Markov Chain Monitoring

Harshal A. Chaudhari* Michael Mathioudakis† Evimaria Terzi*

Abstract

In networking applications, one often wishes to obtain estimates about the number of objects at different parts of the network (e.g., the number of cars at an intersection of a road network or the number of packets expected to reach a node in a computer network) by monitoring the traffic in a small number of network nodes or edges. We formalize this task by defining the Markov Chain Monitoring problem.

Given an initial distribution of items over the nodes of a *Markov chain*, we wish to estimate the distribution of items at subsequent times. We do this by asking a limited number of queries that retrieve, for example, how many items transitioned to a specific node or over a specific edge at a particular time. We consider different types of queries, each defining a different variant of the Markov Chain Monitoring. For each variant, we design efficient algorithms for choosing the queries that make our estimates as accurate as possible. In our experiments with synthetic and real datasets we demonstrate the efficiency and the efficacy of our algorithms in a variety of settings.

1 Introduction

In this paper, we introduce and solve the Markov Chain Monitoring problem. As multiple items move on a graph in *Markov chain*-fashion, our goal is to keep track of their distribution across the graph. Towards that end, the Markov Chain Monitoring problem seeks to identify a limited number of nodes (or edges) so that, once we know exactly how many items reside on (resp. traverse) them, our uncertainty of the distribution of items on the graph is minimized.

This problem finds applications in many network settings. In a typical such setting, one wishes to have accurate estimates of the number of items that reside at various nodes of a network (e.g., vehicles that are at the intersections of a road network) however faces constraints in how big a part of the network (e.g., intersections or road segments) they can monitor at any time. For instance, in urban traffic networks different parts of the network are active at different times of the day or week. Therefore, actively measuring traffic at all points of the network would waste resources. The task applies similarly on other types of networks including computer or peer-to-peer networks where the items that propagate through the network are packages or files.

To the best of our knowledge, we are the first to introduce and study the Markov Chain Monitoring problem. Nevertheless, this problem is similar to tasks such as outbreak detection, sparsification of influence network, and the wider area of node and edge *centrality* on graphs. A major difference with previous work is that in Markov Chain Monitoring the "centrality" of nodes or edges is defined not simply in terms of the underlying graph structure, but also in terms of the dynamic propagation of items through the network.

We make two assumptions in our problem definition. The first is that there is a point in time when we have an accurate estimate of the placement of all items on the graph. The second is that we can monitor the subsequent placement of items by issuing a predefined number of *monitoring operations* (i.e., real-time queries on the *Markov chain*). We consider different types of operations: ones that retrieve the number of items that reside on specific nodes; and ones that retrieve the number of items that traverse specific edges. In the applications we consider, i.e., urban and computer traffic networks, the queries correspond to placing measurement devices on particular nodes or edges of the network, and retrieving their measurements.

Technically, different monitoring operations result in different variants of Markov Chain Monitoring. For each variant, we design efficient polynomial-time algorithms. For some of these algorithms we demonstrate that they are indeed optimal, while for others we show that they perform extremely well in practice. Our experiments use a diverse set of datasets from urban networks and computer networks to demonstrate the practical utility of our setting. For example, our experiments with data from the Hubway bike-sharing network of Boston, where the nodes are bike docks and the items that propagate through the network are the bikes themselves, identify as candidates for monitoring the stations that are close to the most busy Boston spots.

*Computer Science Department, Boston University, USA.

†Universite de Lyon, CNRS. INSA-Lyon, LIRIS, UMR5205, F-69621, France.

2 Related work

To the best of our knowledge, we are the first to introduce and study the MARKOV CHAIN MONITORING problem. However, this problem can also be seen as a graph centrality problem, where one seeks to identify k "central" nodes or edges to intercept the movement of items on a graph. Therefore, our work is related to existing work on graph centrality measures.

Graph centrality measures can be broadly cast into two categories: *individual* and *group centrality* measures. Individual measures assign a score to the each node/edge. Group centrality measures assign scores to sets of nodes or edges. Usually, computing group-centrality measures requires solving a complex combinatorial problems.

Examples of individual centrality scores for nodes and edges are the *Pagerank* [17], betweenness [2, 4, 18] and current flow centralities [3]. Pagerank is one classical example of a centrality measure based on a *Markov chain* – where the centrality of nodes is quantified as the stationary distribution of a *Markov chain* on the graph. Betweenness centrality and current flow centrality assign high centrality score to nodes/edges that participate in one or many short paths between all pairs of nodes in the input network. Although a *Markov chain* is used for Pagerank, its computation is very different than ours – after all, Pagerank is an individual centrality measure, while our measures are group centralities.

Group centrality measures that use a *Markov chain* model include the *Absorbing Random Walk Centrality* introduced by Mavroforakis *et al.* [15]. In that work, the centrality of a set of nodes is defined as the transient time of the *Markov chain* when the given nodes are "absorbing". An absorbing *Markov chain* is also assumed by the work of Gionis *et al.* [8]. In that setting, the authors aim to maximize the positive opinion of the network with respect to a specific topic by picking k nodes appropriately to endorse this positiveness via posts (e.g., in a social network). Both these problems are different from ours because we do not consider absorbing random walks, but rather a simple *Markov chain*. Moreover, our objective to minimize uncertainty is different from the objectives of the above papers.

While *Markov chain* is a simple model that allows us to quantify centrality, different models are more appropriate in other settings. For example, Ishakian *et al.* [10] proposed an extension of betweenness centrality to the group betweenness centrality. In that case, the goal was to find a group of nodes such that many shortest paths pass through those nodes. As another example, in epidimiology and information propagation settings, the underlying process of interest (e.g., the spreading of a virus or piece of information) is better modeled as a random cascade. In such a setting, the measure of *influence* [1, 5, 9, 11, 16, 20] captures another notion of centrality, where the centrality of a node is defined in terms of the expected spread of a random cascade that starts at the given node. In the same setting, two related tasks are those of outbreak detection [12] and graph sparsification [14]. In the former, the task is to identify as central those nodes that would intercept early a set of observed cascades. In the latter, one seeks edges that are central in explaining (from a model-sparsity point of view) the observed cascades as well as possible.

In terms of categorization, the the MARKOV CHAIN MONITORING problem is a group-centrality measure. What distinguishes the setting of MARKOV CHAIN MONITORING from all the above work is that centrality is defined in terms of a combination of static structure (graph), dynamic structure (*Markov chain*), and real-time activity (current placement of items on the graph), which is not the case in the aforementioned works.

3 Setting

In this section, we introduce our setting and notation.

Markov Chain: Consider a weighted directed graph $G = (V, E, p)$ with $|V| = n$ and $|E| = m$. We use $\pi(v) \subseteq V$ and $\kappa(v) \subseteq V$ to denote the set of parent and child nodes of node v, respectively.

$$\pi(v) = \{u \mid u \in V, e(u \to v) \in E\}$$
$$\kappa(v) = \{u \mid u \in V, e(v \to u) \in E\}$$

Edges $e(u \to v) \in E$ are associated with real-valued weights $p(u \to v) \in [0, 1]$ such that $\forall u \in V$: $\sum_{v \in V} p(u \to v) = 1$.

These weights give rise to a *Markov chain* with transition matrix $\mathbf{P}$ – where $\mathbf{P}(u, v) = p(u, v)$ denotes the probability of a transition from node u to node v. Moreover, we assume that a set of items are distributed among the nodes of G. We use the row vector $\mathbf{x}$ to denote the the *initial* number of items per node; that is, $\mathbf{x}(u)$ is the number of items initially at node u. For the entirety of this paper, transition matrix $\mathbf{P}$ and distribution $\mathbf{x}$ are assumed known and part of the input.

Consider now a single step of the *Markov chain*. During this step, each and every item transitions from the node u where it resides originally to another node v, according to transition probabilities $\mathbf{P}(u, v)$. At the end of this step, items are redistributed among the nodes - and we are no longer certain about their position. We use $\mathbf{Z}$ to denote the random vector with the number of items at each node after one transition. The *expected number of items* at each node is given by: $\mathbf{z} = E[\mathbf{Z}] = \mathbf{xP}$.

Quantifying uncertainty: We wish to have good point estimates for values $\mathbf{z}(v)$. We quantify the quality of estimates in terms of the variance in the number of items on each node afer the transition step. Specifically, let us consider the number $\mathbf{Z}(v)$ of items on node u after the transition step: an item previously at node u will transition to node v according to a Bernoulli distribution with success probability $\mathbf{P}(u,v)$; and since items transition from node to node independently from each other, the variance in the number of items $\mathbf{Z}(v)$ is

$$\begin{aligned}\mathbf{var}(\mathbf{Z}(v)) &= \mathbf{var}(\mathbf{Z}(v)|\mathbf{P},\mathbf{x}) \\ &= \sum_{u\in V} \mathbf{x}(u)\mathbf{P}(u,v)\left(1-\mathbf{P}(u,v)\right). \end{aligned} \tag{3.1}$$

To obtain an aggregate measure of uncertainty F_0, we opt to sum the aforementioned quantity over all nodes.

$$\begin{aligned}F_0 &= \sum_{v\in V} \mathbf{var}(\mathbf{Z}(v)) \\ &= \sum_{u\in V} \mathbf{x}(u) \sum_{v\in V} \mathbf{P}(u,v)\left(1-\mathbf{P}(u,v)\right)\end{aligned} \tag{3.2}$$

Monitoring: Given the transition matrix $\mathbf{P}$ and the initial distribution of items $\mathbf{x}$, we estimate the distribution of items $\mathbf{Z}$ after one transition step, with the uncertainty given in Equation (3.2). After one transition step, we are allowed to retrieve information about the position of the items and thus reduce uncertainty. We do this by performing "monitoring operations", i.e. queries on the position of items on the *Markov chain*. These operations are of the following types:

- ParentTransitions Retrieve the number of items that transitioned to node v from each $u \in \pi(v)$;
- NodeItems Retrieve the number of items that reside on node v after the transition step;
- EdgeTransitions Retrieve the number of items that transitioned from node u to node v;
- ChildrenTransitions Retrieve the number of items that transitioned from node u to each child $v \in \kappa(u)$.

From the above four types of monitoring operations, the last one (i.e., ChildrenTransitions) are both a bit unintuitive and they lead to trivial combinatorial problems. Thus, we omit them from the rest of the discussion.

Expected uncertainty Once we retrieve the answer $\mathtt{A}$ to a set of monitoring operations, we have more information about the positioning of items over nodes V – and thus an updated (and non-increased) uncertainty $F(\mathtt{A}) = \sum_{\mathtt{v}\in\mathtt{V}} \mathbf{var}(\mathbf{Z}(\mathtt{v})|\mathtt{A})$. In the setting we consider, however, the challenge we face is *not* to compute the uncertainty *given* the information retrieved via a monitoring operation, but rather to *select the monitoring operations* so that the uncertainty we will face *after* retrieving $\mathtt{A}$ is minimized *in expectation*. Therefore, the quantity of interest is that of *expected uncertainty* for a set of operations that we choose to perform, expressed as $E[\sum_{v\in V} \mathbf{var}(\mathbf{Z}(v)|\mathtt{A})]$.

Let us now assume we have chosen to perform a set of operations of either one of the aforementioned types. In what follows, we provide formulas for the *expected uncertainty* in each case.

Expected uncertainty under ParentTransitions**:** We perform monitoring operations for a subset $S \subseteq V$ of nodes – and obtain an answer $A_{PT}(S) = \{n_{uv}; v \in S,\ e(u \to v) \in E\}$, where n_{uv} is the number of transitions to v from its parent node u. The expected value $\mathbf{F}_{PT}(S)$ of the uncertainty $F(A_{PT}(S))$ after these operations is given by

$$\begin{aligned}\mathbf{F}_{PT}(S) =& E[F(A_{PT}(S))] \\ =& \sum_{u\in V} \mathbf{x}'(u) \sum_{v\in V\setminus S} \mathbf{P}'(u,v)\left(1-\mathbf{P}'(u,v)\right)\end{aligned} \tag{3.3}$$

where

$$\rho(u,S) = \sum_{v\in S} \mathbf{P}(u,v)$$

$$\mathbf{x}'(u) = \mathbf{x}(u)\left(1-\rho(u,S)\right) \tag{3.4}$$

$$\mathbf{P}'(u,v) = \frac{\mathbf{P}(u,v)}{1-\rho(u,S)}. \tag{3.5}$$

Intuitively, $\mathbf{x}'(u)$ expresses the expected number of items that transition from u to nodes v *other than* those in S; and $\mathbf{P}'(u,v)$ expresses the probability an item transitions from u to v *given* that it does *not* transition to those in S. We see, then, that Equation (3.3) has the same form as Equation (3.2) but is evaluated on adjusted values of $\mathbf{x}$ and $\mathbf{P}$, to take into account the information we obtain via A_{PT}.

Note that the expected uncertainty after the monitoring operations is no larger than F_0[1].

Expected uncertainty under NodeItems**:** We perform monitoring operations for a subset $S \subseteq V$ of the nodes – and obtain an answer $A_{NI}(S) = \{n_v; v \in S\}$, where n_v is the number of items at node v after the transition. For an instance of answer $A_{NI}(S)$, let also $A_{PT}(S) = \{n_{uv}; v \in S, e(u \to v) \in E\}$ be an answer for ParentTransitions on the same set S of nodes – which by definition is *consistent with* $A_{NI}(S)$, in the

[1]For a proof, see Supplementary Material, Lemma A.1.

sense that

$$n_v = \sum_{u \in V} n_{uv}, \forall v \in S. \tag{3.6}$$

It can be shown that the *expected uncertainty* is equal for the two cases. That is: $\mathbf{F}_{PT}(S) = \mathbf{F}_{NI}(S)$. [2].

Expected uncertainty under EdgeTransitions**:** We perform monitoring operations for a subset $D \subseteq E$ of the edges – and obtain an answer $A_{ET} = A_{ET}(D) = \{n_e; e \in D\}$, where n_e is the number of transitions over edge e. The expected value $\mathbf{F}_{ET}(D)$ of the uncertainty $F(A_{ET}(D))$ after these operations is given by

$$\begin{aligned}\mathbf{F}_{ET}(D) =& E[F(A_{ET})] \\ =& \sum_{u \in V} \mathbf{x}''(u) \sum_{e(u \to v) \in E \setminus D} \mathbf{P}''(u,v)\left(1 - \mathbf{P}''(u,v)\right) \end{aligned} \tag{3.7}$$

where

$$\rho(u, D) = \sum_{e(u \to v) \in D} \mathbf{P}(u,v)$$

$$\mathbf{x}''(u) = \mathbf{x}(u)\left(1 - \rho(u, D)\right) \tag{3.8}$$

$$\mathbf{P}''(u,v) = \frac{\mathbf{P}(u,v)}{1 - \rho(u,D)} \tag{3.9}$$

Similar to ParentTransitions and NodeItems, expected uncertainty $\mathbf{F}_{ET}(S)$ is no greater than F_0.

4 Problem Definition

The general problem of Markov Chain Monitoring is to select the appropriate monitoring operations to reduce the expected uncertainty after they are performed. Stated formally:

Problem 1. (Markov Chain Monitoring) *Given a transition matrix P and an initial distribution of items $\mathbf{x}$, select a set of up to k monitoring operations to minimize the expected uncertainty $\mathbf{F}$.*

We study variants of the problem – each defined for a specific type of monitoring operation. For simplicity, we refer to these problems with the same name as that of the operation type: ParentTransitions, NodeItems, ChildrenTransitions, and EdgeTransitions.

Furthermore, as we saw in Section 3, variants ParentTransitions and NodeItems are equivalent: for the same set of nodes, operations of the first type reduce expected uncertainty as much as the second. Therefore, in what follows, we treat only the variant of NodeItems, as our claims apply directly to ParentTransitions as well.

[2] For a proof, see Supplementary Material, Theorem A.1.

5 The NodeItems problem

In this section, we provide the formal problem definition of the NodeItems problem variant and describe a greedy polynomial-time algorithm for solving it.

Problem 2. (NodeItems) *Given $G = (V, E)$, transition matrix $\mathbf{P}$, initial distribution of items to nodes $\mathbf{x}$ and integer k, find $S \subseteq V$ such that $|S| = k$ such that $\mathbf{F}_{NI}(S)$ is minimized.*

A brute-force way to solve Problem 2 would be to evaluate the objective function over all node-sets of size k. Obviously such an algorithm is infeasible – and we thus study a natural greedy algorithm for the problem, namely `NodeGreedy`.

The `NodeGreedy` algorithm: This is a greedy algorithm that performs k iterations; at each iteration, it adds one more node in the solution. If S^t is the solution at iteration t, then solution S^{t+1} is constructed by finding the node $u \in V \setminus S^t$ such that:

$$v^* = \arg\min_{v \in V \setminus S^t} \mathbf{F}_{NI}\left(S^t \cup \{v\}\right). \tag{5.10}$$

Although in the majority of our experiments that compare the brute-force solutions with those of `NodeGreedy` the two solutions were identical, we identified some contrived instances for which this was not the case. Thus, `NodeGreedy` is not an optimal algorithm for Problem 2.

Running time `NodeGreedy` evaluates Equation (5.10) at each iteration. A naive implementation of this would require computing Equation (3.3) $O(|V|)$ times per iteration, each time using $O(|V|^2)$ numerical operations. As a first improvement, we avoid the full double summation over V via a summation over edges E,

$$\begin{aligned}\mathbf{F}_{PT}(S) =& \sum_{u \in V} \mathbf{x}'(u) \sum_{v \in V \setminus S} \mathbf{P}'(u,v)\left(1 - \mathbf{P}'(u,v)\right) \\ =& \sum_{(u,v) \in E, v \in V \setminus S} \mathbf{x}'(u)\mathbf{P}'(u,v)\left(1 - \mathbf{P}'(u,v)\right), \end{aligned} \tag{5.11}$$

that involves $O(k|V||E|)$ numerical operations.

We can further speed-up the algorithm if we reuse at each step the computations done in the previous one. To see how, let S_t (resp. S_{t+1}) be the solution we construct after t (resp. $(t+1)$) iterations and let v^* be the node such that $S_{t+1} = S_t \cup v^*$. Then, for any $u \in V$ we have $\rho(u, S_t) = \sum_{v \in S_t} \mathbf{P}(u,v)$, and therefore

$$\rho(u, S_{t+1}) = \rho(u, S_t) + \mathbf{P}(u, v^*). \tag{5.12}$$

Moreover, for any $S \subseteq V$ let

(5.13)
$$B(u,S) = \sum_{(u,v)\in E \ s.t. \ v\in V\setminus S} \mathbf{P}'(u,v)\,(1-\mathbf{P}'(u,v)) = \sum_{v\in V\setminus S} \frac{\mathbf{P}(u,v)}{1-\rho(u,S)}\left(1-\frac{\mathbf{P}(u,v)}{1-\rho(u,S)}\right).$$

We can then express $B(u,S_{t+1})$ in terms of $B(u,S_t)$:

(5.14)
$$B(u,S_{t+1}) = B(u,S_t) - 2\mathbf{P}(u,v^*)\,(1-\rho(u,S_t)-\mathbf{P}(u,v^*)).$$

Finally, using Equations (5.12) and (5.14) and algebraic manipulations, we can express $\mathbf{F}_{NI}(S_{t+1})$ as follows:

(5.15)
$$\mathbf{F}_{NI}(S_{t+1}) = \sum_{u\in V}\mathbf{x}(u)\left(\frac{B(u,S_t)}{1-\rho(u,S_{t+1})} - 2\mathbf{P}(u,v^*)\right)$$

Thus, if we store $B(u,S_t)$ and $\rho(u,S_t)$ at iteration t, then evaluating Equation (5.15) at iteration $t+1$ takes only $O(|V|)$ numerical operations.

For all iterations but the first one, the above sequence of rewrites enables us to achieve a speedup from $O(|V||E|)$ to $O(|V|^2)$ numerical operations per iteration. For the first iteration, initializing the auxiliary quantities $B(u,\emptyset)$, $u \in V$, still takes $O(|E|)$. With this book-keeping, the running time of `NodeGreedy` is reduced from $O(k|V||E|)$ to $O(|E|+k|V|^2) = O(k|V|^2|)$. Note also that `NodeGreedy` is amenable to parallelization, as, given the auxiliary quantities from the previous step, we can compute the objective function independently for each candidate node.

6 The EdgeTransitions problem

Whereas NodeItems (Problem 2) seeks k nodes to optimize expected uncertainty, EdgeTransitions seeks k edges.

Problem 3. (Edge-Monitoring) *Given $G = (V,E)$, transition matrix $\mathbf{P}$, initial distribution of items to nodes $\mathbf{x}$ and integer k, find $S \subseteq V$ such that $|S| = k$ such that $\mathbf{F}_{ET}(S)$ (Equation (3.7)) is minimized.*

We provide two polynomial-time algorithms to solve the problem, namely `EdgeDP` and `EdgeGreedy`. For the former, we can also prove that it is optimal, and thus Problem 3 is solvable in polynomial time.

The `EdgeDP` algorithm: `EdgeDP` is a dynamic-programming algorithm that selects edges in two steps: first, it sorts the *outgoing* edges of each node in decreasing order of transition probability, thus creating $|V| = n$ corresponding lists; secondly, it combines top edges from each list to select a total of k edges.

In more detail, let $SOL_i(k)$ be the cost of an optimal solution for the special case of a budget of k edges allocated among outgoing edges of nodes $V_{i:} = \{i, i+1, \ldots, n\}$. According to this notational convention, the cost of an optimal solution D_{opt} for the problem is given by $SOL_1(k)$. Moreover, considering Equation (3.7), let $\mathbf{F}_i$ be the function that corresponds to the (outer) summation term for node i

(6.16)
$$\mathbf{F}_i(D) = \mathbf{x}''(i)\sum_{v\in V\setminus S}\mathbf{P}''(i,v)\,(1-\mathbf{P}''(i,v))$$

(under the auxilliary definitions of Equations (3.8) and (3.9)) and $ISOL_i(m)$ its optimal value when D contains no more than $m \le k$ outgoing edges from node i. Let also D_i^m be a subset of k outgoing edges of i with the highest transition probabilities. It can be shown that the optimal value for $\mathbf{F}_i(D)$ is achieved for the edges D_i^k with *highest* transition probability[3]. Having the outgoing edges of i sorted by transition probability, we can compute $ISOL_i(m)$ for all $m = 0 \ldots k$.

The dynamic programming equation is:

(6.17)
$$SOL_i(k) = \arg\min_{0\le m\le k}\{ISOL_i(m) + SOL_{i+1}(k-m)\}$$

`EdgeDP` essentially computes and keeps in memory $\|V\| \times (k+1)$ values according to Eq.(6.17).

Given the above, we have the following result:

Lemma 6.1. *The `EdgeDP` algorithm is optimal for the* Edge-Monitoring *problem.*

For a proof sketch of this lemma see Supplementary Material, Theorem A.2.

Running time: `EdgeDP` computes $k \times |V|$ values. For each value to be computed, up to $O(k)$ numerical operations are performed. Therefore, `EdgeDP` runs in $O(k^2|V|)$ operations. Backtracking to retrieve the optimal solution requires at most equal number of steps, so it does not increase the asymptotic running time.

The `EdgeGreedy` algorithm: `EdgeGreedy` is a natural greedy algorithm that selects k edges in an equal number of steps, in each step selecting one more edge to minimize $\mathbf{F}_{ET}$.

In all our experiments the performance of `EdgeGreedy` is the same as the performance of the optimal `EdgeDP` algorithm. However, we do not have a proof that the former is also optimal. We leave this as a problem for future work.

[3] For a proof, see Supplementary Material, Lemma A.2.

Running time: Following Equation (3.7), to select k edges, `EdgeGreedy` invokes up to $k \times O(|E|)$ evaluations of $\mathbf{F}_{ET}$. As we discussed for `NodeGreedy`, if the evaluation of the objective function is naively implemented with a double summation, the running time of `EdgeGreedy` is $O(k|E||V|^2)$ numerical operations. If the objective function is implemented as a summation over edges, the running time improves to $O(k|E|^2)$. Furthermore, following the observations similar to those we saw for `NodeGreedy`, the running time of `EdgeGreedy` becomes $O(|E| + k|E|) = O(k|E|)$.

We notice that `EdgeDP` has better performance than `EdgeGreedy` for dense graphs ($|E| \approxeq |V|^2$) and small k. Moreover, as with `NodeGreedy`, `EdgeGreedy` is amenable to parallelization - the new value of the objective function can be computed in independently for each edge that's considered for selection.

7 Experiments

In this section, we describe the results of our experimental evaluation using real and synthetic data. The results demonstrate that our methods perform better than other baseline methods with respect to our objective function. Moreover, using the bike-sharing network of Boston, we provide anecdotal evidence that our methods pick meaningful nodes to monitor.

7.1 Experimental setup Let us first describe the experimental setup, i.e., the datasets and baseline algorithms used for evaluation.

Graph datasets: We use the following graphs to define *Markov chain*s for our experiments.

`AS` is a graph that contains information about traffic between *Autonomous Systems* of the Internet. The dataset was retrieved through the Stanford Large Network Dataset Collection (SNAP) [13]. We experimented with three snapshots of the communication graphs between years 1997 and 2000. Here we demonstrate results for one of the snapshots (1999-2000), as we did not find significant difference among them.

The `AS` graph contains one node for each AS. Moreover, for every pair of nodes between which there is traffic according to the dataset, we place two directed edges between the nodes, one in each direction. To create an instance of the transition matrix, we assign equal probabilities to the outgoing edges of each node.

`Grid` graphs: The `Grid` graphs are planar, bi-directed grid graphs, where each node has in- and out-degree 4 (with the exception of border nodes).

`Geo` graphs: The `Geo` graphs are bi-directed geographic graphs. They are generated as follows: each node is placed randomly within a unit square on the two-dimensional euclidean plane. Subsequently, pairs of nodes are connected with directed edges in both directions if their euclidean distance is below a pre-defined threshold $ds = 0.01$.

`BA` graphs: The `BA` graphs are generated according to the Barabasi-Albert model. According to the model, nodes are added to the graph incrementally one after the other, each of them with outgoing edges to m existing nodes selected via preferential attachment. Here we show results for $m = 3$, but the results were similar for values $m = 5, 10$.

Similar to the methodology of Gionis *et al.* [7], the `Grid`, `Geo` and `BA` graphs provide us with different varieties of synthetic graphs to explore the performance our methods.

Item distributions: For each aforementioned graph, we generate an initial distribution of items $\mathbf{x}$ according to one of the following four schemes.

- `Ego`. Items are assigned in two steps. Firstly, one node is selected uniformly at random among all nodes. Secondly, 70% of items are assigned randomly on the neighbors of the selected node (including the selected node itself). Finally, the remaining items are distributed randomly to the nodes outside the neighborhood of the selected node.
- `Uniform`. Each node is assigned the same number of items.
- `Direct`. The number of items on each node is directly proportional to its out-degree. Note that items are distributed in a deterministic manner.
- `Inverse`. The number of items on each node is assigned deterministically to be inversely proportional to its out-degree.

Now each graph described above is combined with each item-distribution scheme. As a result, we obtain datasets of the form `G-X`, where `G` is any of `AS`, `Grid`, `Geo` and `BA` and `X` is any of the `Ego`, `Uniform`, `Direct` and `Inverse`. For simplicity, for the datasets that are generated randomly, we perform experiments over a single fixed instantiation.

The `Hubway` dataset: Hubway is a bike-sharing system in the Boston metro area, with a fleet of over 1000 bikes and over 100 hubway stations where users can pick up or drop off bikes at. Every time a user picks up a bike from a Hubway station, the system records basic information about the trip, such as the pick-up and drop-off station, and the corresponding pick-up and drop-off times. Moreover, the data contain the number of available bikes at each Hubway station every minute. The dataset was made publicly available by Hubway for the purposes of its Data Visualization Challenge[4].

[4] http://hubwaydatachallenge.org/

Using the dataset, we create instances of the problems we consider as follows. Firstly, we create a complete graph by representing each station with one node in the graph, and considering all possible edges between them. Subsequently, we consider a time interval (t_s, t_e) and the bikes that are located at each station (node). Representing bikes as items in our setting, we assign a transition probability $\mathbf{P}(u, v)$ between nodes u and v by considering the total number n_u of bikes at station u at start time t_s and, among these bikes, the number n_{uv} of them that were located at station v at end time t_e. We then set $\mathbf{P}(u, v) = n_{uv}/n_u$ and ignore edges with zero transition probability.

We experimented with a large number of such instances for different intervals (t_s, t_e), with a moderate length of 2 hours, to capture real-life transitions from one node to another. For the experiments presented in the paper, we use a fixed instance for the interval between 10am and 12pm on April 1st, 2012. In this interval, we consider 61 stations with at least one trip starting or ending at each. We refer to the dataset so constructed as the `Hubway` dataset.

Baseline algorithms: In order to assess the performance of our proposed algorithms for the Node-Monitoring and Edge-Monitoring problem variants, we compare it to that of well-known baseline algorithms from the literature. Since we are the first to tackle the problem of Markov Chain Monitoring, the baseline algorithms we compare with do not target our objective function directly. Nevertheless, the comparison helps us highlight the settings in which our algorithms are essential to achieve good performance for Markov Chain Monitoring.

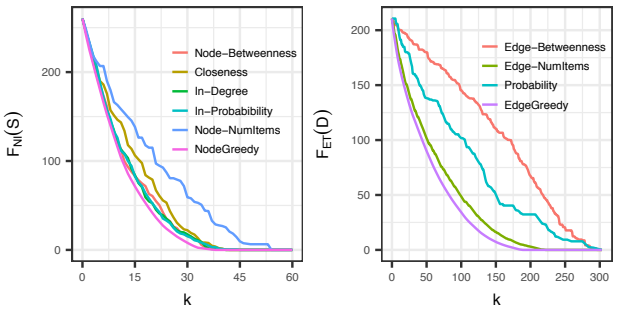
Below, we describe the respective baselines for the two variants of the problem.

Baselines for Node-Monitoring*:* For a budget k, the following baselines return a set of k nodes with highest value for the respective measure:

- `In-Degree`: number of incoming edges;
- `In-Probability`: total probability of incoming edges;
- `Node-Betweenness`: as defined in [2, 4, 18];
- `Closeness`: as defined in [19];
- `Node-NumItems`: number of items before transition;

Baselines for Edge-Monitoring*:*. For a budget k, the following baselines return a set of k edges with highest value for the respective measure:

- `Edge-Betweenness`: as defined in [2, 4, 18];
- `Edge-NumItems`: expected number of items to transition over the edge;
- `Probability`: transition probability of the edge.



(a) Node-Monitoring (b) Edge-Monitoring

Figure 1: `Hubway` data; y-axis: expected uncertainty, x-axis: number of monitored nodes or edges.

In what follows, context determines which baseline and variant we refer to.

7.2 Experimental results In this section, we report the performance of algorithms for the Markov Chain Monitoring problem - first on the graph datasets, combined with item distribution schemes; then on the `Hubway` dataset. As objective we always use the expected uncertainty achieved for a given budget k of nodes or edges – the smaller its value, the better the performance of the algorithm. Note that we do not report separately the performance of `EdgeDP`, as it achieves same performance as `EdgeGreedy`, but is not as efficient.

We provide the results for the graph datasets in Table 1. In all these experiments we use $k = 50$. Moreover, $r(A)$ is the ratio of the achieved objective value (for $k = 50$) over the initial value F_0 of the measure (for no monitoring operations, i.e., $k = 0$). The table shows four quantities for every graph-item distribution pair : $r(A)$ for $A = \{$`NodeGreedy`, `Node-Baseline`*, `EdgeGreedy`, `Edge-Baseline`* $\}$. Note that `Node-Baseline`* (resp. `Edge-Baseline`*) refers to the baseline algorithm with the best performance. For every algorithm A, $r(A) \in [0, 1]$ and the smaller the value of $r(A)$ the better the performance of the algorithm.

From the table, we observe that for the `AS` dataset, `NodeGreedy` significantly outperforms the best baseline for the `Ego` item distribution, while performing marginally better for other item distributions. The value of r(`EdgeGreedy`) is only slightly less than the best baselines across all the configurations. However, we observe that there is no baseline which performs uniformly the best across different item distributions. For example, `Edge-Betweenness` is the best baseline for `Direct` item distribution, the `Edge-NumItems` for `Ego`, while they both perform worse than even randomly chosen edges for `Uniform` and `Inverse` item distributions. Notably, for the `Geo` graphs, the greedy algorithms sig-

Graph	Item Distribution	$r(\texttt{NodeGreedy})$	$r(\texttt{Node}-\texttt{Baseline}^*)$	$r(\texttt{EdgeGreedy})$	$r(\texttt{Edge}-\texttt{Baseline}^*)$
AS	Ego	0.06	0.24	0.14	0.15
	Direct	0.66	0.67	0.99	0.99
	Uniform	0.38	0.40	0.97	0.99
	Inverse	0.38	0.40	0.97	0.99
Geo	Ego	0.00	0.06	0.01	0.02
	Direct	0.00	0.06	0.20	0.65
	Uniform	0.00	0.06	0.15	0.65
	Inverse	0.00	0.07	0.15	0.65
Grid	Ego	0.27	0.27	0.29	0.29
	Direct	0.92	0.92	0.98	0.98
	Uniform	0.92	0.92	0.98	0.98
	Inverse	0.92	0.92	0.98	0.98
BA	Ego	0.18	0.56	0.26	0.26
	Direct	0.71	0.71	0.99	0.99
	Uniform	0.63	0.63	0.98	0.98
	Inverse	0.63	0.63	0.98	0.98

Table 1: Comparison of greedy algorithms with the best-performing baseline ($\texttt{Node}-\texttt{Baseline}^*$ and $\texttt{Edge}-\texttt{Baseline}^*$) for $k = 50$. For a given pair of graph and item-distribution scheme, $r(A)$ expresses the ratio of the expected uncertainty that algorithm A achieves with $k = 50$ monitoring operations over the initial uncertainty F_0 (for $k = 0$). Note that the best-performing baseline is different for different rows of the table.

nificantly outperform the baselines

For the Grid graphs, the baselines perform exactly the same as our algorithms. This can be explained by the nature of the Grid graph, where all the nodes except the ones on the boundary are similar to each other, thereby rendering the Direct, Uniform and Inverse item distributions very similar to each other. For the Ego distribution, the greedy algorithms perform marginally better than the baselines. Again, there is no baseline which performs uniformly the best. Similar is our explanation for the results on BA graphs as in these graphs most of the nodes have almost the same (small) degrees too.

A more thorough investigation of the performance of the algorithms for different values of k and for all datasets we consider is shown in Supplementary Material, Section B.

Experiments with Hubway data: In our last experiment, we explore the performance of our algorithms on the Hubway dataset. From Figure 1a and Figure 1b we observe that the NodeGreedy and EdgeGreedy algorithms are consistently the best at reducing expected uncertainty, although the baselines are competitive on the relatively smaller graph. In Figure 2, we plot the Hubway stations across Boston chosen by the NodeGreedy algorithm with $k = 5$. The nodes chosen by the algorithm are supported by the anecdotal evidence of being exactly some of the of the most popular landmarks around the city. From a managerial perspective, tracking the number of trips starting or ending at these Hubway stations can help the operators better reduce the expected uncertainty around the expected number of bikes available at its different stations and anticipate future bike "re-balancing"[5] operations.

Running times: For all our experiments we use a single process implementation of our algorithms on a 24-core 2.9GHz Intel Xeon E5 processor with 512GB memory. For the largest graph in our experiments, the parallelized version of the NodeGreedy takes about 5−10 seconds per selected node, while the parallelized version of EdgeGreedy takes about 1 minute per selected edge.

Discussion: Our experiments show that NodeGreedy and EdgeGreedy consistently perform better than or on par with other popular baseline methods. Also, for graphs with relatively large number of nodes, the solutions to the Node-Monitoring problem are more effective at reducing the expected uncertainty than the solutions to the Edge-Monitoring problem for the same number of node (resp. edge) monitors. This is especially important considering our analysis from Section 5 and 6 which show that the NodeGreedy algorithm has a better time complexity compared to

[5] https://www.citylab.com/transportation/2014/08/balancing-bike-share-stations-has-become-a-serious-scientific-endeavor/379188/

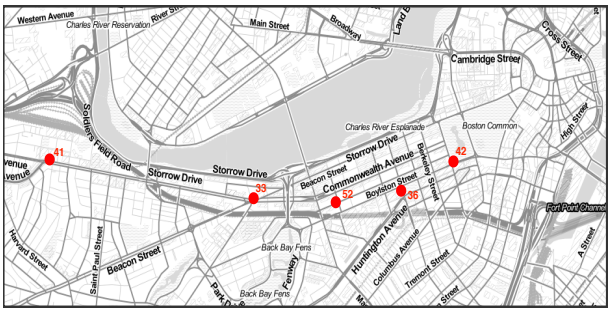


Figure 2: `Hubway` data. IDs of stations picked as a solution to NODE-MONITORING for $k = 5$; 33: Kenmore Sq., 36: Copley Sq./Boston Public Library, 41: Packard's Corner, 42: Boston Public Garden, 52: Newbury St.

the `EdgeGreedy` for dense graphs.

8 Conclusions

In this paper, we introduced the problem of MARKOV CHAIN MONITORING: given a distribution of items over a *Markov chain*, we aim to perform a limited number of monitoring operations, so as to adequately predict the position of items on the chain after one transition step. We studied variants of this problem and provided efficient algorithms to solve them. Our experimental evaluation demonstrated the superiority of the proposed algorithms compared to baselines and the practical utility of the results in real settings.

A natural extension of this work is to to select monitoring operations under incomplete information for initial item distribution – which would allow the operations to be deployed in perpetuity. Another future-work direction is to consider different types of monitoring operations such as those that combine knowledge of item placement on nodes and edges. Finally, we can also consider more complex traffic models (e.g., involving queuing and different transition delays between nodes [6]). In all cases, the algorithms we developed in this paper will serve as the basis for future work.

References

[1] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. In *SODA*, pages 946–957, 2014.

[2] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2), 2001.

[3] Ulrik Brandes and Daniel Fleischer. Centrality measures based on current flow. In *STACS*, pages 533–544, 2005.

[4] Dóra Erdös, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. A divide-and-conquer algorithm for betweenness centrality. In *SDM*, pages 433–441, 2015.

[5] Sainyam Galhotra, Akhil Arora, Srinivas Virinchi, and Shourya Roy. Asim: A scalable algorithm for influence maximization under the independent cascade model. In *WWW*, pages 35–36, 2015.

[6] Robert G Gallager. *Discrete stochastic processes*, volume 321. Springer Science & Business Media, 2012.

[7] Aristides Gionis, Michael Mathioudakis, and Antti Ukkonen. Bump hunting in the dark: Local discrepancy maximization on graphs. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on Data Engineering*, pages 1155–1166. IEEE, 2015.

[8] Aristides Gionis, Evimaria Terzi, and Panayiotis Tsaparas. Opinion maximization in social networks. In *SDM*, pages 387–395, 2013.

[9] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *ICDM*, pages 211–220, 2011.

[10] Vatche Ishakian, Dóra Erdös, Evimaria Terzi, and Azer Bestavros. A framework for the evaluation and management of network centrality. In *SDM*, pages 427–438, 2012.

[11] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *ACM SIGKDD*, page 137, 2003.

[12] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *ACM SIGKDD*, pages 420–429, 2007.

[13] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[14] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of influence networks. *ACM SIGKDD*, 2011.

[15] Charalampos Mavroforakis, Michael Mathioudakis, and Aristides Gionis. Absorbing Random-Walk Centrality - Theory and Algorithms. *ICDM*, 2015.

[16] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Fast and accurate influence maximization on large networks with pruned monte-carlo simulations. In *AAAI*, pages 138–144, 2014.

[17] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[18] Matteo Riondato and Evgenios M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Min. Knowl. Discov.*, 30(2):438–475, 2016.

[19] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

[20] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence Maximization: Near-Optimal Time Complexity Meets Practical Efficiency. In *ACM SIGMOD*, 2014.