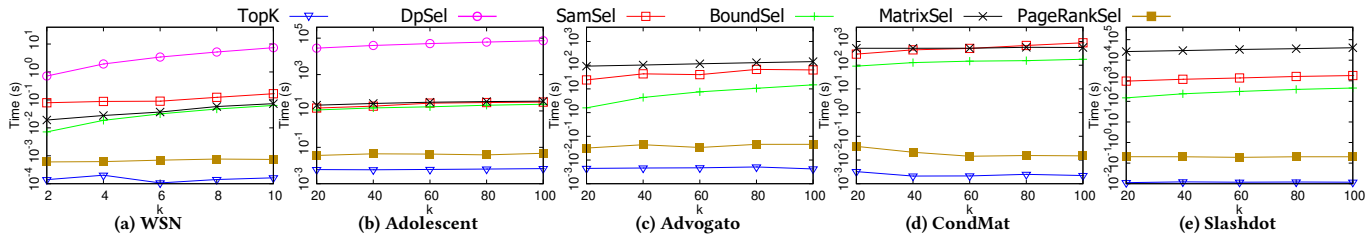


Figure 11: Efficiency of varying the seed set size k over $\mathcal{PR}$

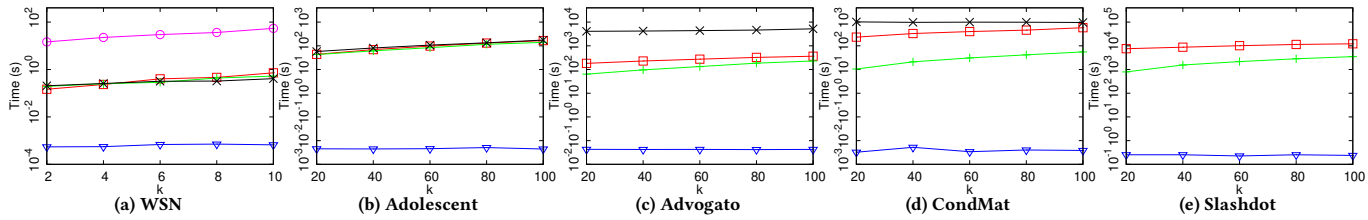


Figure 12: Efficiency of varying the seed set size k over $\mathcal{CR}$

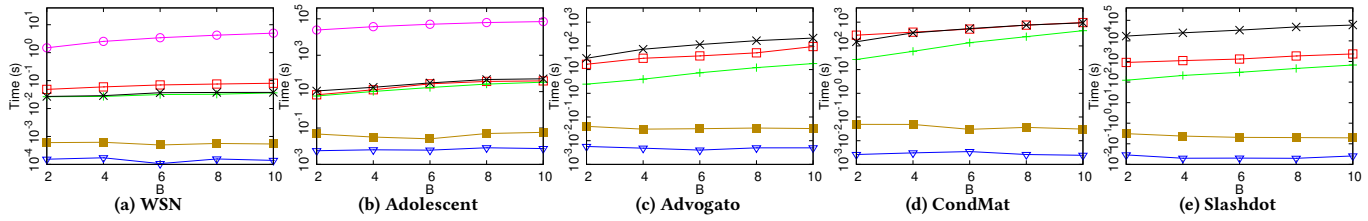


Figure 13: Efficiency of varying the budget B over $\mathcal{PR}$

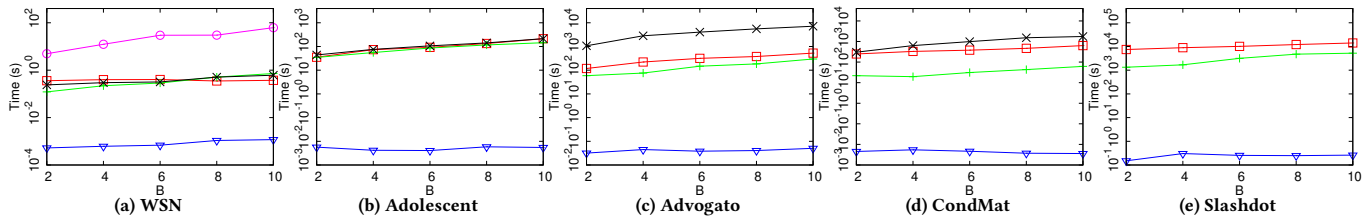


Figure 14: Efficiency of varying the budget B over $\mathcal{CR}$

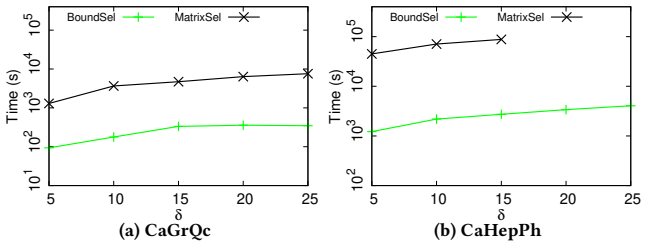


Figure 15: Efficiency of varying the average edge weight δ over $\mathcal{CR}$

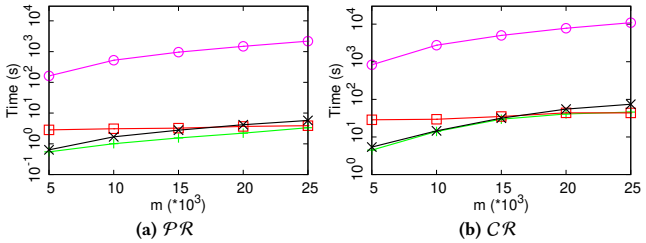


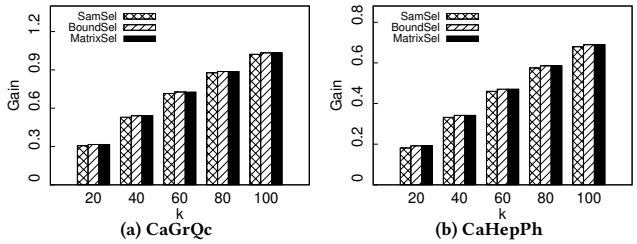
Figure 16: Efficiency of varying the number of edges m

7.4 Our Methods vs. SamSel on Unweighted Graph

In this section, we try to compare our work with the related work [23] on the CaGrQc and CaHepPh dataset without edge weight. Specifically, we test the effectiveness, runtime and memory consumption of each algorithm when B is varying and evaluate the scalability of SamSel and BoundSel by varying the number of nodes from 50,000 to 250,000 on the Youtube dataset

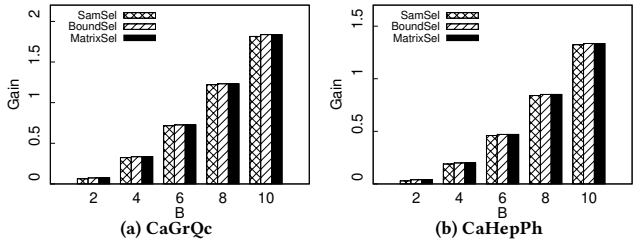
Effectiveness Test. The effectiveness of our methods and SamSel on unweighted graphs is shown in Figure 17 and Figure 18. From these two figures, we find that BoundSel and MatrixSel achieve the same gain, and they are both better than SamSel.

Efficiency Test. Figure 19 and Figure 20 show the runtime of three algorithms on the CaGrQc and CaHepPh datasets by varying k and B, respectively. We have the following observations: (1) BoundSel consistently beats SamSel on all datasets. (2) The gap between BoundSel and SamSel decreases as k increases. This is because the bottleneck of SamSel is the indexing of the Monte Carlo sampling results. It only needs to perform efficient heap tuning during the node selection phase. Therefore, the runtime of SamSel is independent of k. On the contrary, the runtime of BoundSel increases linearly with respect to k. (3) The advantage of BoundSel over SamSel decreases



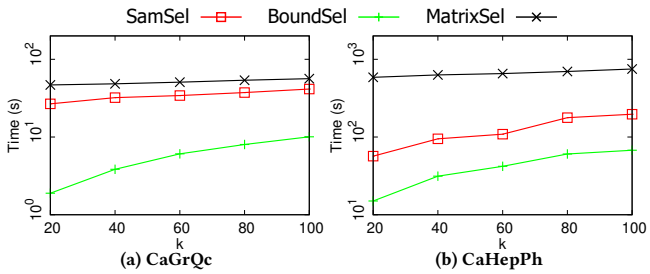
(a) CaGrQc (b) CaHepPh

Figure 17: Effectiveness of varying the seed set size k



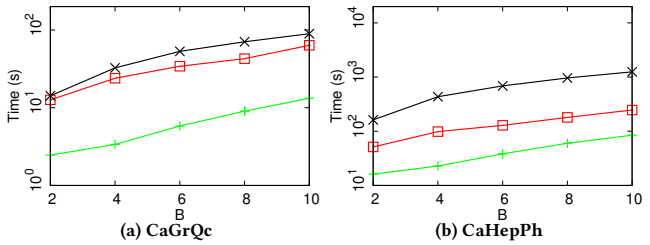
(a) CaGrQc (b) CaHepPh

Figure 18: Effectiveness of varying the Budget B



(a) CaGrQc (b) CaHepPh

Figure 19: Efficiency of varying the seed set size k



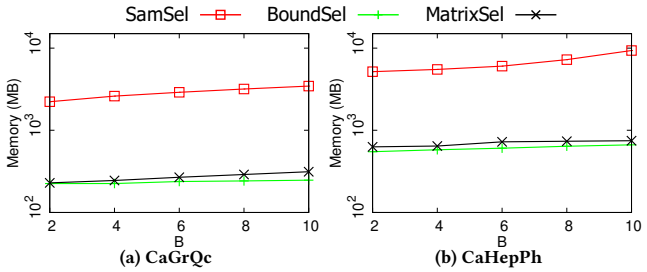
(a) CaGrQc (b) CaHepPh

Figure 20: Efficiency of varying the Budget B

as B increases from 2 to 10 on all datasets. This is because the upper bound gets looser as B increases.

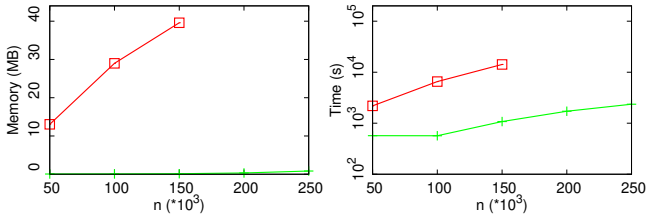
Memory Consumption. Figure 21 reports the average memory consumption of each algorithm when varying B. As shown in Figure 21, we find: (1) BoundSel and MatrixSel have similar memory consumption. (2) The memory consumption of SamSel is an order of magnitude larger than that of BoundSel and MatrixSel.

Scalability Test. In this experiment we evaluate the scalability of SamSel and BoundSel by varying the number of nodes from 50,000 to 250,000 on the Youtube dataset. When we load the graph, we first randomly choose one node and then start a breadth-first search from it to select other nodes until we reach the number of nodes we specified. Since DpSel and MatrixSel are extremely slow, we omit them. The memory consumptions and runtime are reported in Figure 22. In Figure 22a, we can see that BoundSel scales very well and only consumes less than 1GB when $|V|$=250,000; while the space of SamSel grows fast with the increase of $|V|$, and it



(a) CaGrQc (b) CaHepPh

Figure 21: Memory consumption



(a) Memory consumption (b) Runtime

Figure 22: Scalability test (on YouTube dataset)

exhausts the machine memory (30GB) when $|V|$=200,000. From Figure 22b, we find that BoundSel is more scalable than SamSel in execution time. When $|V|$=250,000, BoundSel can return the results in a reasonable time (40 minutes), but SamSel takes almost four hours to return the results. Note that as SamSel is out of memory when the size exceeds 150,000, we thereby have to omit it. Due to limited memory size in reality, it is difficulty for SamSel to be applied to very large networks, i.e., $|V| \geq 1,000,000$.

8 CONCLUSION

In this paper, we proposed and studied the problem of weighted random walk domination in a weighted graph. Given a weighted graph $G(V, E)$ and the maximal budget B of random walk, it aims to find a k-size set S, which can minimize the total costs of the other nodes to access S through a weighted random walk. To solve the WRWD problem, we proposed DpSel and MatrixSel approach based on DP and matrix respectively. They both achieve an approximation ratio of $(1 - 1/e)$. DpSel incurs a time complexity of $O(kBn^2m)$ and $O(\delta kBn^2m)$ for P-WRWD and C-WRWD, respectively, while MatrixSel incurs a time complexity of $O(kBnm)$ and $O(\delta kBnm)$ for P-WRWD and C-WRWD, respectively. In order to further accelerate MatrixSel, we proposed a BoundSel approach to further reduce the number of the gain computations in each candidate selection with an approach of proactively estimating the upper bound of the marginal gain of the candidate node. Meanwhile, BoundSel achieves the same approximation ratio guarantee as that of MatrixSel. Lastly, we conducted extensive experiments on real datasets to verify the efficiency, effectiveness, memory consumption and scalability of our methods.

Acknowledgments. Zhiyong Peng is supported in part by the Key Project of the National Natural Science Foundation of China (Project Number: U1811263), the National Key Research and Development Program of China (Project Number: 2018YFB1003400) and the Research Fund from Alibaba Group. Zhifeng Bao is supported in part by ARC DP200102611, DP180102050, and a Google Faculty Research Award.

REFERENCES

[1] [n.d.]. http://db.csail.mit.edu/labdata/labdata.html.
[2] [n.d.]. http://www-personal.umich.edu/~mejn/netdata/.
[3] [n.d.]. https://github.com/songsong945/dataset/blob/main/moreno_health.zip.
[4] [n.d.]. http://snap.stanford.edu/data/ca-GrQc.html.
[5] [n.d.]. https://github.com/songsong945/dataset/blob/main/advogato.zip.
[6] [n.d.]. http://snap.stanford.edu/data/ca-HepPh.html.
[7] [n.d.]. http://snap.stanford.edu/data/soc-Slashdot0811.html.
[8] [n.d.]. http://snap.stanford.edu/data/com-Youtube.html.
[9] Çigdem Aslay, Laks V. S. Lakshmanan, Wei Lu, and Xiaokui Xiao. 2018. Influence Maximization in Online Social Networks. In *WSDM*. ACM, 775–776.
[10] Song Bian, Qintian Guo, Sibo Wang, and Jeffrey Xu Yu. 2020. Efficient Algorithms for Budgeted Influence Maximization on Massive Social Networks. *PVLDB* 13, 9 (2020), 1498–1510.
[11] Don Coppersmith and Shmuel Winograd. 1987. Matrix multiplication via arithmetic progressions. In *STOC*. ACM, 1–6.
[12] Sudipto Guha and Samir Khuller. 1998. Approximation algorithms for connected dominating sets. *Algorithmica* 20, 4 (1998), 374–387.
[13] Qintian Guo, Sibo Wang, Zhewei Wei, and Ming Chen. 2020. Influence Maximization Revisited: Efficient Reverse Reachable Set Generation with Bound Tightened. In *SIGMOD*. ACM, 2167–2181.
[14] Teresa W. Haynes, Stephen Hedetniemi, and Peter Slater. 2017. *Domination in Graphs: Volume 2: Advanced Topics*. Routledge.
[15] Koushik Kar and Suman Banerjee. 2003. Node placement for connected coverage in sensor networks. In *WiOpt*. 1–2.
[16] Vasileios Karyotis, Fabio Pittalà, Maria Fazio, Symeon Papavassiliou, and Antonio Puliafito. 2009. Topology-Aware Hybrid Random Walk Protocols for Wireless Multihop Networks. In *MONAMI*. Springer, 107–118.
[17] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *SIGKDD*. ACM, 137–146.
[18] Fabian Kuhn and Roger Wattenhofer. 2005. Constant-time distributed dominating set approximation. *Distributed Computing* 17, 4 (2005), 303–310.
[19] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. 2010. Structure and evolution of online social networks. In *Link mining: models, algorithms, and applications*. Springer, 337–357.
[20] Cliff Lampe, Nicole Ellison, and Charles Steinfield. 2006. A Face (book) in the crowd: Social searching vs. social browsing. In *CSCW*. ACM, 167–170.
[21] Kristina Lerman. 2007. Social browsing & information filtering in social media. *arXiv preprint arXiv:0710.5697* (2007).
[22] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 2.
[23] Rong-Hua Li, Jeffrey Xu Yu, Xin Huang, and Hong Cheng. 2014. Random-walk domination in large graphs. In *ICDE*. IEEE, 736–747.
[24] Luisa Lima and Joao Barros. 2007. Random walks on sensor networks. In *WiOpt*. IEEE, 1–5.
[25] Wenqing Lin. 2019. Distributed Algorithms for Fully Personalized PageRank on Large Graphs. In *WWW*. ACM, 1084–1094.
[26] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming* 14, 1 (1978), 265–294.
[27] Neha Sengupta, Amitabha Bagchi, Maya Ramanath, and Srikanta Bedathur. 2019. ARROW: Approximating Reachability Using Random Walks Over Web-Scale Graphs. In *ICDE*. IEEE, 470–481.
[28] Xiance Si, Edward Y. Chang, Zoltán Gyöngyi, and Maosong Sun. 2010. Confucius and Its Intelligent Disciples: Integrating Social with Search. *PVLDB* 3, 2 (2010), 1505–1516.
[29] Frank Spitzer. 2013. *Principles of random walk*. Vol. 34. Springer Science & Business Media.
[30] Lichao Sun, Albert Chen, Philip S. Yu, and Wei Chen. 2020. Influence Maximization with Spontaneous User Adoption. In *WSDM*. ACM, 573–581.
[31] Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. 2018. Online Processing Algorithms for Influence Maximization. In *SIGMOD*. ACM, 991–1005.
[32] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: Near-optimal time complexity meets practical efficiency. In *SIGMOD*. ACM, 75–86.
[33] Hui Tian, Hong Shen, and Teruo Matsuzawa. 2005. Random walk routing for wireless sensor networks. In *PDCAT*. IEEE, 196–200.
[34] Shan Tian, Songsong Mo, Liwei Wang, and Zhiyong Peng. 2020. Deep Reinforcement Learning-Based Approach to Tackle Topic-Aware Influence Maximization. *Data Sci. Eng.* 5, 1 (2020), 1–11.
[35] Peng-Jun Wan, Khaled M Alzoubi, and Ophir Frieder. 2002. Distributed construction of connected dominating set in wireless ad hoc networks. In *INFOCOM*, Vol. 3. IEEE, 1597–1604.
[36] Sibo Wang, Xiaokui Xiao, Yin Yang, and Wenqing Lin. 2016. Effective Indexing for Approximate Constrained Shortest Path Queries on Large Road Networks. *PVLDB* 10, 2 (2016), 61–72.
[37] Wenpu Xing and Ali A. Ghorbani. 2004. Weighted PageRank Algorithm. In *CNSR*. 305–314.
[38] Sukhyun Yun, Jaehun Lee, Wooyong Chung, Euntai Kim, and Soohan Kim. 2009. A soft computing approach to localization in wireless sensor networks. *Expert Systems with Applications* 36, 4 (2009), 7552–7561.
[39] Degan Zhang, Guang Li, Ke Zheng, Xuechao Ming, and Zhao-Hua Pan. 2014. An energy-balanced routing method based on forward-aware factor for wireless sensor networks. *IEEE transactions on industrial informatics* 10, 1 (2014), 766–773.
[40] Ping Zhang, Zhifeng Bao, Yudong Niu, Yipeng Zhang, Songsong Mo, Fei Geng, and Zhiyong Peng. 2019. Proactive rumor control in online networks. *World Wide Web* 22, 4 (2019), 1799–1818.
[41] Haifeng Zheng, Feng Yang, Xiaohua Tian, Xiaoying Gan, Xinbing Wang, and Shilin Xiao. 2015. Data gathering with compressive sensing in wireless sensor networks: A random walk based approach. *IEEE Transactions on Parallel and Distributed Systems* 26, 1 (2015), 35–44.