Uncertain Graph Processing through Representative Instances

PANOS PARCHAS, Hong Kong University of Science and Technology, Hong Kong
FRANCESCO GULLO, Yahoo Labs, Barcelona, Spain
DIMITRIS PAPADIAS, Hong Kong University of Science and Technology, Hong Kong
FRANCESCO BONCHI, Yahoo Labs, Barcelona, Spain

Data in several applications can be represented as an uncertain graph, whose edges are labeled with a probability of existence. Exact query processing on uncertain graphs is prohibitive for most applications, as it involves evaluation over an exponential number of instantiations. Thus, typical approaches employ Monte-Carlo sampling, which i) draws a number of possible graphs (samples), ii) evaluates the query on each of them, and iii) aggregates the individual answers to generate the final result. However, this approach can also be extremely time consuming for large uncertain graphs commonly found in practice. To facilitate efficiency, we study the problem of extracting a single *representative* instance from an uncertain graph. Conventional processing techniques can then be applied on this representative to closely approximate the result on the original graph.

In order to maintain data utility, the representative instance should preserve structural characteristics of the uncertain graph. We start with representatives that capture the expected vertex degrees, as this is a fundamental property of the graph topology. We then generalize the notion of vertex degree to the concept of n-clique cardinality, i.e., the number of cliques of size n that contain a vertex. For the first problem, we propose two methods: Average Degree Rewiring (ADR), which is based on random edge rewiring, and Approximate B-matching (ABM), which applies graph matching techniques. For the second problem, we develop a greedy approach and a game theoretic framework. We experimentally demonstrate, with real uncertain graphs, that indeed the representative instances can be used to answer, efficiently and accurately, queries based on several metrics such as shortest path distance, clustering coefficient and betweenness centrality.

Categories and Subject Descriptors: E.1 [**Data Structures**]: Graphs and Networks; H.2.4 [**Systems**]: Query processing

General Terms: uncertain graph; possible world; representative; n-clique cardinality

1. INTRODUCTION

Graphs constitute an expressive data representation paradigm used to describe entities (vertices) and their relationships (edges) in a wide range of applications. Sometimes the existence of the relationship between two entities is uncertain due to noisy measurements, inference and prediction models, or explicit manipulation. For instance, in biological networks, vertices represent genes and proteins, while edges correspond to *interactions* among them. Since these interactions are observed through noisy and error-prone experiments, each edge is associated with an uncertainty value [Asthana et al. 2004]. In large social networks, uncertainty arises for various reasons [Adar and Re 2007]; the edge probability may denote the accuracy of a *link prediction* [Liben-Nowell and Kleinberg 2003], or the *influence* of one

DOI:http://dx.doi.org/10.1145/0000000.0000000

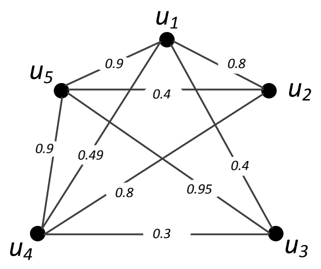


Fig. 1: Uncertain graph $\mathcal{G}$

person on another [Kempe et al. 2003]. Uncertainty can also be injected intentionally for obfuscating the identity of users for privacy reasons [Boldi et al. 2012].

In all these applications the data can be modeled as an *uncertain graph* (also called probabilistic graph), whose edges are labeled with a probability of existence. This probability represents the confidence that the relation corresponding to the edge holds in reality. Given the wide spectrum of application domains, querying and mining uncertain graphs has received considerable attention recently.

1.1. Query processing in uncertain graphs

Let $\mathcal{G} = (V, E, p)$ be an uncertain graph, where function $p : E \to (0, 1]$ assigns a probability of existence to each edge. Following the literature, we consider the edge probabilities independent [Potamias et al. 2010; Jin et al. 2011a; Jin et al. 2011b], and we assume *possible-worlds* semantics [Abiteboul et al. 1987; Dalvi and Suciu 2007]. Specifically, the possible-world semantics interprets $\mathcal{G}$ as a set $\{G = (V, E_G)\}_{E_G \subseteq E}$ of $2^{|E|}$ possible deterministic graphs (worlds), each defined by a subset of E. The probability of observing any possible world $G = (V, E_G) \sqsubseteq \mathcal{G}$ is:

$$\Pr(G) = \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G} (1 - p(e)). \tag{1}$$

Figure 1 illustrates an example of an uncertain graph $\mathcal{G}$, and the associated edge probabilities. Since $\mathcal{G}$ has $|E|$=9 edges, there are 2^9=512 possible worlds. The exponential number of possible worlds usually renders exact query evaluation prohibitive. Indeed, even simple queries on deterministic graphs may be expensive on uncertain graphs. As an example, consider a *reachability* query, which returns true if two input vertices are reachable from each other. The corresponding *reliability* query in uncertain graphs, which outputs the probability that the vertices are connected, is a #**P**-complete problem [Valiant 1979].

In general, for any real-valued query $q : G \to \mathbb{R}$ the most natural choice in the uncertain setting is to consider its expected value, i.e., the average value in all possible worlds, weighted by the probability of the possible worlds.

$$q(\mathcal{G}) = \sum_{G \sqsubseteq \mathcal{G}} q(G) \Pr(G).$$

As one cannot afford to materialize $2^{|E|}$ possible worlds, a common solution is to apply Monte-Carlo sampling, i.e., to assess the query on a subset of random possible worlds. However, sampling is not always a viable option for large graphs [Jin et al. 2011b; Rubino 1998]. Although smart sampling techniques, e.g., [Li et al. 2014], have provably smaller variance and thus require fewer samples for good approximation, still producing a single possible world incurs a non-negligible cost, as it requires generating a random number for each edge

$e \in E$. Moreover, the query may be computationally intensive. For instance, betweenness centrality, a measure of vertex importance in the graph, involves all-pairs shortest path computations, which simply cannot be performed many times (i.e., for each sample) in any graph of even moderate size.

Finally, when the output of the query is a complex structure (e.g., a data mining task producing a set of patterns) the application of sampling is not straightforward as it is not clear how to aggregate query results of different samples. In these cases the semantics of the analysis must be redefined for uncertain graphs, and new ad-hoc algorithms must be developed. However, designing new methods for data analysis problems is not always feasible. Organizations may have already invested in infrastructure (e.g., graph databases, graph processing software, etc.) for deterministic graphs, which they would wish to utilize, regardless of the uncertainty inherent in the data.

Motivated by the above, in this work we study the problem of producing a single representative instance (i.e., a deterministic graph G^*) of a given uncertain graph $\mathcal{G}$. Queries can then be processed efficiently on the deterministic instance using conventional graph algorithms. Clearly, any possible world could be used as a representative of the uncertain graph: the question is *which* instance, among the $2^{|E|}$ possible ones, we should select.

1.2. Properties of a good representative instance

In order to maintain data utility, the representative instance G^* should preserve the expected (underlying) structure of the uncertain graph $\mathcal{G}$. In some sense, we would like G^* to be the *average* or the *expected* graph given the distribution over possible worlds induced by $\mathcal{G}$. As it is not clear how to define such a concept, we focus on extracting a good representative instance that preserves fundamental properties of the graph structure.

Modeling, understanding, and synthetically generating graphs with real-world characteristics is crucial for a variety of simulation-based studies. One key insight in this area, is that one of the most pervasive and persistent characteristics of complex networks is their heavy-tailed degree sequence distribution, as observed on the Web, biological and social networks [Mihail and Vishnoi 2002; Faloutsos et al. 1999]. Following this observation, many approaches generate synthetic graphs with a target degree sequence: these studies have shown that such graphs exhibit several characteristics that resemble those of real networks [Aiello et al. 2000; Tangmunarunkit et al. 2002; Chung and Lu 2002; Chung et al. 2003].

In the theory of *structural network controllability* [Liu et al. 2011], one key property is the number of *driver nodes*, i.e., the set of nodes that can guarantee full control over the network. Here control means the ability to guide a dynamical system from any initial state to any desired final state in finite time, with a suitable choice of the driver nodes. [Liu et al. 2011] show that by fully randomizing real-world networks (i.e., generating an Erdős-Rényi random graph with the same number of nodes and edges as the original network), changes the number of driver nodes dramatically, failing to preserve the topological characteristics of the original network. Instead, when *degree-preserving randomization* is used (i.e., maintaining the degree of each node while randomly rewiring the edges), the number of driver nodes remains almost unchanged, indicating that network controllability is captured by the degree distribution.

Motivated by the above, we study the problem of extracting, from an uncertain graph, the representative instance that better preserves the expected degree of each vertex. Going a step further, we generalize the notion of vertex degree to the concept of *n-clique cardinality* of a vertex u, i.e., the number of cliques of size n that contain u. In particular, the degree of a vertex is equivalent to its 2-clique cardinality, whereas its triangle connectivity corresponds to its 3-clique cardinality. The notion of n-clique is extended naturally to $n > 3$. Intuitively, this generalization aims at capturing the expected structure in the neighborhood of vertices. The importance of neighborhood connectivity in the overall structure of a deterministic graph has been highlighted in graph triangulation [Hu et al. 2013] and complex network modeling [Mahadevan et al. 2006].

1.3. Contributions and roadmap

For the first problem, extracting representative instances that capture the expected vertex degree sequence, we propose two methods: Average Degree Rewiring (ADR) and Approximate B-matching (ABM). ADR involves two phases: first, it generates an instance with the same average vertex degree as the uncertain graph; then, it randomly rewires edges if they lead to better approximation of the vertex degrees. ABM applies b-matching [Hougardy 2009] to obtain an initial instance, which then improves using weighted maximum bipartite matching.

For the problem of extracting the representative instance that best preserves the expected n-clique cardinality (for $n \geq 2$), we develop GREEDY and GAME. GREEDY arranges the edges in a dynamic heap and greedily inserts the one that leads to the best approximation of the expected n-clique cardinality. GAME applies a game theoretic framework that models the edges of the uncertain graph as players of an *exact potential game*, and uses *best response dynamics* [Monderer and Shapley 1996] to generate the representative.

Our extensive experimental evaluation on real datasets confirms that an instance whose vertices have n-clique cardinality close to their expected value (even for small values of n, e.g. $n = \{2, 3\}$), captures several structural properties of the uncertain graph, including *shortest path distance*, *betweenness centrality* and *clustering coefficient*.

Summarizing, the contributions of the paper are:

(1) We propose a novel framework for querying uncertain graphs, based on the extraction of representatives, which has vast potential due to the prevalence of uncertain graphs in several modern applications, the large data volume involved, and the high cost of uncertain graph processing.
(2) We propose ADR, ABM, GREEDY and GAME, which efficiently generate representatives with desirable properties. Hence, they are applicable to large uncertain graphs of millions of vertices and edges.
(3) We experimentally demonstrate that the extracted representatives are accurate in answering a variety of common graph statistics. Moreover, query processing through representative instances requires only a fraction of the time spent by conventional Monte Carlo techniques.

The rest of the paper is organized as follows. Section 2 provides an overview of the related work and the necessary background. Section 3 formally defines the problems we tackle in this work. Section 4 introduces ADR and ABM that aim at approximating the expected vertex degrees. Section 5 discuses GREEDY and GAME that generate representatives preserving the expected n-clique cardinality for $n \geq 2$. Section 6 contains an extensive experimental evaluation on real datasets, and Section 7 concludes the paper.

2. BACKGROUND AND RELATED WORK

[Parchas et al. 2014] contains a short version of this article, where we focus on representatives based on the expected vertex degrees, using ADR and ABM. This work extends [Parchas et al. 2014] by generalizing to the concept of n-clique cardinality, and proposing GREEDY and GAME. Moreover, all the sections and the experimental evaluation have been revised and enhanced.

Section 2.1 overviews uncertain graphs. Section 2.2 presents research on vertex degree distribution due to its relevance to our problem. Section 2.3 provides background on best-response algorithms and exact potential games, as they are used by our GAME method.

2.1. Uncertain graphs

Uncertain relational databases have been well studied from different perspectives, such as SQL query evaluation, mining, ranking and top-k queries [Aggarwal and Yu 2009]. However, in many application domains, such as social, biological, and mobile networks, graphs serve as better models than relational tables. Processing on uncertain graphs can be classified into three main approaches: i) queries based on shortest path distances and reliability, ii) pattern mining and graph decomposition, and iii) subgraph (similarity) search.

Towards the first direction, [Jin et al. 2011b] introduce the *distance-constrained reachability* query, which, given two vertices s and t, and a threshold d, returns the probability that the distance from s to t is less than d. The authors propose two estimators for the distance-constrained reachability query that have provably less variance than naïve Monte Carlo methods. [Potamias et al. 2010] redefine traditional nearest neighbor queries by using statistical distance metrics (e.g. majority, median). These metrics are computed by applying Dijkstra's algorithm on a subset of the possible worlds. Similarly, based on the possible world semantics, [Yuan et al. 2010] return shortest paths, whose probability exceeds an input threshold. Finally, [Khan et al. 2014] study the problem of reliability search, i.e., the discovery of all vertices reachable from a set of query vertices with probability higher than a given threshold. Their solution builds a novel index based on hierarchical clustering of the vertices. At query time, the index is traversed in a bottom up fashion and returns a set of candidate result vertices, which are then validated through Monte Carlo sampling.

In the second line of research, Zou et al. investigate mining frequent subgraphs [Zou et al. 2010b] and top-k maximal cliques [Zou et al. 2010a] in uncertain graphs. [Moustafa et al. 2014] propose efficient algorithms for subgraph pattern matching for graphs, where in addition to edges, vertices are also uncertain. [Jin et al. 2011a] aim at finding subgraphs that are connected with high probability, whereas [Bonchi et al. 2014] decompose the uncertain graph in probabilistic k-cores, i.e., maximal subgraphs, whose vertices have degree greater than k with high probability. [Kollios et al. 2013] define clustering using a modified graph edit distance, whereas [Liu et al. 2012] find reliable clusters, i.e., clusters that have a good chance of staying connected among the different instantiations (possible worlds).

In the third direction of research, [Yuan et al. 2011] propose a feature-based framework for subgraph search, while [Yuan et al. 2012] study subgraph similarity. In a rather different type of research, [Boldi et al. 2012] intentionally inject uncertainty in a social graph in order to obfuscate the identity of its users. Finally, in a recent work [Li et al. 2014], improve the naïve Monte Carlo by performing a smarter sampling that has provably smaller variance and requires fewer samples for good approximation.

2.2. Degree sequences and distributions

A sequence of non-negative integers $d = \{d_1, d_2, \cdots, d_n\}$, with $d_1 \geq d_2 \geq \cdots \geq d_n$ is called *graphic*, if it is the degree sequence of some *simple* graph G. In such case, we say that G *realizes* the sequence d. [Erdös and Gallai 1960] describe the necessary and sufficient conditions for a sequence to be graphic. Graphical realization is fundamental for simulation of network properties [Del Genio et al. 2010]. For instance, in epidemiology of sexually transmitted diseases [Liljeros et al. 2001], anonymous surveys collect only the *number* of sexual partners of individuals, rather than their identity. The construction of the implied *contact graph* is crucial for the simulation of spread scenarios. In order to capture the observed data, a common approach is to generate graphs that realize a predicted degree sequence of the complex network [Hakimi 1962; Lovász 1970]. However, some structural characteristics (e.g. density, connectivity, etc.) depend heavily on the vertex processing order in [Hakimi 1962]. Thus, several techniques use the output of [Hakimi 1962] as a seed, and take additional steps to *randomize* the graph [Blitzstein and Diaconis 2011], or induce specific characteristics, such as connectivity. These techniques perform local search by swapping edges in order to reach desired properties. We follow a similar approach in the proposed ADR algorithm.

Another related line of research aims at the more general *degree distributions*. According to [Mihail and Vishnoi 2002] the degree distribution of a graph is the most frequently used topology characteristic. For instance, observations on the Internet's degree distribution had a huge impact on network topology research [Faloutsos et al. 1999]. [Mahadevan et al. 2006] model complex networks by reproducing degree distributions, extracted from observed data. They generalize their approach to *joint* degree distributions called *dK-series*, i.e., probability distributions of vertex pairs, triplets etc. For instance, the *2K-series* refers to the probability distribution of pairs of vertices, i.e., the probability that a vertex with degree k is connected to a vertex with degree k'. In their experimental evaluation, they perform several queries on both the original and the simulated graph, concluding that their method with $d = 3$ (i.e., triplets of vertices) generates models that approximate very well the original graphs in all metrics. Inspired by their work, we focus on similar distributions in uncertain graphs and we evaluate our methods on similar metrics.

The popular Chung-Lu model [Chung and Lu 2002] predicts the average distance of random graphs constructed with a given *expected* degree sequence, in order to justify facts such as the *small world phenomenon* [Kleinberg 2006]. Specifically, assuming a weight w_u for each vertex u, *any* edge (u, v) exists with probability $\rho = \frac{w_u w_v}{\sum_i w_i}$. On the other hand, in our uncertain graph model, edge probabilities are given explicitly as an input to the problem, instead of being related to vertex weights. The Chung-Lu model has been generalized beyond degree sequence, to output graphs conforming with spectral properties, or connectivity [Leskovec et al. 2010].

Similar in spirit to graph generation models is the literature on *subgraph sampling*, whose goal is to generate a subgraph of an input deterministic graph that preserves well the structural properties, while containing considerably fewer nodes [Leskovec and Faloutsos 2006; Hübler et al. 2008]. This line of research differs from our work as i) we focus on uncertain graphs, thus we aim at preserving the structural properties in expectation, and ii) our goal is to extract a representative deterministic graph having the same number of nodes. In our experiments, we use *forest fire* [Leskovec and Faloutsos 2006], a subgraph sampling approach, in order to reduce the size of real graphs, for computationally intensive queries.

2.3. Background in game theory

Players of a *strategic game* compete on common resources. The objective of each player is to minimize its own cost, defined by a cost function. After an initialization step, which assigns a strategy to each player, the game proceeds in rounds. At every round each player chooses a strategy that minimizes its cost, given the other players' strategies. However, as a player's strategy affects the cost of others, each change may cause other players to alter their strategy as well. This process is called *best-response dynamics* [Monderer and Shapley 1996]. Players do not consider the effect their strategy has on the future of the game, which results in the dynamical rule often called *myopic best response*.

Formally, a strategic game is a triplet $< E, \{S_e\}_{(e \in E)}, \{C_e : S\}_{(e \in E)} \to R >$, where E is the set of players, S_e is the set of possible strategies of player $e \in E$, and $C_e(S)$ is the cost function that e wishes to minimize, considering the strategies $S = \times_{(e \in E)} S_e$ of all other players. Due to the interdependence of the players' decisions, many games never terminate; the decision of a player e may trigger a change in strategy of e', which in turn might force e to reconsider, and so on. If a strategic game terminates, we say that it has a *pure Nash equilibrium*, i.e., there exists a specific choice of strategies $s_e \in S_e, \forall e \in E$ such that no player has incentive to change strategy.

Table I: List of frequent symbols

Symbol	Definition
$\mathcal{G}$	uncertain graph
G	instance (possible graph) of $\mathcal{G}$
n	target clique size
G^*	a representative instance of $\mathcal{G}$
E^*	set of edges in G^*
$\gamma_n(u, G)$	n-clique cardinality of vertex u in G
$dis_n(u, G)$	n-discrepancy of u in G, i.e., $\gamma_n(u, G) - [\gamma_n(u, \mathcal{G})]$
$deg(u, G)$	degree of vertex u in G, i.e., $\gamma_2(u, G)$
$\Delta_n(G)$	overall discrepancy of an instance G of $\mathcal{G}$
$Q_m(e, E^*)$	set of m-cliques that contain both endpoints u and v of edge $e = (u, v)$ in E^*
$\mathbf{P}$	sum of probabilities $\sum_{e \in E} p_e$

Potential games constitute a special class of strategic games, in which the incentive of all players to change their strategy can be expressed using a single global function $\Phi : \times_{(e \in E)} S_e \to \mathbb{R}$ called the *potential function.* Let $\bar{S}_e$ denote the strategies of all players other than e, i.e., $\bar{S}_e = \{s_1, \cdots, s_{e-1}, s_{e+1}, \cdots, s_{|E|}\}$. A potential game is called *exact*, when the change of a player's cost due to his strategy update is reflected exactly in the potential function, i.e.,

$$C_e(s'_e, \bar{S}_e) - C_e(s_e, \bar{S}_e) = \Phi(s'_e, \bar{S}_e) - \Phi(s_e, \bar{S}_e)$$

The theory of best response dynamics on exact potential games ensures that they always converge to a Nash Equilibrium, independently of the initialization step [Monderer and Shapley 1996]. The proposed GAME algorithm constitutes an exact potential game.

3. PROBLEM DEFINITION

Let $\mathcal{G} = (V, E, p)$ be an undirected uncertain graph, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, and $p : E \to (0, 1]$ is a function that assigns a probability of existence to each edge. For the sake of brevity, we denote the probability $p(e)$ of any edge $e \in E$ with p_e. We assume independent edge probabilities and possible world semantics, i.e., $\mathcal{G}$ is interpreted as a set $\{G = (V, E_G)\}_{E_G \subseteq E}$ of $2^{|E|}$ possible deterministic graphs (worlds), each defined by a subset of E.

Since most query processing tasks are very expensive for large uncertain graphs, we propose the extraction of a deterministic *representative instance* $G^* \sqsubseteq \mathcal{G}$ that captures the underlying properties of $\mathcal{G}$. Then, queries on the uncertain $\mathcal{G}$ can be efficiently processed using deterministic algorithms on G^*. Consider for example a nearest-neighbor query. State-of-the-art approaches to this query type perform Dijkstra expansions on multiple samples [Potamias et al. 2010]. Depending on the definition of the distance measure, expansion for some samples can be avoided or terminated early, when it cannot improve the current result. Nevertheless, the method has usually very high cost due to the large number of samples it requires. On the other hand, the same query in our framework can be processed efficiently by applying any deterministic nearest neighbor algorithm on the representative G^*. As shown in our experimental evaluation, the representatives extracted by our algorithms indeed capture well the relevant properties of $\mathcal{G}$, in this case shortest path distances. Thus, the query on G^* is expected to return a good approximation of the nearest neighbor set.

Section 3.1 discusses desired properties of a representative instance. Section 3.2 introduces the problem of generating representatives based on the expected degree of vertices. Section 3.3 generalizes to instances that preserve the neighborhood connectivity of vertices. Table I contains the most common symbols throughout the paper.

3.1. Representative Instance

The representative instance G^* should conform well with the structural properties of $\mathcal{G}$ in *expectation.* A direct extraction of the "expected graph" from all possible worlds yielded by $\mathcal{G}$ is not easily achievable, as the definition of expected graph is intrinsically ill-posed. Indeed, the notion of expected value of any probability distribution needs i) an ordering among the points/objects of the domain of the distribution, and ii) a way of averaging (aggregating) among such objects. In our context the domain objects are graphs, hence it is not clear how to carry over either i) or ii). Instead, we propose a criterion that aims at preserving the expected structure of individual vertices. Specifically, we use the notion of *n-clique cardinality*, defined as follows:

Definition 3.1. The *n-clique cardinality* $\gamma_n(u, G)$ of a vertex u in a deterministic graph G is the number of cliques of size n that contain u in G.

Definition 3.2. The *expected n-clique cardinality* $[\gamma_n(u, \mathcal{G})]$ of a vertex u in an uncertain graph $\mathcal{G}$ is the expected number of cliques of size n that contain u in $\mathcal{G}$.

When the graph is implied, we write for convenience $\gamma_n(u)$ and $[\gamma_n(u)]$, respectively. The following lemma derives the expected n-clique cardinality of a vertex u in $\mathcal{G}$.

LEMMA 3.3. *Given an uncertain graph $\mathcal{G} = (V, E, p)$, an integer $n \geq 2$ and a vertex $u \in V$, the expected n-clique cardinality of u is*

$$[\gamma_n(u)] = \sum_{c \in \mathcal{Q}_n(u)} \prod_{\substack{e=(u_i,u_j),\\ i<j}} p_e \tag{2}$$

where $\mathcal{Q}_n(u)$ is the set containing all cliques of size n that involve vertex u in $\mathcal{G}$.

PROOF. Let an uncertain graph $\mathcal{G} = (V, E, p)$ and a vertex $u \in V$. Due to the independence of edge probabilities, a set of n vertices $\{v_1, v_2, \cdots, v_n\}$ forms a clique c with probability $p_c = \prod_{i<j} p(v_i, v_j)$. In the entire vertex set V, there are $q = \binom{|V|-1}{n-1}$ different possible n-cliques that include u. Let $\mathcal{Q}_n(u) = \{c_1, c_2, \cdots, c_q\}$ be the ordered set of all possible n-cliques. Using an indicator variable X_i:

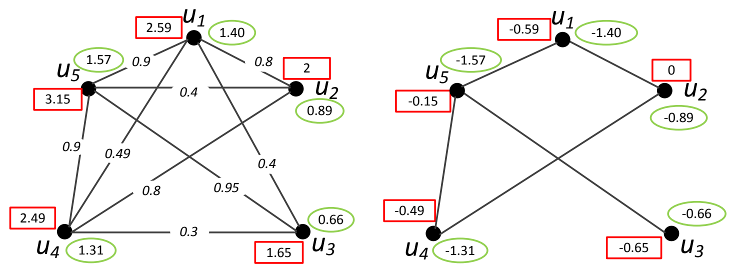
$$X_i = \begin{cases} 1, & \text{if } c_i \text{ forms an } n\text{-clique} \\ 0, & \text{otherwise} \end{cases}$$

The total number of n-cliques that contain u is $X = \sum_{i=1}^{q} X_i$. The expected value of X is thus,

$$E[X] = E[\sum_{i=1}^{q} X_i] = \sum_{i=1}^{q} E[X_i] = \sum_{c \in \mathcal{Q}_n(u)} (1 \cdot p_c - 0 \cdot (1 - p_c)) = \sum_{c \in \mathcal{Q}_n(u)} \prod_{\substack{e=(u_i,u_j),\\ i<j}} p_e$$

□

The 2-clique cardinality of a vertex corresponds to its *expected degree.* For $n = 3$, the expected 3-clique cardinality of a vertex u is the sum of probabilities of the triangles containing u. The rectangles (resp. ellipses) of Figure 2(a) contain the expected 2-clique (resp. 3-clique) cardinality of each vertex, computed by Lemma 3.3. For instance, $[\gamma_2(u_2)] = p_{(u_1,u_2)} + p_{(u_2,u_4)} + p_{(u_2,u_5)} = 0.8 + 0.4 + 0.8 = 2$. Accordingly, $[\gamma_3(u_2)]$ equals the sum of the probabilities of the triangles $\{u_1, u_2, u_4\}, \{u_1, u_2, u_5\}, \{u_2, u_4, u_5\}$ that contain u_2, i.e., $[\gamma_3(u_2)] = 0.8 \cdot 0.8 \cdot 0.49 + 0.8 \cdot 0.4 \cdot 0.9 + 0.8 \cdot 0.9 \cdot 0.4 = 0.89$.



(a) $\mathcal{G}$ and expected n-clique cardinalities, $n = \{2, 3\}$ (b) representative of MP and $dis_n(u)$, $n = \{2, 3\}$

Fig. 2: Uncertain graph and its most probable instance

Definition 3.4. The *n-discrepancy* $dis_n(u, G)$ of a vertex u in an instance $G \sqsubseteq \mathcal{G}$ is the difference of u's n-clique cardinality in G to its expected n-clique cardinality, i.e., $dis_n(u, G) = \gamma_n(u, G) - [\gamma_n(u, \mathcal{G})]$. If the graph instance is implied, we equivalently write $dis_n(u)$.

Definition 3.5. Given an uncertain graph $\mathcal{G} = (V, E, p)$ and an integer $n \geq 2$, the *discrepancy* $\Delta_n(G)$ of a possible graph $G \sqsubseteq \mathcal{G}$ is defined as

$$\Delta_n(G) = \sum_{u \in V} |dis_n(u)| \tag{3}$$

A straightforward way to generate a representative of an uncertain graph is to consider the instance with the highest probability [Potamias et al. 2010]. According to Equation 1, this *most probable* (MP) instance corresponds to the graph containing all the edges e that have probability $p_e \geq 0.5$. Since MP does not conform with any structural property of $\mathcal{G}$, it is expected to be a poor representative. For instance, if the probability of all edges is below 0.5, then MP contains no edges. Figure 2(b) illustrates the MP representative of the uncertain graph of Figure 2(a), where the rectangles and ellipses next to each vertex u correspond to $dis_2(u)$ and $dis_3(u)$ respectively. Given the high importance of the individual vertex degrees, in the following we distinguish the special case where $n = 2$ in Definition 3.5.

3.2. Vertex degree

From Lemma 3.3, the expected degree of a vertex u is $[deg(u)] = \sum_{e=(u,v)} p_e$, and its 2-discrepancy is $dis_2(u, G) = \gamma_2(u, G) - [\gamma_2(u, \mathcal{G})] = deg(u, G) - [deg(u, \mathcal{G})]$. Similarly, the overall discrepancy Δ_2 is simplified to $\Delta_2(G) = \sum_{u \in V} |dis_2(u)|$. The first problem we tackle in this work is:

PROBLEM 1 (2-REPRESENTATIVE INSTANCE). *Given an uncertain graph $\mathcal{G} = (V, E, p)$, find a possible graph $G_2^* \sqsubseteq \mathcal{G}$ such that:*

$$G_2^* = \arg\min_{G \sqsubseteq \mathcal{G}} \Delta_2(G) \quad \square$$

Intuitively, Problem 1 aims at finding an instance, such that the degree of each vertex is as close as possible to its expected value. Characterizing the complexity class of Problem 1 is non-trivial and represents an interesting open question. Our conjecture is that the problem is hard, or at least not solvable exactly in reasonable time for large graphs. To this purpose, note that Problem 1 can alternatively be formulated as an integer linear programming problem. Each edge $e \in E$ is assigned a binary variable $x_e = \{0, 1\}$, where $x_e = 1$ if and only if e is included in the result set. Then, the discrepancy of a vertex u in G can be expressed as $dis_2(u) = \sum_{e=(u,v)\in E}(x_e - p_e)$. Thus, Problem 1 becomes:

$$\begin{aligned} &\min |\mathbf{A}(\mathbf{x}-\mathbf{p})| \\ &\mathbf{x} = \{0,1\}^{|E|} \end{aligned} \tag{4}$$

where $\mathbf{p} = (0,1]^{|E|}$ is the vector containing the edge probabilities of the input uncertain graph $\mathcal{G}$ and $\mathbf{A} = \{0,1\}^{|V|\times|E|}$ is the incidence matrix of $\mathcal{G}$. The formulation in Equation 4 corresponds to a special case of the *closest vector* problem, which is known to be **NP**-hard [Micciancio 2001]. Moreover, as discussed in Section 4.3, when *all* expected degrees are integers, Problem 1 can be solved by *b-matching* algorithms, among which the fastest runs in $O(|E|^{3/2})$ time [Micali and Vazirani 1980]. For the general case of real degrees, a brute-force approach would generate all $2^{|E|}$ possible worlds, and select the one minimizing the objective function of Problem 1. Given that our main goal is to provide solutions that are scalable enough to deal with the large size of real-world graphs, we directly aim at approximate, but efficient algorithms.

3.3. Neighborhood connectivity

For certain graph metrics (e.g. clustering coefficient), the connectivity among the neighbors of a vertex u is important. To capture such scenarios, we further explore larger values of n-clique cardinality, in which case the targeted problem is:

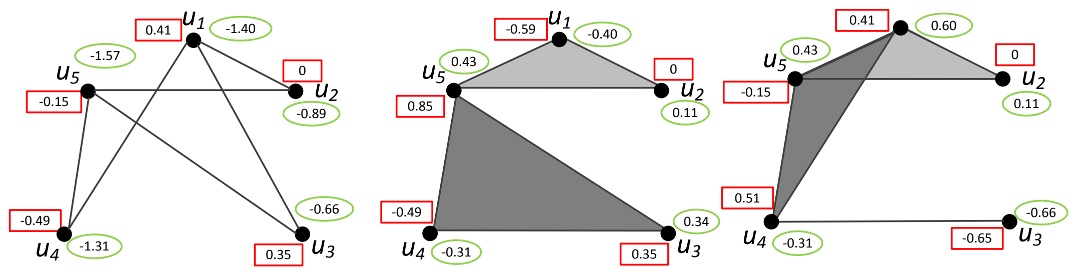
PROBLEM 2 (n-REPRESENTATIVE INSTANCE). *Given an uncertain graph $\mathcal{G} = (V, E, p)$ and two integers $2 \le l \le n$, find a possible graph $G^*_{l,n} \sqsubseteq \mathcal{G}$ such that:*

$$G^*_{l,n} = \arg\min_{G \sqsubseteq \mathcal{G}} \sum_{m=l}^{n} \Delta_m(G) \quad \square$$

Problem 2 aims at extracting an instance that preserves the m-clique connectivity of the vertices, for values of m within a given range $[l, n]$. If $l = n$, to simplify notation, we denote $G^*_{n,n}$ as G^*_n. Problem 1 is a special case of Problem 2, where $l = n = 2$. Since Problem 2 constitutes a generalization of Problem 1, it is also expected to be **NP**-Hard. Our framework is generic and can be directly applied to extracting representatives $G^*_{l,n}$ with arbitrary values of n. However, we focus on values of n up to 3, for the following reasons. i) The complexity of finding n-cliques of a vertex with degree d is $\mathcal{O}(d^{n-1})$ [Nešetřil and Poljak 1985]. Thus, although Lemma 3.3 still applies, for $n > 3$ its computation is prohibitive for realistic graphs. ii) Recall from Section 2.2, in the context of deterministic graphs, [Mahadevan et al. 2006] model complex networks by reproducing joint degree distributions called dK-series, i.e., probability distributions of vertex pairs, triplets etc. Their experimental evaluation concludes that $d = 3$ (i.e., triplets of vertices, corresponding to $n = 3$ in our framework) generates models that approximate very well the original graphs in all evaluated metrics.

Figure 3 shows the optimal representatives for different values of m in the range $[2, 3]$. Specifically, Figure 3(a) illustrates the representative G^*_2 that minimizes Δ_2, Figure 3(b) the representative G^*_3 that minimizes Δ_3, and Figure 3(c) the representative $G^*_{2,3}$ that minimizes $\Delta_2 + \Delta_3$, i.e., the objective of Problem 2 for $l = 2$ and $n = 3$. The rectangles and ellipses next to each vertex u correspond to $dis_2(u)$ and $dis_3(u)$ respectively. Observe that G^*_3 and $G^*_{2,3}$ preserve the triangle connectivity of the vertices, as shown by the shaded regions. For instance, vertex u_5 participates in two triangles of both representatives since its expected 3-clique cardinality is 1.57. On the other hand, the representatives MP (Figure 2(b)) and G^*_2 (Figure 3(a)) do not contain any triangle.

Table II summarizes the overall discrepancies (i.e., Δ_2, Δ_3 and $\Delta_2 + \Delta_3$) of the various representatives (i.e., MP, G^*_2, G^*_3 and $G^*_{2,3}$). The values in bold correspond to the minima of each column. Although $G^*_{2,3}$ is slightly worse than G^*_2 and G^*_3 in terms of Δ_2 and Δ_3



(a) G_2^* (b) G_3^* (c) $G_{2,3}^*$

Fig. 3: Representative instances

respectively, it yields better $\Delta_2 + \Delta_3$. Intuitively, $G_{2,3}^*$ combines the desirable properties of G_2^* and G_3^*, i.e., it has similar edges to G_2^* and similar triangles to G_3^*. Because of this, as shown in our experiments, $G_{2,3}^*$ has balanced performance on all evaluated metrics. On the other hand, representatives that minimize Δ_3, underperform for metrics unrelated to neighborhood connectivity, e.g., shortest path distance.

Table II: Representatives and the corresponding discrepancies

representative	Δ_2	Δ_3	$\Delta_2 + \Delta_3$
MP	1.88	5.83	7.71
G_2^*	**1.40**	5.83	7.23
G_3^*	2.28	**1.59**	3.87
$G_{2,3}^*$	1.72	2.11	**3.83**

Representative instances vastly accelerate query processing on uncertain graphs because: i) they eliminate the overhead of generating a large number of samples and ii) the query is executed once (on the representative) instead of numerous times (for each sample). Sections 4 and 5 propose algorithms that generate representative instances, using the objective functions of Problems 1 and 2 respectively.

4. ALGORITHMS FOR MINIMIZING VERTEX DEGREE DISCREPANCY

The following methods aim explicitly at minimizing the vertex degree discrepancy. Section 4.1 discusses a benchmark approach PS. Sections 4.2 and 4.3 present ADR and ABM, respectively.

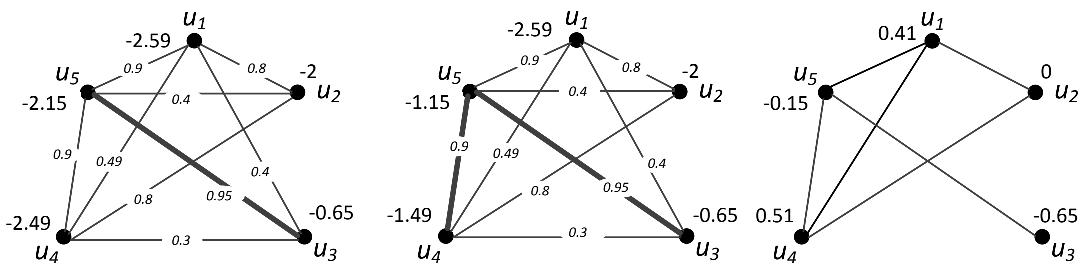
4.1. Benchmark solution

Probability Sorting (PS) first sorts the edges of the graph in non-increasing order of their probabilities. Then, at each iteration, the algorithm considers an edge $e = (u, v)$ of the sorted list, and includes it to the result set, if $|dis_2(u) + 1| + |dis_2(v) + 1| < |dis_2(u)| + |dis_2(v)|$, i.e., the addition of e decreases the total discrepancy. The complexity of the algorithm is $O(|E| \cdot \log |V|)$ because it is dominated by the sorting step. Algorithm 1 presents the pseudocode of PS. Intuitively, each edge (u, v) affects only the degrees of vertices u, v. Thus, if the condition in line 4 is satisfied, the value of Equation 3 decreases, leading to a better solution. Due to the initial sorting, the most probable edges are considered first. Such edges have large contribution to the expected degrees of the incident vertices, and at the same time they lead to a highly probable representative.

Algorithm 1 PS

```
Input: uncertain graph $\mathcal{G} = (V, E, p)$
Output: representative $G^* = (V, E^*)$
1: $E^* \leftarrow \emptyset$
2: sort $E$ in non-increasing order of their probabilities
3: for each $e = (u, v) \in E$ do
4:     if $|dis_2(u) + 1| + |dis_2(v) + 1| < |dis_2(u)| + |dis_2(v)|$ then
5:         $E^* \leftarrow E^* \cup \{e\}$
```

Figure 4 illustrates the execution of PS on the uncertain graph of Figure 2(a). At the first iteration, PS picks the edge (u_3, u_5) with the highest probability and adds it to the result set E^*, containing the edges of the representative G^*. Figure 4(a) shows E^* after the first iteration, where the number next to vertex u denotes $dis_2(u)$. At the second iteration, PS considers edge (u_4, u_5). The inclusion of (u_4, u_5) in E^*, decreases the total discrepancy of u_4 and u_5 from $2.49 + 2.15 = 4.64$ to $1.49 + 1.15 = 2.64$ (see Figure 4(b); edges of E^* are in bold). The procedure continues until all edges have been examined, at which point PS returns the representative of Figure 4(c) with $\Delta_2(G^*) = 0.41 + 0 + 0.65 + 0.51 + 0.15 = 1.72$. Note that for the same example, the representative produced by MP (Figure 2(b)) yields overall discrepancy $\Delta_2(G) = 1.88$.



(a) first iteration of PS (b) second iteration of PS (c) representative of PS

Fig. 4: PS example

4.2. Average Degree Rewiring (ADR)

Average Degree Rewiring (ADR) involves two phases: 1) it creates an instance $G_1 = (V, E_1)$ of the uncertain graph that preserves the average vertex degree and 2) it iteratively improves G_1 by rewiring, i.e., replacing edges in E_1, so that the total discrepancy is reduced. The following lemma describes the efficient computation of the expected average degree for an uncertain graph.

LEMMA 4.1. *The expected average degree $[deg(\mathcal{G})]$ of an uncertain graph $\mathcal{G} = (V, E, p)$, is $[deg(\mathcal{G})] = \frac{2}{|V|}\mathbf{P}$, where $\mathbf{P}$ is the sum of all the edge probabilities in $\mathcal{G}$.*

PROOF. By definition, the average degree $deg(G)$ of a deterministic graph G is equal to $deg(G) = \frac{1}{|V|}\sum_{u \in V} deg_u$. Due to the linearity of expectation, the expected average degree is:

$$[deg(\mathcal{G})] = \left[\frac{1}{|V|} \cdot \sum_{u \in V} deg_u\right] = \frac{1}{|V|} \cdot \sum_{u \in V} [deg_u] = \frac{2}{|V|} \cdot \sum_{e \in E} p_e = \frac{2}{|V|} \cdot \mathbf{P} \quad \square$$

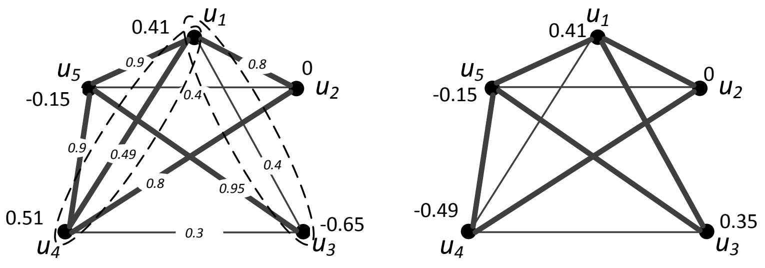
Given Lemma 4.1, a representative that preserves $[deg(\mathcal{G})]$ should contain $\mathbf{P}$ edges. Initially, ADR rounds $\mathbf{P}$ to the closest integer $\lfloor \mathbf{P} \rceil$ and sorts the edges in descending order of their probabilities. Consequently, it iterates through the sorted list, and samples each edge e with probability p_e, until it has included $\lfloor \mathbf{P} \rceil$ edges. Algorithm 2 illustrates the pseudocode of ADR, where lines 1-7 correspond to Phase 1.

Algorithm 2 Average Degree Rewiring (ADR)

```
Input: uncertain graph $\mathcal{G} = (V, E, p)$, steps
Output: representative $G^* = (V, E^*)$
                                                              // Phase 1
 1: $E_1 \leftarrow \emptyset$, $i \leftarrow 0$
 2: $\mathbf{P} \leftarrow \sum_{e \in E} p_e$
 3: sort $E$ in non increasing order of their probabilities
 4: while $|E_1| < \lfloor \mathbf{P} \rceil$ do
 5:     $e \leftarrow E.next()$; $r \leftarrow$ random number $\in [0, 1]$
 6:     if $r \leq p_e$ then
 7:         $E_1 \leftarrow E_1 \cup e$
                                                              // Phase 2
 8: $dis_2(u) = deg(u) - [deg_2(u)], \forall u \in V$
 9: for $i = 1..\#rounds$ do
10:     for each $u \in V$ do
11:         pick a random edge $e_1 = (u, v)$ from $E_i$
12:         pick a random edge $e_2 = (x, y)$ from $E \setminus E_i$
13:         $d_1 \leftarrow |dis_2(u) - 1| + |dis_2(v) - 1| - (|dis_2(u)| + |dis_2(v)|)$
14:         $d_2 \leftarrow |dis_2(x) + 1| + |dis_2(y) + 1| - (|dis_2(x)| + |dis_2(y)|)$
15:         if $d_1 + d_2 < 0$ then
16:             $E_{i+1} \leftarrow (E_i - \{e_1\}) \cup \{e_2\}$
17:             update $dis_2$ for $\{u, v, x, y\}$
18: $E^* \leftarrow E_i$
```

Phase 2 starts with E_1. At each iteration / round i, let the current set of edges be E_i. For each vertex $u \in V$, ADR randomly picks two edges $e_1 = (u, v) \in E_i$ and $e_2 = (x, y) \in E \setminus E_i$ (lines 10-11) and computes $d_1 \leftarrow |dis_2(u) - 1| + |dis_2(v) - 1| - (|dis_2(u)| + |dis_2(v)|)$ and $d_2 \leftarrow |dis_2(x) + 1| + |dis_2(y) + 1| - (|dis_2(x)| + |dis_2(y)|)$. Specifically, d_1 is the difference of the absolute discrepancies of u and v, caused by the removal of e_1. Accordingly, d_2 is the difference of the absolute discrepancies of x and y caused by the addition of edge e_2. ADR replaces e_1 with e_2 if $d_1 + d_2 < 0$, i.e., the swapping of edges decreases the overall discrepancy. Since the total number of edges remains $\lfloor \mathbf{P} \rceil$, the expected average degree of $\mathcal{G}$ is preserved throughout the process. The procedure terminates after a user-defined number of rounds. The value of *#rounds* depends on the desired trade-off between quality and efficiency. Sorting the edges has cost $O(|E| \cdot \log |V|)$. Each round incurs constant cost for each vertex $u \in V$. Thus, the complexity of ADR is $O(|E| \cdot \log |V| + \#rounds \cdot |V|)$.

We illustrate the application of ADR on the uncertain graph of Figure 2(a). Initially, ADR computes $\mathbf{P} = 5.94$ and approximates it to the closest integer $\lfloor \mathbf{P} \rceil = 6$. Then, it picks the 6 most probable edges of the graph and forms the set $E_1 = \{(u_1, u_2), (u_1, u_4), (u_1, u_5), (u_2, u_4), (u_3, u_5), (u_4, u_5)\}$. Figure 5(a) depicts the edges of E_1 with bold lines, and shows the resulting vertex degree discrepancies next to each vertex. The value of the total discrepancy at this stage is $\Delta_2 = 1.72$. Next, ADR starts the second phase. Assume that at round 1 the algorithm randomly considers the replacement of $e_1 = (u_1, u_4) \in E_1$ with $e_2 = (u_1, u_3) \in E \backslash E_1$. Since $d_1 = 0.41 + 0.49 - (0.41 + 0.51) = -0.02$, $d_2 = 0.41 + 0.35 - (0.41 + 0.65) = -0.3$ and $d_1 + d_2 = -0.32 < 0$, the edges are swapped (Figure 5(b)). Intuitively, the swapping reduces the overall discrepancy by $|d_1 + d_2|$. The discrepancy of the new instance $E_2 = \{(u_1, u_2), (u_1, u_3), (u_1, u_5), (u_2, u_4), (u_3, u_5), (u_4, u_5)\}$ is $\Delta_2' = \Delta_2 - |d_1 + d_2| = 1.4$. Note that according to Table II, the instance of Figure 5(b) is an optimal solution of Problem 1, i.e., it minimizes Δ_2.



(a) after Phase 1 (b) after first iteration of Phase 2

Fig. 5: ADR example

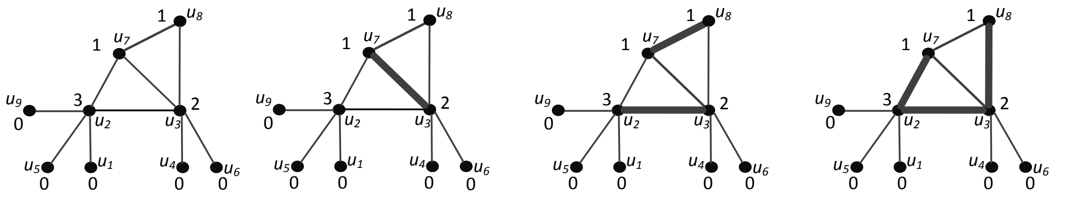
4.3. Approximate B-Matching (ABM)

This section presents ABM, which stands for Approximate B-Matching. We first present the motivation behind ABM, and then provide the algorithmic framework.

Motivation

Let an undirected graph $G = (V, E)$, and a set of capacity constraints $b(u)$: $V \rightarrow \mathbb{N}$. A subgraph $g = (V, E_g)$ of G is a *b-matching* of G, if the degree of each vertex $u \in V$ in g is at most $b(u)$. The term *b-matching* is used interchangeably to denote the subgraph g, or its edge set E_g, depending on the context. If $b(u) = 1$ for all vertices of G, then *b-matching* reduces to the well known matching problem in graph theory. A b-matching is *maximal*, if the addition of any edge violates at least one capacity constraint. A *maximum* b-matching is the maximal b-matching with the largest number of edges.

Figure 6(a) shows an example graph, where the capacity constraint $b(u_i)$ is shown next to u_i. Figure 6(b) illustrates a b-matched graph of Figure 6(a), where the matched edge (u_3, u_7) is shown in bold. Since there is no violation of any capacity constraint, it is a valid b-matching. Figure 6(c) depicts a maximal b-matching: adding any other edge (e.g., (u_2, u_7)) would violate the capacity constraint of at least one vertex (e.g., u_7). Finally, Figure 1(d) illustrates a maximum b-matching.



(a) G and constraints (b) b-matching of G (c) *maximal* b-matching (d) *maximum* b-matching

Fig. 6: b-matching example

Numerous exact and approximate solutions have been proposed for finding a maximum *b-matching* (see [Hougardy 2009] for a survey). If the capacity constraints are bounded by a constant, the fastest exact algorithm is $O(|E|^{3/2})$ [Micali and Vazirani 1980]. A greedy 1/2-approximation technique [Hougardy 2009] solves the problem in $O(|E|)$. Several methods aim at weighted versions of the problem [Mestre 2006].

We next investigate the relationship of Problem 1 to b-matching, starting with the special case where all the expected degrees of $\mathcal{G}$ are integers, i.e., $[deg(u,\mathcal{G})] \in \mathbb{Z}, \forall u \in V$. As we shall prove shortly, the *optimal instance* for Problem 1 (i.e., the one that minimizes the overall discrepancy Δ_2), is given by a maximum b-matching computed on the uncertain graph $\mathcal{G}$ with capacity constraints $[deg(u,\mathcal{G})]$ for each vertex $u \in V$.

LEMMA 4.2. *Assume that $[deg(u,\mathcal{G})] \in \mathbb{Z}, \forall u \in V$. Then, there is at least one optimal instance G_2^* for which $deg(u,G_2^*) \leq [deg(u,\mathcal{G})]$, for all $u \in V$.*

PROOF. Assume an optimal solution $G_2^* = (V, E^*)$ that contains *illegal* vertices, i.e., vertices $u \in V$ with $deg(u,G_2^*) > [deg(u,\mathcal{G})]$. E^* cannot contain an edge (u,v) between two illegal vertices u and v, otherwise G_2^* is not optimal (i.e., the exclusion of edge (u,v) would decrease the overall discrepancy Δ_2). Thus, an illegal vertex u can only be adjacent to *legal* vertices, i.e., vertices $x \in V$ for which $deg(x,G_2^*) \leq [deg(x,\mathcal{G})]$. Assume an edge $e = (u,x) \in E^*$, where u is illegal and x is legal. We first prove that if we remove edge e from E^*, then the remaining graph $G' = (V, E^* - \{e\})$ is also an optimal instance.

Specifically, $dis_2(u,G_2^*) > 0$ whereas $dis_2(x,G_2^*) \leq 0$, and $dis_2(u,G') = dis_2(u,G_2^*) - 1 \geq 0$ whereas $dis_2(x,G') = dis_2(x,G^*) - 1 < 0$. The overall discrepancy of G_2^* is:

$$\begin{aligned}\Delta_2(G_2^*) &= |dis_2(u,G_2^*)| + |dis_2(x,G_2^*)| + \sum_{v \neq \{u,x\} \in V} |dis_2(v,G_2^*)| = \\ &= dis_2(u,G_2^*) - dis_2(x,G_2^*) + \sum_{v \neq \{u,x\} \in V} |dis_2(v,G_2^*)|\end{aligned}$$

Similarly, the discrepancy of G' is:

$$\begin{aligned}\Delta_2(G') &= |dis_2(u,G')| + |dis_2(x,G')| + \sum_{v \neq \{u,x\} \in V} |dis_2(v,G')| = \\ &= (dis_2(u,G_2^*) - 1) + (1 - dis_2(x,G_2^*)) + \sum_{v \neq \{u,x\} \in V} |dis_2(v,G')|\end{aligned}$$

Since graphs G_2^* and G' only differ by the edge (u,x), $\sum\limits_{v \neq \{u,x\} \in V} |dis_2(v,G_2^*)| = \sum\limits_{v \neq \{u,x\} \in V} |dis_2(v,G')|$, and thus $\Delta_2(G_2^*) = \Delta_2(G')$. By applying the above argument to all the illegal vertices of G_2^*, we construct an optimal instance that contains only legal vertices. □

THEOREM 4.3. *Let $\mathcal{G} = (V,E,p)$ be an uncertain graph where $[deg_u] \in \mathbb{Z}, \forall u \in V$. An optimal solution of Problem 1 on input $\mathcal{G}$ is given by solving a maximum b-matching on graph $\mathcal{G}$ using $[deg(u,\mathcal{G})]$ as capacity constraint of vertex u.*

PROOF. Using the previous lemma, there is always an optimal solution $G_2^* = (V, E^*)$ that ensures that $deg(u,G_2^*) \leq [deg(u,\mathcal{G})]$, for all $u \in V$. Thus, we can remove the absolute values from the definition of Δ_2:

$$\begin{aligned}\Delta_2(G_2^*) &= \sum_{u \in V} \left| deg(u,G_2^*) - [deg(u,\mathcal{G})] \right| = \\ &= \sum_{u \in V} \left([deg(u,\mathcal{G})] - deg(u,G_2^*) \right) = \sum_{u \in V} [deg(u,\mathcal{G})] - \sum_{u \in V} deg(u,G_2^*)\end{aligned}$$

Since the expected degrees $[deg(u,\mathcal{G})]$ are fixed, this is equivalent to maximizing $\sum_{u \in V} deg(u,G_2^*)$, which in turn leads to the maximization of $|E^*|$. Therefore, G_2^* is a maximum b-matching on $\mathcal{G}$ with capacity constraints $[deg(u,\mathcal{G})]$. □

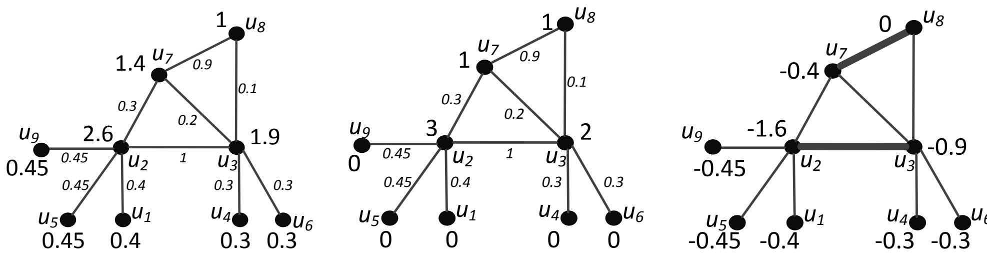
Algorithm

According to Theorem 4.3, a maximum b-matching on graph $\mathcal{G}$ leads to the optimal solution if *all* expected degrees are integers. However, since actual uncertain graphs have real valued expected degrees, the b-matching technique of the previous section cannot be applied directly. Instead, ABM involves two phases. Phase 1 rounds the expected vertex degrees to the closest integers, and computes a maximal b-matching using the rounded values as capacity constraints. Phase 2 partitions the vertices according to their discrepancy. Then, it extracts additional edges that improve the total discrepancy Δ_2, by performing a bipartite matching. We use approximation techniques for the two phases (i.e., b-matching and bipartite matching) for efficiency reasons.

Algorithm 3 contains the pseudocode of ABM. Phase 1 (lines 3-7) corresponds to a greedy approximate maximum b-matching [Hougardy 2009] that considers all edges in random order. For each edge $e = (u, v)$, if the capacity constraints of both vertices u and v are not violated, then e is inserted into the result set E_m, and the degrees of u and v are incremented. After all edges have been considered, E_m contains a maximal b-matching of $\mathcal{G}$, whose cardinality is at least half of that of the maximum [Hougardy 2009]. Figure 7(b) shows the vertex degrees after rounding on the uncertain graph of Figure 7(a). ABM considers in turn edges $(u_2, u_3), (u_7, u_8)$, which are added to E_m. After that, no other edge can be included in E_m because it would cause a capacity violation. Figure 7(c) includes the vertex discrepancies after the termination of Phase 1, with respect to their original (i.e., before rounding) degree.

Algorithm 3 Approximate b-Matching (ABM)

```
Input: uncertain graph $\mathcal{G} = (V, E, p)$
Output: representative $G^* = (V, E^*)$.
 1: calculate the expected degree $[deg(i, \mathcal{G})]$ for all vertices in $V$.
 2: $E_m \leftarrow 0, deg(u) \leftarrow 0$
                                                            // Phase 1
 3: let $b_i = \mathsf{round}([deg_i])$ to the closest integer
 4: for each $e = (u, v) \in E$ do
 5:     if $deg(u) < b_u$ AND $deg(v) < b_v$ then
 6:         $E_m \leftarrow E_m \cup \{e\}$
 7:         $deg(u) \leftarrow deg(u) + 1; deg(v) \leftarrow deg(v) + 1$
                                                            // Phase 2
 8: $A \leftarrow \emptyset, B \leftarrow \emptyset, C \leftarrow \emptyset$
 9: for each $u \in V$ do
10:     let $dis_2(u) = deg_2(u) - [deg_2(u, \mathcal{G})]$
11:     if $dis_2(u) \leq -0.5$ then $A \leftarrow A \cup \{u\}$
12:     else if $-0.5 < dis_2(u) < 0$ then $B \leftarrow B \cup \{u\}$
13:     else $C \leftarrow C \cup \{u\}$
14: $E' \leftarrow E \setminus E_m$
15: for each edge $e = (u, v) \in E'$ do
16:     weight= $|dis_2(u)| + 2|dis_2(v)| - |1 + dis_2(u)| - 1$
17:     if $(u \in A)$ AND $(v \in B)$ AND (weight> 0) then $w(e) \leftarrow$ weight
18:     else discard $e$
19: Let $G' = ((A \cup B), E', W)$ where $W : w(e) \rightarrow \mathbb{R}$
20: $E_{BP} = \mathsf{bipartite}(G')$
21: $E^* = E_m \cup E_{BP}$
```



(a) initial graph $\mathcal{G}$ (b) rounded graph (c) result of b-matching

Fig. 7: ABM Phase 1

Based on their discrepancies, Phase 2 partitions the vertices into three groups A, B and C. A contains vertices with discrepancy $dis_2(u) \leq -0.5$, B the vertices for which $-0.5 < dis_2(u) < 0$, and C vertices with $dis_2(u) \geq 0$. The partitioning is complete (i.e., $A \cup B \cup C = V$) and there is no overlap (i.e., $A \cap B \cap C = \emptyset$). In our running example the groups are $A = \{u_2, u_3\}$, $B = \{u_1, u_4, u_5, u_6, u_7, u_9\}$ and $C = \{u_8\}$.

Intuitively, A contains vertices, whose absolute discrepancy will decrease by the addition of an edge. B contains vertices whose absolute discrepancy will increase (after the addition of an edge) by less than 1. C contains vertices that have already reached or exceeded their expected degree[1]; thus, a new adjacent edge will increase their discrepancy by 1. Edges between vertices of A (e.g., (u_2, u_3)) have been added to the result set E_m during Phase 1. The following lemmas discuss the potential for including additional edges, depending on the group of their incident vertices.

LEMMA 4.4. *Let an edge (u, v) where $u, v \in B$. Including edge (u, v) in the result set cannot improve the overall discrepancy Δ_2.*

PROOF. Since both vertices belong to the set B, it holds that $-0.5 < dis_2(i) < 0$ for $i = \{u, v\}$. Thus, their total discrepancy is $d_1 = |dis_2(u)| + |dis_2(v)| < 1$. The addition of edge (u, v) will change the discrepancies to $dis'_2(u) = dis_2(u) + 1 > 0.5$, and $dis'_2(v) > 0.5$. Thus, the total discrepancy becomes $d_2 = |dis'_2(u)| + |dis'_2(v)| > 1$. Since $d_1 < d_2$, edge (u, v) increases Δ_2. $\square$

LEMMA 4.5. *Let an edge (u, v) where $u \in C$. Including edge (u, v) in the result cannot improve the overall discrepancy Δ_2.*

PROOF. Since u has exceeded its expected degree, adding edge (u, v) will increase $dis_2(u)$ by exactly 1. On the other hand, the absolute discrepancy of v can increase or decrease by at most 1. Thus, Δ_2 can only increase (if $v \in B$ or $v \in C$) or remain the same (if $v \in A$). $\square$

The above lemmas state that the inclusion in the result of an edge that connects i) two vertices in B or ii) a vertex in C with any vertex, cannot improve Δ_2. Therefore, after Phase 1, the only possible additions are edges that connect vertices in A with vertices in B. Let such an edge $e = (a, b)$ with $a \in A$ and $b \in B$: if the addition of e in the result set decreases the absolute discrepancy of a more than it increases the absolute discrepancy of b, then it improves Δ_2. The following lemma, quantifies this improvement.

LEMMA 4.6. *Let $e = (a, b)$ where $a \in A$ and $b \in B$. Including edge e in the result set changes the overall discrepancy Δ_2 by $g(e) = |dis_2(a)| + 2|dis_2(b)| - |dis_2(a) + 1| - 1$.*

[1]Such vertices are possible because b-matching is performed on the rounded degrees.

PROOF. The total discrepancy of vertices a and b *before* (a, b) is, $d_1 = |dis_2(a)| + |dis_2(b)|$. After adding the edge, it becomes $d_2 = |dis_2(a) + 1| + (1 - |dis_2(b)|)$. We define the *gain* $g(e)$ of e as the difference between the two discrepancies, i.e., $g(e) = d_1 - d_2 = (|dis_2(a)| + |dis_2(b)|) - \big(|dis_2(a) + 1| + (1 - |dis_2(b)|)\big) = |dis_2(a)| + 2|dis_2(b)| - |dis_2(a) + 1| - 1$. If $g(e)$ is positive, edge e decreases Δ_2; otherwise it increases it. If $dis_2(a) \leq -1$, then $|dis_2(a) + 1| = |dis_2(a)| - 1$, and the gain becomes $g(e) = 2|dis_2(b)|$, i.e., $g(e)$ depends only on $|dis_2(b)|$. $\square$

An edge e can improve the overall discrepancy, only if $g(e) > 0$. Lines 15-18 of Algorithm 3 consider each edge e in $E' = E - E_m$ connecting vertices in A and B. If $g(e) > 0$, e is inserted in a graph $G' = (A \cup B, E', W)$ where $W : E' \to \mathbb{R}$, with $w(e) = g(e)$. Figure 8(a) shows the graph G', including the vertex discrepancies and edge weights, which correspond to their gains. The next question is how to efficiently select the subset of edges in E' that minimizes Δ_2. Towards this, subroutine bipartite(G') performs an approximate maximum weight bipartite matching on graph G' with a twist: a vertex of A may be matched with multiple vertices of B, if it has high absolute discrepancy.

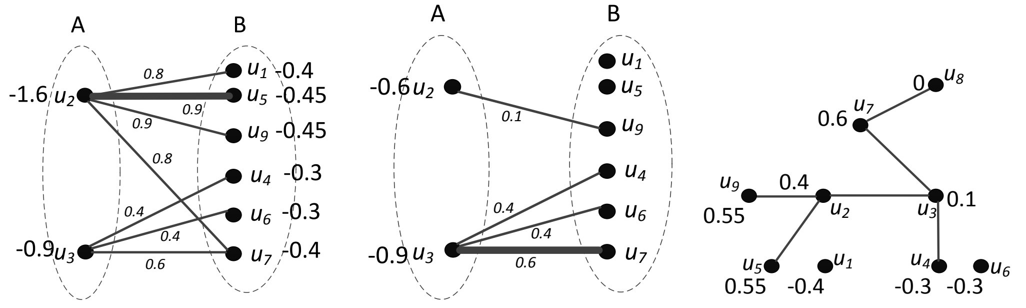
Algorithm 4 illustrates bipartite, which utilizes a max-heap H to arrange the edges $e \in E'$ based on their weights/gains $g(e)$. Initially, all edges of E' are inserted in H. At each iteration, the top of the heap $e = (a, b)$ is added to the result. The inclusion of (a, b) increases the discrepancies of a and b. Specifically, the new discrepancy of b becomes positive; thus, according to Lemma 4.5, edges adjacent to b cannot reduce Δ_2, and are removed from H. Then, bipartite updates the discrepancy of a; let $dis_2(a)$ be the new value. If $dis_2(a) \leq -1$, from Lemma 4.6, the gain $g(e')$ of every edge $e' = (a, x) \in H$ does not change since it depends only on $|dis_2(x)|$. If $-1 < dis_2(a) < -0.5$, then $g(e')$ decreases since $|dis_2(a)|$ has been decreased. bipartite computes the new gain $g(e')$ using Lemma 4.6 (line 10), and triggers a decrease-key(e') operation to relocate e' in H (line 12). If $g(e')$ becomes negative, the edge is removed. Finally, if $dis_2(a) > -0.5$, vertex a cannot further improve Δ_2 as it does not belong to group A anymore; thus, it is discarded and all adjacent edges (a, x) are expunged from H. The function terminates when H becomes empty.

Algorithm 4 bipartite

```
Input: bipartite graph $G' = (A \cup B, E', W)$
Output: set of edges $E_{BP} \subseteq E'$ with high gain for $\Delta_2$
1:  $E_{BP} \leftarrow \emptyset$
2:  Insert all edges $e \in E'$ in a max-heap $H$, based on their gains $g(e)$
3:  while $H$ is not empty do
4:      $e = (a, b) \leftarrow H$.extract-max()
5:      $E_{BP} \leftarrow E_{BP} \cup \{e\}$
6:      discard all edges in $H$ incident to $b$
7:      $dis_2(a) \leftarrow dis_2(a) + 1$        // The discrepancy of $a$ increases by one
8:      if $-1 < dis_2(a) < -0.5$ then
9:          for each edge $e' = (a, x) \in H$ do
10:             $g(e') \leftarrow |dis_2(a)| + 2|dis_2(x)| - |dis_2(a) + 1| - 1$
11:             if $g(e') > 0$ then $H$.decrease-key($e'$)
12:             else $H$.remove($e'$)
13:     else if $dis_2(a) > -0.5$ then
14:         for each edge $e' = (a, x) \in H$ do $H$.remove($e'$)
```

Continuing the example of Figure 8(a), bipartite first picks the heaviest edge (u_2, u_5) and adds it to the result. Then, it updates the discrepancy of u_2 to $dis_2(u_2) = -1.6 + 1 = -0.6$;

since $-1 < dis_2(u_2) < -0.5$, the gains of edges adjacent to u_2, $\big($i.e., $(u_2, u_1), (u_2, u_7)$ and $(u_2, u_9)\big)$ must be updated as well. Edges (u_2, u_1) and (u_2, u_7) yield a negative gain, and are discarded. Figure 8(b) shows the bipartite graph after the first iteration. The next edge (u_3, u_7) extracted from H is included in the result, updating the discrepancy of vertex u_3 to $dis_2(u_3) = -0.9 + 1 = 0.1$. Since $dis_2(u_3) > 0$, all edges adjacent to u_3 are expunged from H. Finally, bipartite extracts the last edge (u_2, u_9) of H, adds it to the result and terminates. Figure 8(c) shows the final output of ABM, which combines the edges added during the two phases. The discrepancy of the extracted graph is 3.2. The total cost of ABM includes the linear-time processing of edges in Phase 1, and the heap operations of Phase 2 on E'. Each edge of E' can be processed at most $|B|$ times. Therefore, the complexity of ABM is $O(|E| + |B| \cdot |E'| \cdot \log |E'|)$, where $|E'| \leq |E|$.



(a) input of bipartite (b) after first iteration (c) G^* of ABM

Fig. 8: ABM Phase 2

5. ALGORITHMS FOR MINIMIZING NEIGHBORHOOD CONNECTIVITY DISCREPANCY

Recall that ADR and ABM aim explicitly at capturing the expected degree of the vertices. Specifically, ADR requires an instance that preserves the average n-clique cardinality as a seed. This is an easy task for $n = 2$ as each edge affects exactly two vertices i.e., its endpoints. For $n > 2$, each edge potentially affects all other vertices of the graph, hence ADR cannot be applied. Accordingly, ABM exploits some properties of b-matching, which are specific to vertex degrees. On the other hand, GREEDY and GAME generate representative instances minimizing the objective function $\sum_{m=l}^{n} \Delta_m$ of Problem 2, for larger values of n and/or l. Section 5.1 introduces basic procedures that are used by both GREEDY and GAME frameworks, which are presented in sections 5.2 and 5.3, respectively.

5.1. Basic procedures

Let E^* be the set of edges in the current representative G^*, stored in the form of adjacency lists. Given an edge $e = (u, v)$ and an integer $m \geq 2$, $Q_m(e, E^*)$ denotes the set of m-cliques that contain both endpoints u and v in E^*. $\mathcal{L}_m(e)$ is the set of vertices that belong to at least one clique of $Q_m(e, E^*)$, i.e. $\mathcal{L}_m(e) = \bigcup_{c \in Q_m(e,E^*)} c$. The union of all these sets $\mathcal{L}(e) = \bigcup_{m=l}^{n} \mathcal{L}_m$ contains the *affected vertices* of edge e. The following lemma derives the cardinality of $Q_m(e, E^*)$.

LEMMA 5.1. *Given an edge $e = (u, v)$, the number of m-cliques that contain both endpoints u and v is* $|Q_m(e, E^*)| = O(\frac{|V|^{m-2}}{(m-2)!})$.

Proof. In the worst case G^* is a complete graph of $|V|$ vertices. In order to generate all m-cliques that contain u and v we have to choose $m-2$ vertices out of the $|V|-2$ vertices in $V \setminus \{u, v\}$. Thus,

$$|Q_m(e, E^*)| = \binom{|V|-2}{m-2} = \frac{(|V|-2)\cdot(|V|-3)\cdots(|V|-m)}{(m-2)\cdot(m-3)\cdots 1} = O\Big(\frac{|V|^{m-2}}{(m-2)!}\Big)$$

□

Algorithm 5 presents a recursive subroutine, which outputs $Q_m(e, E^*)$. The base of the recursion is $m = 2$, where the result $Q_2(e, E^*)$ contains a single clique $\{u, v\}$. For $m > 2$, each clique c of size $m-1$ is extended by the addition of a vertex w that is a common neighbor of all vertices in c, i.e., $w \in \bigcap_{i \in c} adj(i)$. The computation is performed by intersecting the adjacency lists of all elements in c. For instance, if $m = 3$, for each element w in $adj(v) \cap adj(u)$, a clique $\{u, v, w\}$ is added to the set $Q_3(e, E)$.

Algorithm 5 Q_m

```
Input: edge $e = (u, v)$, set of edges $E^*$
Output: set $Q_m(e, E^*)$ of cliques of size $m$ in $E^*$ that contain both endpoints of $e$
1: if $m = 2$ then
2:     $Q_2(e, E^*) \leftarrow \{\{u, v\}\}$
3: else
4:     $Q_m(e, E^*) \leftarrow \emptyset$
5:     for each clique $c \in Q_{m-1}(e, E^*)$ do
6:         $W \leftarrow \bigcap_{i \in c} adj(i)$
7:         for each vertex $w \in W$ do
8:             $Q_m(e, E^*) \leftarrow Q_m(e, E^*) \cup \{c \cup \{w\}\}$
```

To analyze Algorithm 5, we focus on the last step; assuming that we have computed the set Q_{n-1}, we wish to generate Q_n. According to Lemma 5.1, $|Q_{n-1}(e, E^*)| = O\Big(\frac{|V|^{n-3}}{(n-3)!}\Big)$. For each clique $c \in Q_{n-1}$, line 6 finds the set W of vertices that appear in the adjacency lists of all $n-1$ nodes of c. This intersection can be performed in linear time to the size of the lists, using an array of size $|V|$ that counts the number of occurrences of each vertex in the lists (the intersection consists of vertices, whose counter equals $n-1$). Since, there are $n-1$ lists, each with size less than $|V|$, the cost per clique is upper bounded by $(n-1)\cdot|V|$. Repeating the process for all cliques in Q_{n-1} yields cost:

$$O\big((n-1)\cdot|V|\cdot\frac{|V|^{n-3}}{(n-3)!}\big) = O\big((\frac{1}{(n-4)!} + \frac{2}{(n-3)!})\cdot|V|^{n-2}\big) = O(|V|^{n-2})$$

By applying the same reasoning, we derive the complexity of each step; e.g., generating cliques of size $n-1$ from Q_{n-2} costs $O(|V|^{n-3})$. Thus, the total complexity of Algorithm 5 is described by a geometric series, which is dominated by the largest term, i.e., $O(|V|^{n-2})$.

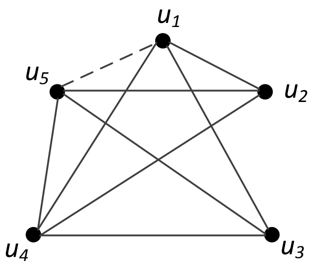
The inclusion or removal of an edge e, alters the n-clique cardinalities of affected vertices and the *gains* of their incident edges, i.e., the benefit of adding those edges in E^*. Algorithm 6 illustrates the function update-vertices(e, E^*) that outputs the set $\mathcal{L}(e)$ of vertices affected by e. In addition, for each vertex $w \in \mathcal{L}(e)$ and each $m \in [l, n]$, it updates the m-clique cardinality $\gamma_m(w)$ and counts the occurrences $k_m(w)$ of w in $Q_m(e, E^*)$. Line 3 checks whether e belongs to E^* and sets the value of $flag$ accordingly. Specifically, $flag$ is set to $1(-1)$, if e is inserted to (removed from) E^*. Then, for each clique cardinality m, lines 4-10 compute the set of cliques $Q_m(e, E^*)$ that contain both u and v in E^*, by calling Algorithm 5. The m-clique cardinality γ_m of each node of every clique in $Q_m(e, E^*)$ is incremented (decremented) depending on the value of $flag$. When the algorithm terminates, the value of $k_m(w)$ equals the absolute difference between the new and previous value of $\gamma_m(w)$.

Algorithm 6 update-vertices

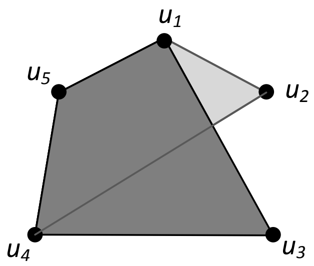
```
Input: edge $e = (u, v)$, set of edges $E^*$
Output: set $\mathcal{L}(e)$ of affected vertices
1: $\mathcal{L}(e) \leftarrow \emptyset$
2: $k_m(w) \leftarrow 0, \forall w \in V$ and $m \in [l, n]$
3: if $e \in E^*$ then $flag = -1$ else $flag = 1$
4: for $m \leftarrow l..n$ do
5:     $Q_m(e, E^*)$                                   // Algorithm 5
6:     for each clique $c \in Q_m(e, E^*)$ do
7:         for each node $w \in c$ do
8:             $\gamma_m(w) \leftarrow \gamma_m(w) + flag$
9:             $\mathcal{L}(e) \leftarrow \mathcal{L}(e) \cup \{w\}$
10:            $k_m(w) \leftarrow k_m(w) + 1$
```

The complexity of update-vertices is analyzed as follows. For each value of m, the set of cliques $Q_m(e, E^*)$ is computed in $O(|V|^{m-2})$ time. According to Lemma 5.1, the loop of line 6 needs to be executed $O(\frac{|V|^{m-2}}{(m-2)!})$ times, while the inner loop of line 7 needs to be executed m times, as each clique has m vertices. Thus, the time complexity of lines 6-10 is $O(|V|^{m-2})$. The overall complexity of Algorithm 6 is $O(|V|^{n-2})$, dominated by the greatest value of $m = n$.

As an example of update-vertices, consider the addition of edge $e = (u_1, u_5)$ in the representative graph of Figure 9(a). If $m = 3$, the 3-clique cardinality γ_3 of vertices u_2, u_3 and u_4 that appear in both $adj(u_1)$ and $adj(u_5)$ increases by one, while $\gamma_3(u_1)$ and $\gamma_3(u_5)$ increase by three due to the creation of triangles $\{u_1, u_5, u_2\}$, $\{u_1, u_5, u_3\}$, $\{u_1, u_5, u_4\}$. In general, $\gamma_m(u)$ and $\gamma_m(v)$ are updated $|Q_m(e, E^*)|$ times since u and v belong to all cliques in $Q_m(e, E^*)$. Observe that, while for $m = 3$ a vertex $w \in Q_3(e, E^*) \setminus \{u, v\}$ is updated exactly once i.e., $k_3(w) = 1$, for $m > 3$, its m-clique cardinality may increase by $k_m(w) > 1$. For instance, if $m = 4$, the addition of edge $e = (u_1, u_5)$ yields two 4-cliques $Q_4(e, E^*) = \{\{u_1, u_2, u_4, u_5\}, \{u_1, u_3, u_4, u_5\}\}$ as shown in Figure 9(b). Figure 9(c) illustrates $\gamma_4(u)$ of all vertices before and after the insertion, as well as their difference $k_4(u)$.



(a) addition of (u_1, u_5)



(b) 4-cliques

	before adding (u_1,u_5)	after adding (u_1,u_5)	$k_4(u) =$
u	$\gamma_4(u)$	$\gamma'_4(u)$	$\gamma'_4(u) - \gamma_4(u)$
u_1	0	2	2
u_2	0	1	1
u_3	0	1	1
u_4	0	2	2
u_5	0	2	2

(c) $\gamma_4(u)$ before and after

Fig. 9: Example for $m = 4$

Given an edge $e = (u, v)$ and the current set E^*, compute-gain returns the gain $g(e)$ of adding (removing) e to (from) E^*. The gain is given by the following formula:

$$g(e) = \sum_{m=l}^{n} \sum_{w \in \mathcal{L}_m(e)} \big(|dis_m(w)| - |dis_m(w) + flag \cdot k_m(w)|\big) \quad (5)$$

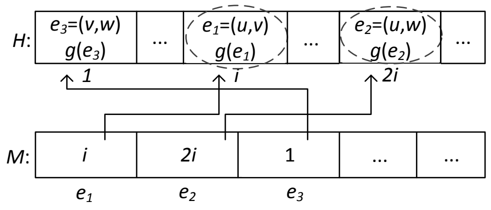
where, $\mathcal{L}_m(e)$ is the set of vertices in the same m-clique as $\{u, v\}$, $k_m(w)$ is the difference in the m-clique cardinality $\gamma_m(w)$ incurred by e and $flag$ is $+1$ or -1 for insertion or deletion, respectively. Intuitively, the gain corresponds to the sum of discrepancy differences of the affected vertices caused by the addition / removal of e. The pseudocode for compute-gain is similar to Algorithm 6 and omitted. The time complexity is also identical ($O(|V|^{n-2})$) since gain computation involves the generation of Q_n.

5.2. Greedy Framework

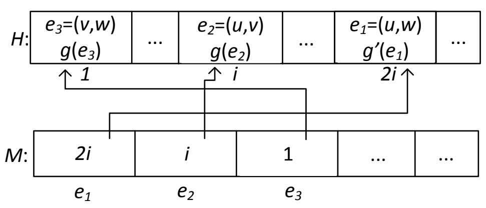
GREEDY extends the concept of PS to Problem 2 using a max heap to dynamically rearrange the edges according to their gain for the current representative G^* (instead of the fixed order, based on probabilities, used by PS). Specifically, the key in the heap H is the gain computed by Equation 5; if two edges have the same gain, then the one with the higher probability precedes the other. At the beginning, all edges are added to H. Note that if $m > 2$, the initial gains of all edges are 0 because the inclusion of any edge in $E^* = \emptyset$ cannot create any m-clique. In this case, the edges are inserted in H based on their probabilities. If $m = 2$, all gains are positive and most of them equal 2, except for edges that are incident to at least a vertex with expected degree less than 1.

At each iteration the top of the heap e (i.e., the edge with the maximum gain) is extracted and, if it has non-negative gain[2], it is added to E^*. Then, the gains of the affected edges are updated and the heap entries are rearranged. The process terminates when the next extracted edge has negative gain, or the heap is empty. The challenge lies in i) efficiently updating the edges whose gain is affected by the inclusion of e and ii) maintaining the heap property of the max-heap.

Figure 10(a) contains the data structures that facilitate the execution of GREEDY. The max-heap H is implemented as an array of initially $|E|$ elements, such that the children of an element i are located at positions $2 \cdot i$ and $2 \cdot i + 1$. Each element contains an edge $e = (u, v)$ and its corresponding gain $g(e)$. An array of pointers M keeps track of the location of edges in H. Every time an update occurs to an edge e in H, increase-key(e) or decrease-key(e) relocate e to ensure that H maintains the heap property. Accordingly, M changes the relevant pointer to the new location of e. For example, assume that in Figure 10(b) an update reduces the gain of e_1 to a value smaller than that of its child e_2. Then, decrease-key(e_1) is triggered and e_1 swaps places with e_2. M records the update by swapping the pointers of e_1 and e_2.



(a) Before update



(b) After decrease-key(e_1)

Fig. 10: Structures utilized by GREEDY

[2]Edges with gain 0 are also accepted because, if $l > 2$, initially every edge has gain 0.

Algorithm 7 illustrates the pseudocode of GREEDY. Lines 1-3 initialize the data structures and set $\gamma_m(u) = 0$ for all $u \in V$ and $m \in [l, n]$. For every edge e, lines 4-6 first compute e's gain $g(e)$ by invoking compute-gain, and then insert e in H according to $g(e)$. At each iteration in lines 7-14, the edge e at the top of the heap, and its gain $g(e)$, are retrieved. If $g(e) \geq 0$, the addition of e to the result improves the overall discrepancy. Consequently, Algorithm 6 returns the affected vertices $\mathcal{L}(e)$ and computes their m-clique cardinality. Given $\mathcal{L}(e)$, update-heap is invoked in order to update structures H and M. Algorithm 7 terminates when either all edges have been included, or the extracted edge has negative gain, i.e., its inclusion to the result worsens $\sum_{m=l}^{n} \Delta_m$.

Algorithm 7 GREEDY

```
Input: the set of vertices $V$, set of edges $E$, clique cardinality range $[l, n]$
Output: representative $G^* = (V, E^*)$
1: $E^* \leftarrow \emptyset$
2: $H \leftarrow$ empty max-heap of size $|E|$
3: $\gamma_m(v) \leftarrow 0 \; \forall v \in V$ and $m \in [l, n]$
4: for each edge $e = (u, v) \in E$ do
5:     $g(e) \leftarrow$compute-gain $(e, E^*)$
6:     $H$.insert($e$) using $g(e)$ as the key. Break ties using edge probability $p_e$.
7: repeat
8:     $e \leftarrow H$.extract-max()
9:     $g(e) \leftarrow H$.get-key($e$)
10:    if $g(e) \geq 0$ then
11:        $E^* \leftarrow E^* \cup \{e\}$
12:        $\mathcal{L}(e) \leftarrow$ update-vertices($e, E^*$)                // Algorithm 6
13:        update-heap($e, \mathcal{L}(e), E^*$)
14: until $g(e) < 0$ OR $H$.isEmpty()
```

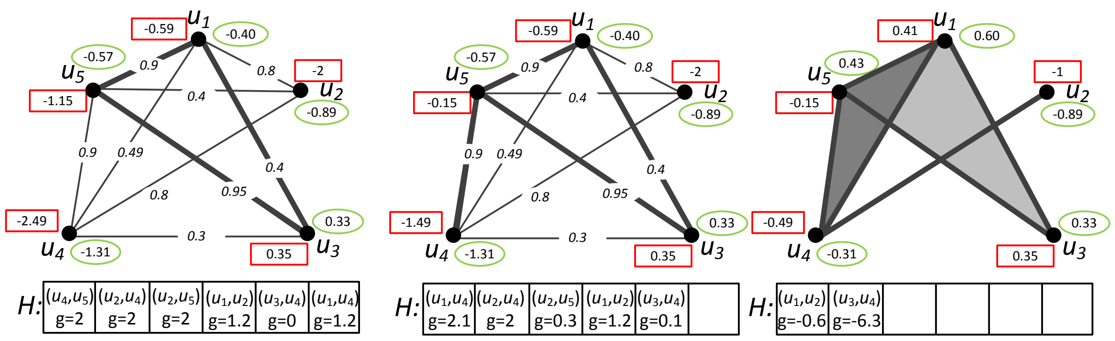
Algorithm 8 shows the pseudocode of update-heap that rearranges the elements of H after the insertion of e to E^*. For each affected vertex $w \in \mathcal{L}(e)$, update-heap retrieves all incident edges $e' = (w, x)$ that have not been added to the result yet, i.e., $e' \in E \setminus E^*$. For each such edge, the new gain $g'(e')$ is computed and then compared to its previous value $g(e')$. The index M facilitates the efficient retrieval of $g(e')$. Depending on $g'(e')$, the corresponding heap operation relocates e_i within H: a decrease triggers decrease-key(e'), while a gain increase triggers increase-key(e').

Algorithm 8 update-heap

```
Input: edge $e = (u, v)$, affected vertices $\mathcal{L}(e)$, set of edges $E^*$
Output: updates $H$ after insertion of $e$
1: for each edge $e' = (w, x) \in E \setminus E^*$ incident to at least one vertex $w \in \mathcal{L}(e)$ do
2:     $g(e') \leftarrow H$.get-key($e'$)
3:     $g'(e') \leftarrow$ compute-gain $(e', E^*)$
4:     if $g'(e') < g(e')$ then $H$.decrease-key($e'$)
5:     else $H$.increase-key($e'$)
```

The complexity of update-heap is dominated by compute-gain, which is $O(|V|^{n-2})$, as opposed to $O(\log |E|)$ for the heap operations. In the worst case, the number of affected edges is $|E|$, incurring cost $O(|E| \cdot |V|^{n-2})$ per call of Algorithm 8. Since all edges may be extracted from the heap, invoking update-heap, the overall complexity of GREEDY is $O(|E|^2 \cdot |V|^{n-2})$.

Figure 11 demonstrates an example of GREEDY for $l = 2$ and $n = 3$, based on the uncertain graph of Figure 2(a), where the expected m-clique cardinalities of the vertices have been replaced by the corresponding 2-(rectangles) and 3-(ellipses) discrepancies. Each subfigure contains the current representative G^* (with bold lines) and the contents of H. Initially, all edges have gain 2 as the gain is non zero only for clique cardinality $m = 2$, and ties are resolved by edge probabilities. Figure 11(a) illustrates G^* and H after the three first iterations. At the fourth iteration, the head of H, edge $e = (u_4, u_5)$ with gain $g(e) = 2$ is extracted. Since the gain is non negative, e is added to the result set. Then, update-vertices sets $\gamma_2(u_4) = 1$ and $\gamma_2(u_5) = 3$ and returns the set of affected vertices $\mathcal{L}(e) = \{u_4, u_5\}$. Next, update-heap computes the gains of edges in H incident to vertices in $\mathcal{L}(e)$, i.e., (u_1, u_4), (u_2, u_4), (u_3, u_4) and (u_2, u_5). For example, compute-gain calculates $g'(u_1, u_4)$ as follows. Initially, $g'(u_1, u_4) = 0$. Then, for $m = 2$, the only 2-clique containing u_1 and u_4 is $\{u_1, u_4\}$, thus $g'(u_1, u_4) = 1 + 0.18 = 1.18$. For $m = 3$ the only 3-clique that contains u_1 and u_4 is $\{u_1, u_4, u_5\}$. Thus $g'(u_1, u_4) = 1.18 - 0.2 + 1 + 0.14 = 2.12$.



u1
u2
u3
u4
u5
-0.59
-0.40
-0.57
-1.15
-2
-0.89
-2.49
-1.31
0.33
0.35
0.9
0.8
0.4
0.49
0.95
0.3
H: (u4,u5) g=2 | (u2,u4) g=2 | (u2,u5) g=2 | (u1,u2) g=1.2 | (u3,u4) g=0 | (u1,u4) g=1.2
-0.15
-1.49
H: (u1,u4) g=2.1 | (u2,u4) g=2 | (u2,u5) g=0.3 | (u1,u2) g=1.2 | (u3,u4) g=0.1
0.41
0.60
0.43
-1
-0.49
-0.31
H: (u1,u2) g=-0.6 | (u3,u4) g=-6.3

(a) after three iterations (b) after fourth iteration (c) result of GREEDY

Fig. 11: GREEDY example (representative G^* and heap H) for $l = 2$ and $n = 3$

Figure 11(b) shows the snapshot after the fourth iteration. Note that the elements of the heap have been rearranged to maintain the heap property after the gain updates. For instance, edge (u_1, u_4), which was at the bottom of the heap before the inclusion of (u_4, u_5), is now at the top with gain $g(u_1, u_4) = 2.1$. The iterative process continues for two more rounds to include edges (u_1, u_4) and (u_2, u_4). Then, the algorithm terminates as the edge (u_2, u_5) extracted from the top of H has negative gain $g = -0.3$. Figure 11(c) contains the result of GREEDY and the contents of H. The resulting representative has sub-optimal discrepancy $\Delta_2 + \Delta_3 = 5.0$ (recall from Figure 3 that the optimal instance has overall discrepancy $\Delta_2 + \Delta_3 = 3.83$).

5.3. Game Theoretic Framework (GAME)

In GREEDY once an edge is added to the representative, it will not be subsequently removed. GAME allows for corrections during run-time by enabling the removal of edges from E^*. In our game, the players are the edges of the uncertain graph, which decide whether to participate in the representative. The decision / strategy s_e of each edge e is binary (1 corresponds to participation), and minimizes its own cost function $C_e(S)$, given the strategy vector of all players $S = (s_e : e \in E)$. For Problem 2, this cost function is:

$$C_e(S) = \sum_{m=l}^{n} \Big(\sum_{w \in \mathcal{L}_m(e)} |dis_m(w)| \Big) \tag{6}$$

Intuitively, e participates in the representative, if it improves the sum of the discrepancies of its *affected* vertices; or equivalently, using the terminology of the previous section, if e has a non-negative gain given the other edges in E^*. Conversely, e is removed from the representative, if its elimination also yields a non-negative gain.

Algorithm 9 describes the application of *best response dynamics* [Monderer and Shapley 1996] in our context. Initially, a seed representative G^* is created by assigning a random strategy $s_e = \{0, 1\}$ to every edge e. $\mathcal{L}$ denotes the set of affected vertices, and is initialized to the entire vertex set V. Then, the algorithm proceeds in rounds. In every round (lines 5-13), for each edge e incident to a vertex in $\mathcal{L}$, compute-gain returns the gain of e (as derived by Equation 5) and the set $\mathcal{L}(e)$ of vertices affected by e. If $e \notin E^*$ and e has a non negative gain, then e switches strategy from 0 to 1, and is included to E^*. On the other hand, if $e \in E^*$ and its removal yields non negative gain, then it switches strategy from 1 to 0. In both cases, the vertices of $\mathcal{L}(e)$ are added to the set $\mathcal{L}_{new}$. Only edges incident to vertices in $\mathcal{L}_{new}$ will be considered during the next round, as the gain of the rest remains unchanged.

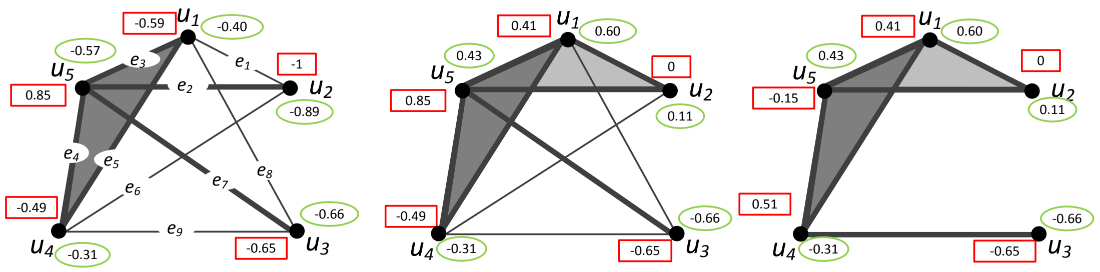
Algorithm 9 game theoretic framework (GAME)

```
Input: uncertain graph $\mathcal{G} = (V, E, p)$, clique cardinality range $[l, n]$
Output: representative $G^* = (V, E^*)$
1: create an initial representative $G^* = (V, E^*)$ by assigning a seed strategy to each edge
2: $\mathcal{L}_{new} \leftarrow V$
3: repeat
4:     $\mathcal{L} \leftarrow \mathcal{L}_{new}$; $\mathcal{L}_{new} \leftarrow \emptyset$
5:     for each edge $e$ incident to a vertex in $\mathcal{L}$ do
6:         $g(e) \leftarrow$ compute-gain $(e, E^*)$
7:         if $e \notin E^*$ and $g(e) \geq 0$ then
8:             $E^* \leftarrow E^* \cup \{e\}$                          // add edge to $G^*$
9:             $\mathcal{L}_{new} \leftarrow \mathcal{L}_{new} \cup \mathcal{L}(e)$
10:        if $e \in E^*$ and $g(e) \geq 0$ then
11:            $E^* \leftarrow E^* \setminus \{e\}$                     // remove $e$ from $G^*$
12:            $\mathcal{L}_{new} \leftarrow \mathcal{L}_{new} \cup \mathcal{L}(e)$
13: until Termination
```

Figure 12 contains an example of GAME for $l = 2, n = 3$, based on the uncertain graph of Figure 2(a). Next to each vertex, we denote its 2- and 3- discrepancy with a rectangle and ellipse, respectively. Figure 12(a) illustrates the initial representative of the GAME framework. Bold (resp. thin) lines indicate that edges participate (resp. do not to participate) in the seed graph; accordingly, the initial strategy vector is $S = (0, 1, 1, 1, 1, 0, 1, 0, 0)$ for the corresponding edges $e_1, e_2, \cdots, e_9$. Assume that at round 1, edge $e_1 = \{u_1, u_2\}$, which is not currently in E^*, is considered first. The set $\mathcal{L}(e_1)$ consists of u_1, u_2 and u_5. Specifically, the vertices whose 2- and 3- discrepancies are affected by e_1 are $\mathcal{L}_2(e_1) = \{u_1, u_2\}$ and $\mathcal{L}_3(e_1) = \{u_1, u_2, u_5\}$ (observe that the inclusion of e_1 in E^* would create a new triangle with u_5).

If e_1 retains its strategy, then its cost according to Equation 6 is $C_{e_1}(S) = \sum_{w \in \mathcal{L}_2(e_1)} |dis_2(w)| + \sum_{w \in \mathcal{L}_3(e_1)} |dis_3(w)| = (0.59 + 1) + (0.40 + 0.89 + 0.57) = 3.45$. On the other hand, if e_1 decides to participate, i.e., the strategy vector changes to $S' = (1, 1, 1, 1, 1, 0, 1, 0, 0)$, then its cost becomes $C_{e_1}(S') = (0.41{+}0){+}(0.60{+}0.11{+}0.43) = 1.55$. Since $C_{e_1}(S') < C_{e_1}(S)$ (i.e., $g(e_1) > 0$), e_1 switches strategy, yielding the representative and discrepancies of Figure 12(b). The vertices u_1, u_2, u_5 of $\mathcal{L}(e_1)$ are added to $\mathcal{L}_{new}$ because their discrepancies have been updated, influencing the gains of their incident edges. The procedure continues for the remaining edges, possibly for several rounds.

As we will discuss shortly, it always reaches a Nash equilibrium, where no edge can further improve its cost function. Figure 12(c) illustrates such an equilibrium, with strategy vector $S_{NE} = (1, 1, 1, 1, 1, 0, 0, 0, 1)$ and overall discrepancy $\Delta_2 + \Delta_3 = 3.83$. Although in this case the solution is the optimal representative, in general an equilibrium only corresponds to a local minimum.



(a) initial seed (b) after first decision (e_1) (c) Nash equilibrium

Fig. 12: GAME example for $l = 2$ and $n = 3$

Let *#rounds* be the number of rounds performed by GAME. In the worst case, at every round, $|E|$ edges change their strategy, invoking compute-gain, which has cost $O(|V|^{n-2})$. Thus, the overall complexity of GAME is $O(\#rounds \cdot |E| \cdot |V|^{n-2})$. Similar to ADR, *#rounds* can be used to adjust the trade-off between solution quality and efficiency. However, unlike ADR, where this parameter is necessary as the algorithm may never terminate, as shown in the following lemma, GAME always converges to a pure Nash equilibrium, eliminating the need for explicitly providing *#rounds*.

LEMMA 5.2. *GAME constitutes an exact potential game with potential function* $\Phi_{l,n}(G) = \sum_{m=l}^{n} \Delta_m(G)$.

PROOF. Let $\overline{S_e}$ be the strategy vector of all edges excluding e, i.e., $\overline{S_e} = (s_1, \cdots, s_{e-1}, s_{e+1}, \cdots, s_{|E|})$. Without loss of generality, assume that edge e changes its strategy from $s_e = 0$ to $s'_e = 1$, i.e., e decides to participate in the representative. Accordingly, let G^* and G' be the two representative instances, before and after the inclusion of e. The overall gain in the objective function $\Phi_{l,n}(G^*) = \sum_{m=l}^{n} \Delta_m(G^*)$ due to the addition of edge e is given by Equation 5. Specifically:

$$\Phi_{l,n}(G^*) - \Phi_{l,n}(G') = g(e) = \sum_{m=l}^{n} \sum_{w \in \mathcal{L}_m(e)} \Big(|dis_m(w)| - |dis_m(w) + k_m(w)|\Big)$$

The corresponding change in the individual cost function of e is,

$$C_e(s_e, \overline{S_e}) - C_e(s'_e, \overline{S_e}) = \sum_{m=l}^{n} \Big(\sum_{w \in \mathcal{L}_m(e)} |dis_m(w)| \Big) - \sum_{m=l}^{n} \Big(\sum_{w \in \mathcal{L}_m(e)} |dis_m(w) + k_m(i)| \Big) =$$
$$= \Phi_{l,n}(G^*) - \Phi_{l,n}(G') \quad \square$$

Intuitively, Lemma 5.2 proves that the gain in cost induced by the individual decision of any edge, equals the gain in the overall discrepancy. The equality $\Phi_{l,n}(G^*) - \Phi_{l,n}(G') = C_e(s_e, \overline{S_e}) - C_e(s'_e, \overline{S_e})$ holds independently of the choice of parameters l and n. Due to this property, GAME constitutes an *exact potential game*, guaranteeing that Algorithm 9 always terminates, reaching a Nash equilibrium [Monderer and Shapley 1996].

6. EXPERIMENTS

In this section, we assess the quality and efficiency of the proposed techniques. The evaluated methods are PS (*Probability Sorting*), ADR (*Average Degree Rewiring*), ABM (*Approximate B-Matching*), $\mathsf{GR}_{l,n}$ (the *greedy* approach) and $\mathsf{GM}_{l,n}$ (the *game* theoretic framework), where $2 \leq l \leq n \leq 3$. When $l = n$ we use the notation GR_n and GM_n respectively. For the heuristic methods, ADR and GAME, we allow a sufficient number of rounds so that they both converge. MP (the *Most Probable* instance) is excluded from our evaluation because, as shown in [Parchas et al. 2014], it has consistently poor performance in all evaluation metrics.

Section 6.1 describes the datasets used in the experiments. Section 6.2 presents the results on *structural measures* i.e., metrics for which the expected value (i.e., the ground truth) can be computed analytically. Section 6.3 focuses on *query metrics*, for which the ground truth is obtained through Monte Carlo simulations. Finally, Section 6.4 compares the running times of the algorithms.

6.1. Datasets

We use four uncertain, undirected graphs that capture different real-world scenarios and data characteristics e.g., size, density, edge probability. The details of the datasets are summarized in Table III.

Table III: Characteristics of real datasets

dataset	**vertices** ($\|V\|$)	**edges** ($\|E\|$)	$\|E\|/\|V\|$	**edge probabilities (mean):**	**exp. degrees (mean):**	**exp. triangles (mean):**
Flickr	78 322	10 171 509	129.89	0.09	22.93	164.50
Twitter	26 362	663 766	25.17	0.15	7.71	4.98
DBLP	9 442	144 887	15.34	0.29	9.04	18.77
BioMine	1 008 201	6 742 939	6.68	0.27	3.59	2.44

Flickr [Potamias et al. 2010](www.flickr.com): a social network, where the probability of an edge between two users is computed assuming *homophily*, i.e., the principle that similar interests indicate social ties [McPherson et al. 2001]. Homophily is measured by the Jaccard coefficient [Tan et al. 2005] of the interest groups of the two users. This is the densest dataset in our evaluation, as a vertex has on average the largest number of neighbors (i.e., about 130). The expected average degree (triangles) per node is 22.93 (164.50).

Twitter [Bonchi et al. 2014] (twitter.com): a network extracted from the popular online micro-blogging service and used in [Bonchi et al. 2014] in the context of influence maximization. Edge probabilities are learned from the log of past URL propagations and correspond to the influence that any two users exert on each other. Compared to the other datasets, Twitter has medium density, with expected average degree 7.71 and 4.98 triangles per node.

DBLP [Potamias et al. 2010; Jin et al. 2011b] (www.informatik.uni-trier.de/~ley/db/): a database of scientific publications and their authors. Two authors are connected, if they have coauthored a publication. The probability of an edge is derived from an exponential function to the number of collaborations and indicates the likelihood that the authors will collaborate again in the future. Similar to Twitter, DBLP has medium density; the difference is that each node has a smaller number of adjacent edges with, however, higher probabilities.

BioMine [Sevon et al. 2006] (biomine.org): a snapshot of the database of the BioMine project containing biological interactions. The probability of any edge corresponds to the confidence that the interaction actually exists. Particularly, this confidence is quantified by the *genomic-context* method, which measures interaction based on how much proteins are encoded by genes that share similar selection pressures [von Mering et al. 2003]. BioMine is the sparsest dataset with expected average degree 3.39, and 2.44 triangles per node.

Figure 13 illustrates the number of edges in the representatives PS, ABM, ADR, GR_2, GR_3, $\mathsf{GR}_{2,3}$, GM_2, GM_3, and $\mathsf{GM}_{2,3}$. The highest edge cardinality occurs in BioMine; although the uncertain graph of Flickr has more edges (10M as opposed to 6.7M), the mean edge probability is larger in BioMine (0.09 versus 0.29). Observe that for the same dataset, representatives focusing explicitly on Δ_3 are much denser than the rest. For instance, in Twitter, GR_3 and GM_3 have about 300K edges, whereas all the other representatives have about 100K. The implication is that by trying to minimize Δ_3, GR_3 and GM_3 may fail to capture Δ_2.

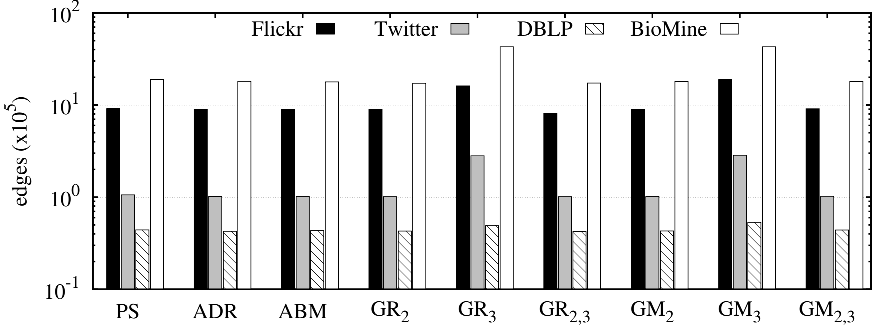


Fig. 13: Number of edges per representative

6.2. Structural measures

We assess the accuracy of the methods on the following measures:

- $dis_2(u)$ *discrepancy* corresponds to the absolute error between the degree $\gamma_2(u)$ of node u in the representative and its expected degree $[\gamma_2(u)]$ in the uncertain graph (i.e., $dis_2(u) = \gamma_2(u) - [\gamma_2(u)]$). The minimization of the sum Δ_2 of discrepancies over all nodes is the objective of Problem 1.
- $dis_3(u)$ *discrepancy* corresponds to the *triangle discrepancy*, i.e., the absolute error between the triangles $\gamma_3(u)$ of u and its expected triangles $[\gamma_3(u)]$ (i.e., $dis_3(u) = \gamma_3(u) - [\gamma_3(u)]$). The minimization of the sum Δ_3 over all nodes is the objective of Problem 2 for $l = 3$ and $n = 3$.
- $dis_2(u) + dis_3(u)$ *discrepancy* is the sum of degree and triangle discrepancies. The minimization of the sum $\Delta_2 + \Delta_3$ over all nodes is the objective of Problem 2 for $l = 2$ and $n = 3$.
- *2-stars discrepancy* [Akers et al. 1994] is the difference of the number of *2-star* patterns $S_2(u)$ of u to its expected value $[S_2(u)]$. For each vertex u, the *2-star* measure corresponds to the number of open triplets $\{u, v, w\}$, where (u, v) and (u, w) are edges of the representative, but (v, w) is not.

The expected degrees and triangles (i.e., $[\gamma_2(u)]$ and $[\gamma_3(u)]$) are calculated by applying Lemma 3.3 for each vertex $u \in V$. The number $S_2(u)$ of *2-stars* is closely related to the above two metrics. In particular, $S_2(u)$ of a vertex u is the difference of all possible pairs of edges incident to u, minus the number of closed triangles containing u, i.e.,

$$S_2(u) = \frac{\gamma_2(u) \cdot (\gamma_2(u) - 1)}{2} - \gamma_3(u)$$

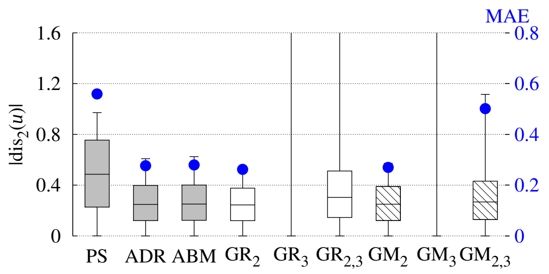
Thus, the expected value $[S_2(u)]$ is:

$$\begin{aligned}[S_2(u)] &= \Big[\frac{\gamma_2(u) \cdot (\gamma_2(u) - 1)}{2}\Big] - [\gamma_3(u)] = \frac{1}{2} \cdot ([\gamma_2^2(u)] - [\gamma_2(u)]) - [\gamma_3(u)] = \\ &= \frac{1}{2} \cdot (\mathsf{VAR}(\gamma_2(u)) + [\gamma_2(u)]^2 - [\gamma_2(u)]) - [\gamma_3(u)]\end{aligned} \tag{7}$$

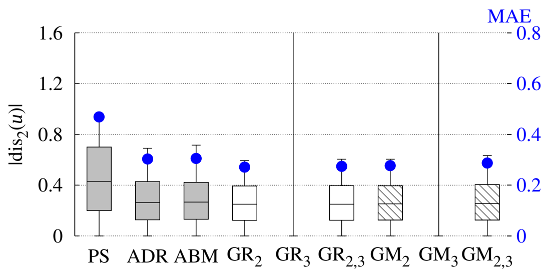
where $\mathsf{VAR}(\gamma_2(u))$ is the variance of the random variable $\gamma_2(u)$, derived from the probability distribution of degree values $\langle \Pr(\gamma_2(u)) = 0, \ldots, \Pr(\gamma_2(u) = k_u) \rangle$ (where $k_u = |\{(u, v) \in E\}|$). Each $\Pr(\gamma_2(u)) = k$ is computed by the dynamic-programming method of [Bonchi et al. 2014].

We plot our experimental results using *boxplots* (left y-axis), and the Mean Absolute Error (MAE), (disks projected on the right y-axis). Specifically, for each vertex $u \in V$ and a metric q, we first compute u's expected value over q, i.e., $[q(u)]$ and the corresponding value in the representative G^*, i.e., $q_{G^*}(u)$. MAE is defined as $\sum_{u \in V} |q_{G^*}(u) - [q(u)]|/|V|$. Each boxplot represents the error distribution of the vertex set V; the vertical line includes 95% of the vertices, the rectangle contains 50% of the vertices, and the horizontal line corresponds to the median error. The methods are grouped in three categories depicted by different patterns. The first category consists of methods PS, ABM and ADR that aim explicitly at Problem 1. The second (third) contains variants of the greedy (game theoretic) approach.

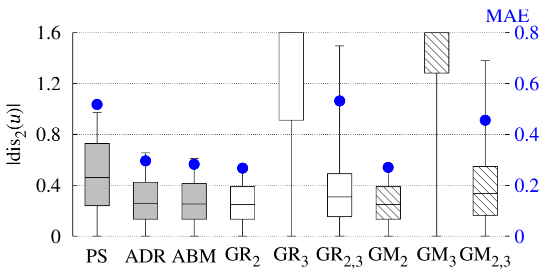
Figure 14 illustrates the boxplots and MAE of $|dis_2(u)|$ distribution. For instance, in Flickr for the representative produced by PS, 95% of the vertices have absolute vertex degree discrepancy less than 1, 50% of the vertices in the range [0.23, 0.76], and the median discrepancy is 0.49. The Mean Absolute Error is 0.56. PS is clearly outperformed by ADR and ABM, whose median discrepancy is below 0.4, in all datasets. The other two methods that focus on 2-discrepancy, GR_2 and GM_2, are slightly better than ADR and ABM. On the other hand, GR_3 and GM_3, which aim at minimizing the triangle discrepancy, fail (their MAE and parts of their boxplots are so high that they are excluded, for readability of the plots). This is due to their large number of edges, as discussed in the context of Figure 13. Although $\mathsf{GM}_{2,3}$ does not explicitly target vertex discrepancy, it captures it rather well. This can also be explained by its edge cardinality, which is similar to that of representatives focusing on Δ_2.



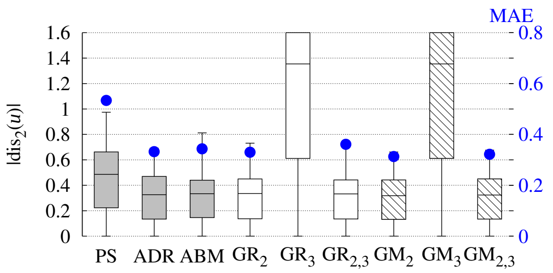
(a) Flickr



(b) Twitter



(c) DBLP

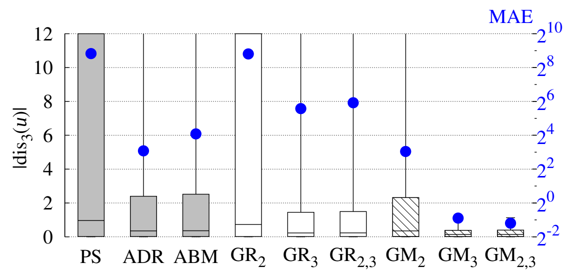


(d) BioMine

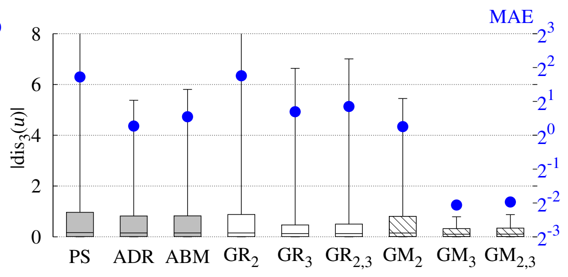
Fig. 14: Vertex degree discrepancy (Δ_2) (real graphs)

MAE is the ratio of Δ_2 over $|V|$ and corresponds to average degree discrepancy per vertex. In Flickr, where the expected average vertex degree is 22.93, most methods are very accurate. For instance, ADR, ABM, GR_2 and GM_2 yield an average degree discrepancy per node below 0.3. On the other hand, in BioMine, where the expected average degree is only 3.59, the relative error (the ratio of MAE over the expected average degree) of all methods is higher but still acceptable (observe that the values of MAE are similar in all datasets). In general, the proposed methods are most beneficial in dense graphs; if the uncertain graph is very sparse (i.e., small $|E|/|V|$ ratio, low edge probabilities), a single representative may fail to capture its properties.

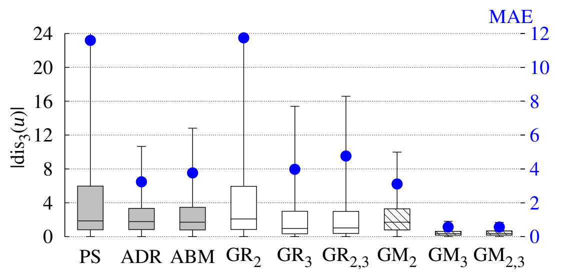
Figure 15 illustrates the boxplots and MAE of $|dis_3|$ distribution. GM_3 and $\mathsf{GM}_{2,3}$ exhibit the best performance. Specifically, in the densest dataset (Flickr), their average Δ_3 is more than two orders of magnitude lower than PS and GR_2, and more than one order of magnitude lower than ADR, ABM, GR_3 and $\mathsf{GR}_{2,3}$. Although GR_3 focuses on triangle discrepancy, it only yields very good results for BioMine; in the rest of the datasets it is outperformed by ADR and ABM. Observe that in those datasets the vertical line representing 95% of the vertices extends above those of ADR and ABM. This suggests that although GR_3 is able to capture most of the vertices well, its overall error is negatively affected by outliers. It is noteworthy that unlike GR_2 that fails in this metric, GM_2 manages to preserve relatively well Δ_3.



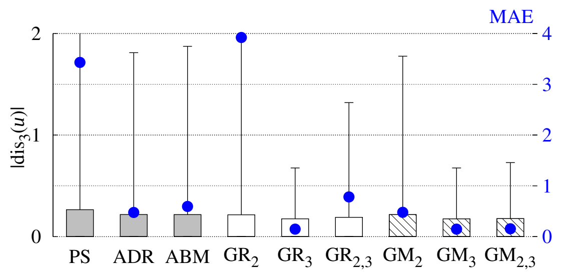
(a) Flickr



(b) Twitter



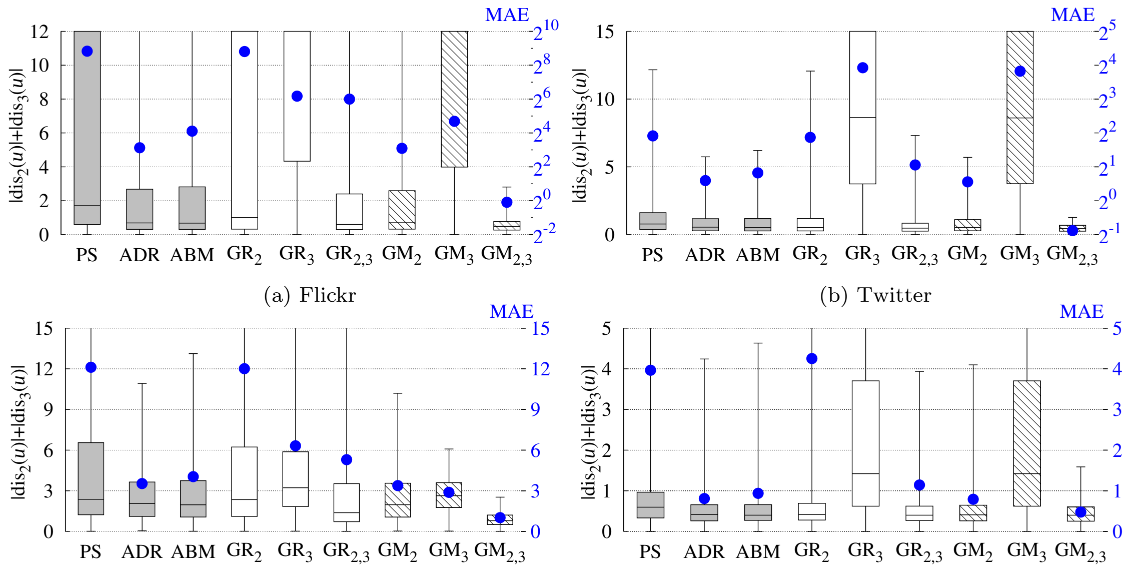
(c) DBLP



(d) BioMine

Fig. 15: Triangle discrepancy (Δ_3) (real graphs)

Figure 16 contains the boxplots and MAE of the discrepancy $|dis_2|+|dis_3|$. Overall, Figure 16 aggregates the results of Figures 14 and 15, i.e., a method that yields high error in $|dis_2|$ or $|dis_3|$, is also likely to under-perform in $|dis_2| + |dis_3|$. $\mathsf{GM}_{2,3}$ is the best representative, followed by GM_2, ADR, ABM and $\mathsf{GR}_{2,3}$ whose relative performance depends on the dataset. On the other hand, PS, GR_2 and GR_3, GM_3 fail in some datasets due to their inability to preserve $|dis_3|$ or $|dis_2|$, respectively.

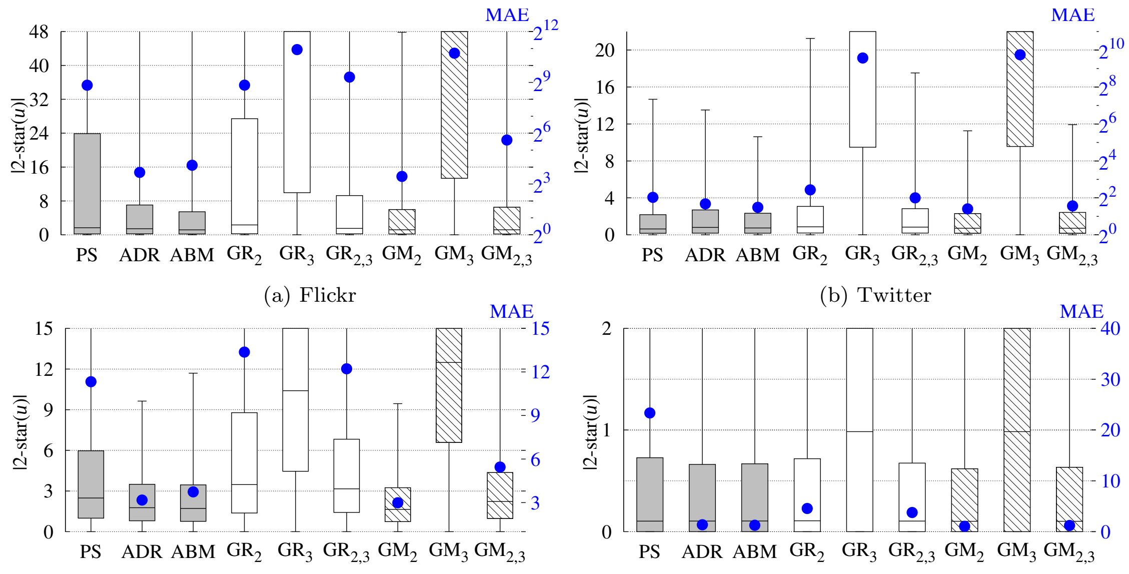


(a) Flickr (b) Twitter

(c) DBLP (d) BioMine

Fig. 16: Combined discrepancy ($\Delta_2 + \Delta_3$) (real graphs)

Figure 17 illustrates the performance of the methods in *2-star* discrepancy. According to Equation 7, the expected $S_2(u)$ depends more on its vertex than triangle cardinality because of the $[\gamma_2(u)]^2$ factor. Thus, the relative performance of the algorithms is similar to that in Figure 14, with GM_2, $\mathsf{GM}_{2,3}$, ABM and ADR having the highest accuracy. GR_2 yields large error due to its failure on Δ_3. Summarizing the results on structural measures, $\mathsf{GM}_{2,3}$, and to a lesser degree GM_2 achieve the best quality overall. ABM and ADR have balanced performance under all settings, whereas the rest of the algorithms may fail in some datasets for a measure beyond their intended objective function.



(a) Flickr (b) Twitter

(c) DBLP (d) BioMine

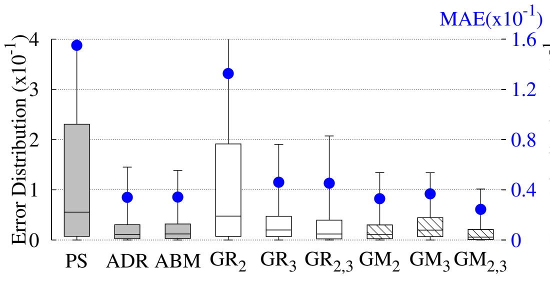
Fig. 17: 2stars discrepancy (real graphs)

6.3. Query metrics

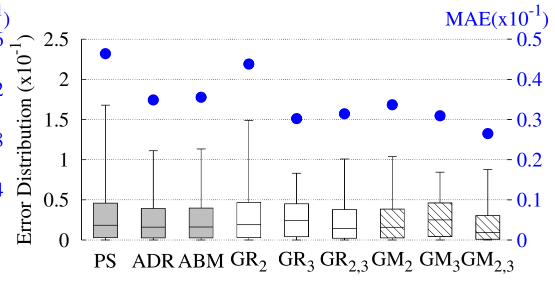
We evaluate the accuracy of our methods in the following graph related queries:

- *Clustering coefficient* is a measure of how close neighbors of a vertex are to forming a clique. Specifically, it is the ratio of the number of edges between the neighbors of a vertex to the maximum number of such links. It is an important metric for search strategies [Fraigniaud 2007] and social networks [Kossinets and Watts 2006].
- *Betweenness centrality* is a measure of the node's importance in the graph: it corresponds to the ratio of shortest paths that pass through the node over all pairs of shortest paths. It has been used widely to assess link value of ecological graphs [Gonzalez et al. 2010], and router utilization of communication networks [Tizghadam and Leon-Garcia 2010].
- *Shortest path distance* is the percentage of pairs at a certain distance, over all pairs of reachable vertices. This metric is crucial for spatial queries [Potamias et al. 2010], routing protocols, and in general, any task involving shortest path computations.

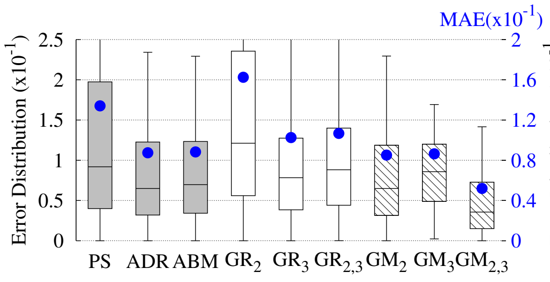
Since there do not exist closed formulae for the query metrics, the *ground truth* is approximated by Monte Carlo sampling. Specifically, we create a number of random instances of the input uncertain graph, and we compute the *expected* value of each metric using the average of the sampled graphs. We use 1000 samples since it has been shown [Potamias et al. 2010; Jin et al. 2011b] that they usually suffice to achieve accuracy convergence. Similarly to the previous section, we plot our experimental results using boxplots (left y-axis), and the Mean Absolute Error (MAE), (disks projected on the right y-axis), starting with the clustering coefficient in Figure 18. In general, the ranking of the algorithms in terms of accuracy, is analogous to that for Δ_3 (see Figure 15) because the clustering coefficient of a vertex u is highly correlated to the number of triangles containing u. $GM_{2,3}$ has the lowest MAE in all datasets except BioMine. GM_2, GM_3, GR_3, $GR_{2,3}$, ABM and ADR yield acceptable results, but GR_2 and PS lead to large error.



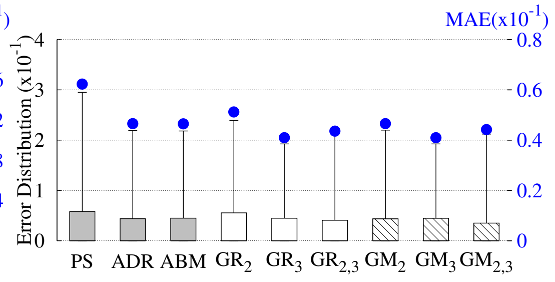
(a) Flickr



(b) Twitter



(c) DBLP

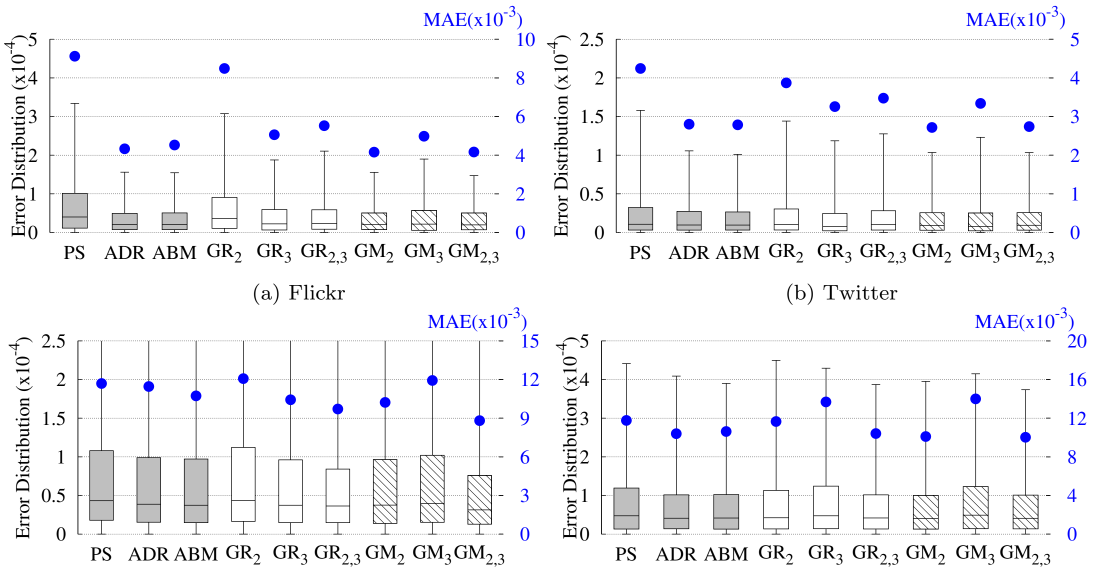


(d) BioMine

Fig. 18: Clustering coefficient distribution (real graphs)

Betweenness centrality and *shortest-path distance* are very expensive because they involve *all-pairs shortest path* computations. Their evaluation over 1000 samples of large graphs is prohibitive. To overcome this problem, given an uncertain graph, we use *forest fire* [Leskovec and Faloutsos 2006] to create a subgraph that has similar properties, and perform the evaluation on the reduced graph. The number of vertices and edges in the reduced graphs are: i) Flickr, 5000 and 655275, ii) Twitter 10000 and 353399, iii) DBLP, 5000 and 76884 and iv) BioMine, 5000 and 69367. Note that the method used to create reduced graphs is orthogonal to our work; we expect the proposed techniques to have similar performance with other size reduction methods.

Figure 19 shows the betweenness centrality boxplots and MAE. The representatives of ADR, ABM, GM_2 and $GM_{2,3}$ capture very well this metric in all datasets; $GM_{2,3}$ is best in DBLP, whereas there are not considerable differences in the other datasets. PS performs well for DBLP, acceptably for BioMine and poorly for Flickr and Twiter. Similar to previous diagrams, GREEDY methods are usually outperformed by the corresponding GAME approaches with the same objective function.



(a) Flickr

(b) Twitter

(c) DBLP

(d) BioMine

Fig. 19: Betweenness centrality (reduced graphs)

Figure 20 plots the mean absolute error for the average *shortest path distance* between every pair of vertices. Clearly, in all datasets, approaches that aim at minimizing only Δ_3, i.e., GR_3 and GM_3, provide the worst results. Due to the high density of their representatives (see Figure 13), they seriously underestimate the distance. PS and GR_2 also exhibit poor performance. The rest of the methods yield similar accuracy.

Figure 21 illustrates the shortest path distance error distribution, i.e., the MAE versus the distance of the two end-nodes. To keep the diagrams readable, we only include the benchmark PS, and a single method of each category, namely ABM, GR_3 and $GM_{2,3}$. The relative performance of the methods is consistent with Figure 20. Note that MAE decreases as the distance increases. This occurs because there are numerous alternative paths between pairs of vertices that have long distances. Even if the actual shortest path is not maintained in the representative, another one, with similar distance, is likely to exist.

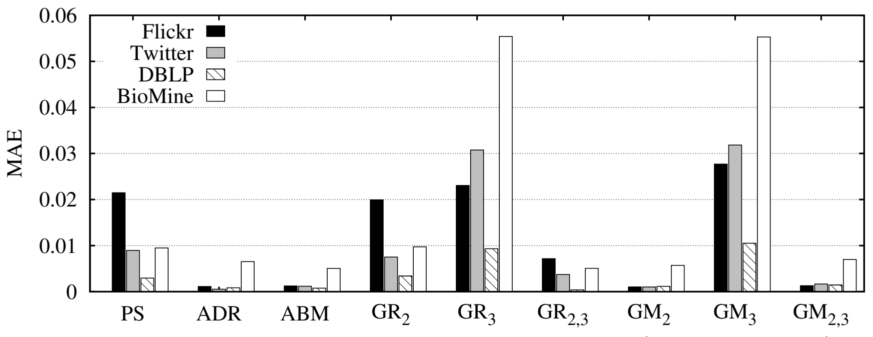
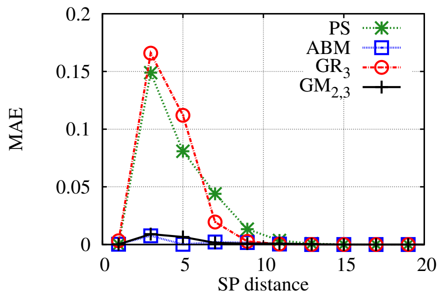
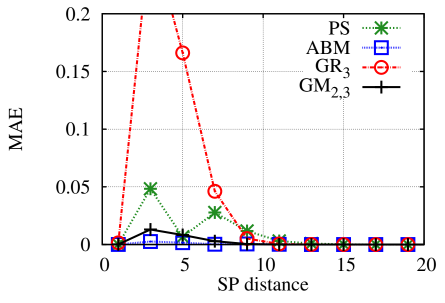


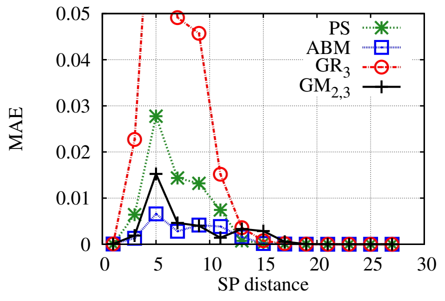
Fig. 20: MAE for average shortest path distance (reduced graphs)



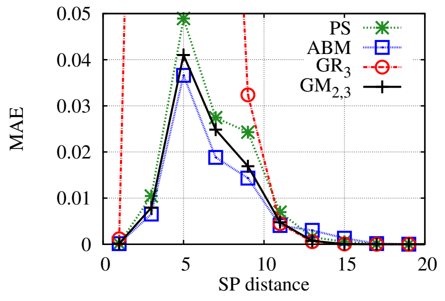
(a) Flickr



(b) Twitter



(c) DBLP



(d) BioMine

Fig. 21: Shortest path distance average error (reduced graphs)

The summary of the query metrics agrees with that of the structural measures. $\mathsf{GM}_{2,3}$ provides the highest accuracy for most metrics and datasets, followed by GM_2. Although ABM and ADR rarely achieve the best results, they do not fail either. On the other hand, the GREEDY approaches are unpredictable. Regarding the objective functions, minimization of Δ_3 should be applied only for metrics, such as the clustering coefficient, that are highly correlated to the number of triangles. For most other metrics, minimization of $\Delta_2 + \Delta_3$ is likely to yield the best results.

6.4. Efficiency experiments

All methods were implemented in C++ and executed in a single core of an Intel Xeon E5-2660 with 2.20GHz CPU and 96GB RAM. Figure 22 illustrates the running times in logarithmic scale. ABM is the fastest algorithm, generating its representative in less than 10 seconds for all datasets. PS, ADR, GM_2 are also very efficient and their running time never exceeds 100 seconds. At the other extreme, GR_3 requires several hours to terminate for BioMine. In general, GREEDY approaches are the slowest, which is expected by their high complexity. The objective function also affects the cost, with methods aiming at Δ_3, i.e., GR_3 and GM_3, being the most expensive, followed by those that minimize $\Delta_2 + \Delta_3$.

For ease of comparison, the last three sets of columns in Figure 22 contain the running times of the three query metrics on a single sample/representative. Specifically, *CC* corresponds to the clustering coefficient, *BC* to betweenness centrality and *SP* to shortest path distance. Even the most efficient query *CC* on a single sample requires roughly the same time as the generation of a representative by PS, ADR and GM_2, and is more expensive than ABM. *BC* and *SP* are much slower due to the all-pair shortest path computations. In Monte Carlo approaches, all queries must be executed on each individual sample (in our experiments 1000 times). In addition, these samples have a non-negligible generation cost, which is not considered in the diagram. Thus, the overhead of the proposed techniques is very small compared to that of query processing.

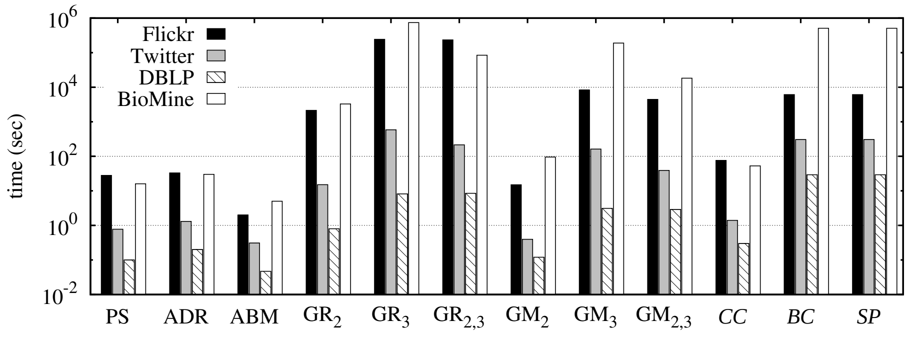
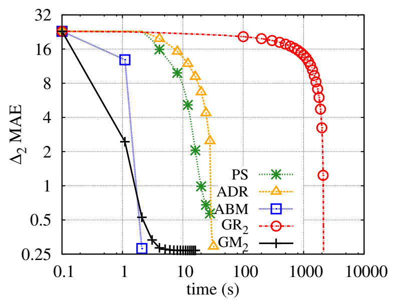
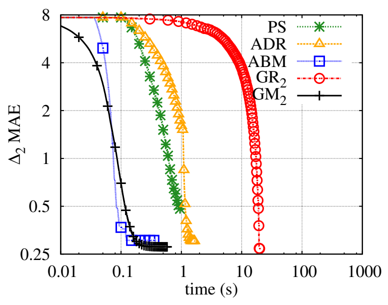


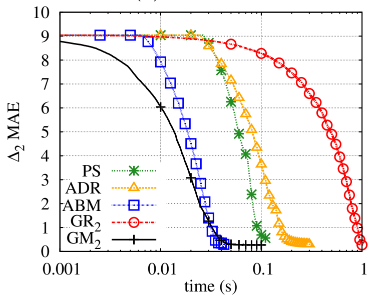
Fig. 22: Running times (real graphs)



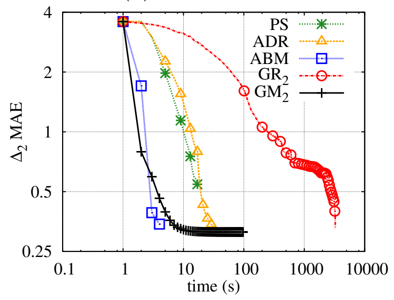
(a) Flickr



(b) Twitter



(c) DBLP

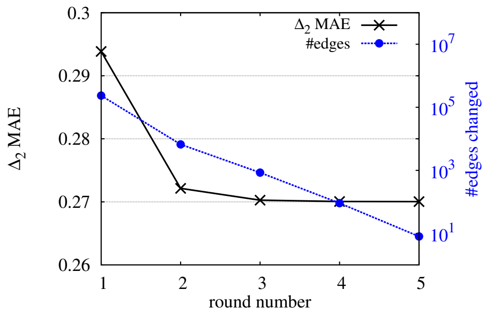


(d) BioMine

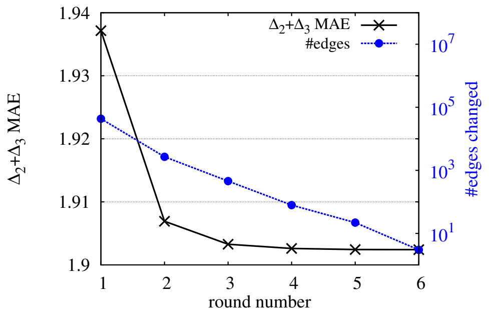
Fig. 23: Convergence time (real graphs)

Figure 23 illustrates the average Δ_2 versus time in log-log scale for the methods that aim at minimizing Δ_2. We can classify the algorithms in three categories based on their speed of convergence. i) ABM and GM_2 converge very fast to good quality solutions in all datasets. ii) ADR and PS need more time to create a good representative, although their execution times are comparable to that of GM_2. iii) In addition to being the least efficient, GR_2 is also the slowest to converge.

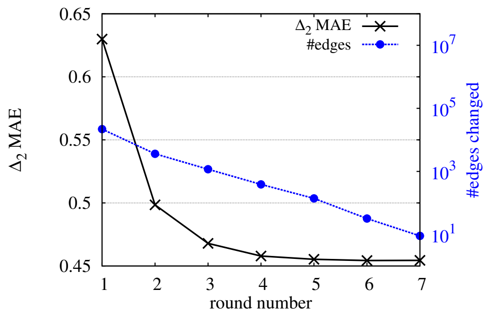
Since as discussed in the previous subsections, GM_2 and $GM_{2,3}$ produce in general the best representatives, the last experiment focuses on their performance. Specifically, in each diagram of Figure 24, the x-axis corresponds to the round number, the left y-axis shows the discrepancy (Δ_2 or $\Delta_2 + \Delta_3$), and the right y-axis plots the number of edges that changed strategy during the round. For instance, in Figure 24(a), GM_2 in Flickr terminates after 6 rounds; the last round is omitted because the discrepancy remains the same and no edge switches strategy. The initial Δ_2 discrepancy of the seed graph is 2.453 and drops to 0.294 after the first round. The number of strategy changes decreases exponentially, which causes the execution time of each round to diminish accordingly. Figure 24(b) shows the corresponding diagram for $\Delta_2 + \Delta_3$ in Twitter. Although $\Delta_2 + \Delta_3$ is not the objective function of GM_2, the convergence behavior is similar to that of Figure 24(a).



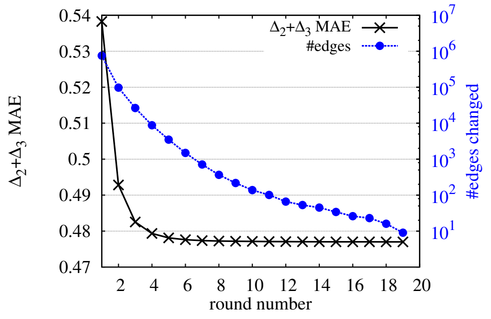
(a) Flickr GM_2



(b) Twitter GM_2



(c) DBLP $GM_{2,3}$



(d) BioMine $GM_{2,3}$

Fig. 24: Performance of GAME (real graphs)

Figures 24(c) and 24(d) repeat the same experiments for $GM_{2,3}$ in DBLP and BioMine respectively. The results are analogous to GM_2 except that more rounds are required for termination, especially for BioMine. This is because, as shown in Figure 13, BioMine yields the representative with the largest number of edges. In all cases the number of rounds is at most 20, whereas the discrepancy always converges within the first few rounds. Accordingly, for large graphs and time critical applications, a small number of rounds of GAME can produce representatives of high quality, without necessarily reaching a Nash equilibrium.

7. CONCLUSION

In this paper we propose extracting representative instances of uncertain graphs. Expensive tasks can then be processed by applying deterministic algorithms on these instances. We focus on two problems: the first aims at minimizing the vertex degree discrepancy between the representative and the uncertain graph, while the second minimizes n-clique discrepancies. For Problem 1, we present ADR a heuristic technique, and ABM that utilizes matching algorithms. For Problem 2, we develop methods based on the greedy and game theoretic frameworks. In order to assess the quality of representatives, we perform extensive experiments on real datasets. Our results confirm that indeed the proposed methods approximate well various structural measures and query metrics. Given the cost savings of our techniques with respect to Monte Carlo sampling, we expect them to have a significant impact on query processing for uncertain graphs.

In the future we intend to investigate additional properties that maybe of interest for specialized tasks, e.g., graph pattern mining, graph clustering, etc. It will also be interesting to generate and combine multiple representatives for better approximation. We also aim at extending our methods to alternative uncertain settings such as, time dependent or streaming graphs, attributed graphs etc., where the extraction of representatives is even more challenging. Finally, an interesting direction is the incorporation of subgraph sampling in the process of representative extraction. The idea is to generalize the problem definitions, so that in addition to preserving the structural properties of the uncertain graph, the representatives also reduce the number of nodes to further facilitate efficiency.

Acknowledgements

Panos Parchas and Dimitris Papadias were supported by GRF grants HKUST 617713 and 16201615 from Hong Kong RGC.

REFERENCES

Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. 1987. On the Representation and Querying of Sets of Possible Worlds. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '87)*. 34–48. DOI:http://dx.doi.org/10.1145/38713.38724

Eytan Adar and Christopher Re. 2007. Managing uncertainty in social networks. *IEEE Data Engineering Bulletin* 30, 2 (July 2007), 15–22.

Charu C. Aggarwal and Philip S. Yu. 2009. A Survey of Uncertain Data Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering* 21, 5 (May 2009), 609–623. DOI:http://dx.doi.org/10.1109/TKDE.2008.190

William Aiello, Fan Chung, and Linyuan Lu. 2000. A Random Graph Model for Massive Graphs. In *Proceedings of the ACM Symposium on Theory of Computing (STOC '00)*. 171–180. DOI:http://dx.doi.org/10.1145/335305.335326

Sheldon B. Akers, Dov Harel, and Balakrishnan Krishnamurthy. 1994. The Star Graph: An Attractive Alternative to the N-cube. In *Interconnection Networks for High-performance Parallel Computers*. IEEE Computer Society Press, Los Alamitos, CA, USA, 145–152.

Saurabh Asthana, Oliver D. King, Francis D. Gibbons, and Frederick P. Roth. 2004. Predicting protein complex membership using probabilistic network reliability. *Genome Research* 14, 4 (June 2004), 1170–1175. DOI:http://dx.doi.org/10.1101/gr.2203804

Joseph Blitzstein and Persi Diaconis. 2011. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics* 6, 4 (March 2011), 489–522. DOI:http://dx.doi.org/10.1080/15427951.2010.557277

Paolo Boldi, Francesco Bonchi, Aristides Gionis, and Tamir Tassa. 2012. Injecting Uncertainty in Graphs for Identity Obfuscation. *Proceedings of the VLDB Endowment* 5, 11 (July 2012), 1376–1387. DOI:http://dx.doi.org/10.14778/2350229.2350254

Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core Decomposition of Uncertain Graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. 1316–1325. DOI:http://dx.doi.org/10.1145/2623330.2623655

Fan Chung and Linyuan Lu. 2002. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences* 99, 25 (Dec. 2002), 15879–15882. DOI:http://dx.doi.org/10.1073/pnas.252631999

Fan Chung, Linyuan Lu, and Van H. Vu. 2003. Spectra of Random Graphs with Given Expected Degrees. *Proceedings of the National Academy of Sciences* 100, 11 (May 2003), 6313–6318. DOI:http://dx.doi.org/10.1073/pnas.0937490100

Nilesh Dalvi and Dan Suciu. 2007. Efficient Query Evaluation on Probabilistic Databases. *The VLDB Journal* 16, 4 (Oct. 2007), 523–544. DOI:http://dx.doi.org/10.1007/s00778-006-0004-3

Charo I. Del Genio, Hyunju Kim, Zoltn Toroczkai, and Kevin E. Bassler. 2010. Efficient and Exact Sampling of Simple Graphs with Given Arbitrary Degree Sequence. *PLoS ONE* 5, 4 (April 2010), e10012. DOI:http://dx.doi.org/10.1371/journal.pone.0010012

Paul Erdös and Tibor Gallai. 1960. Graphs with prescribed degrees of vertices (Hungarian). *Mat. Lapok* 11 (1960), 264–274.

Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On Power-law Relationships of the Internet Topology. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '99)*. 251–262. DOI:http://dx.doi.org/10.1145/316188.316229

Pierre Fraigniaud. 2007. Small Worlds as Navigable Augmented Networks: Model, Analysis, and Validation. In *Algorithms ESA*. Vol. 4698. Springer, 2–11. DOI:http://dx.doi.org/10.1007/978-3-540-75520-3_2

Ana M. Gonzalez, Bo Dalsgaard, and Jens M. Olesen. 2010. Centrality measures and the importance of generalist species in pollination networks. *Ecological Complexity* 7, 1 (March 2010), 36–43. DOI:http://dx.doi.org/10.1016/j.ecocom.2009.03.008

Seifollah L. Hakimi. 1962. On realizability of a set of integers as degrees of the vertices of a linear graph. *Journal of the Society for Industrial and Applied Mathematics* 10, 3 (Sept. 1962), 496–506. http://www.jstor.org/stable/2098746

Stefan Hougardy. 2009. Linear Time Approximation Algorithms for Degree Constrained Subgraph Problems. In *Research Trends in Combinatorial Optimization*. Springer, 185–200. DOI:http://dx.doi.org/10.1007/978-3-540-76796-1_9

Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. 2013. Massive Graph Triangulation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. 325–336. DOI:http://dx.doi.org/10.1145/2463676.2463704

Christian Hübler, Hans-Peter Kriegel, Karsten Borgwardt, and Zoubin Ghahramani. 2008. Metropolis Algorithms for Representative Subgraph Sampling. In *Proceedings of the IEEE International Conference on Data Mining (ICDM '08)*. 283–292. DOI:http://dx.doi.org/10.1109/ICDM.2008.124

Ruoming Jin, Lin Liu, and Charu C. Aggarwal. 2011a. Discovering Highly Reliable Subgraphs in Uncertain Graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*. 992–1000. DOI:http://dx.doi.org/10.1145/2020408.2020569

Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. 2011b. Distance-constraint Reachability Computation in Uncertain Graphs. *Proceedings of the VLDB Endowment* 4, 9 (June 2011), 551–562. DOI:http://dx.doi.org/10.14778/2002938.2002941

David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the Spread of Influence Through a Social Network. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*. 137–146. DOI:http://dx.doi.org/10.1145/956750.956769

Arijit Khan, Francesco Bonchi, Aristides Gionis, and Francesco Gullo. 2014. Fast Reliability Search in Uncertain Graphs. In *Proceedings of the International Conference on Extending Database Technology (EDBT '14)*. 535–546. DOI:http://dx.doi.org/10.5441/002/edbt.2014.48

Jon Kleinberg. 2006. Complex networks and decentralized search algorithms. In *Proceedings of the International Congress of Mathematicians (ICM '06)*. 1019–1044.

George Kollios, Michalis Potamias, and Evimaria Terzi. 2013. Clustering Large Probabilistic Graphs. *IEEE Transactions on Knowledge and Data Engineering* 25, 2 (Feb. 2013), 325–336. DOI:http://dx.doi.org/10.1109/TKDE.2011.243

Gueorgi Kossinets and Duncan J. Watts. 2006. Empirical Analysis of an Evolving Social Network. *Science* 311, 5757 (Jan. 2006), 88–90. DOI:http://dx.doi.org/10.1126/science.1116869

Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker Graphs: An Approach to Modeling Networks. *Journal of Machine Learning Research* 11 (March 2010), 985–1042. http://dl.acm.org/citation.cfm?id=1756006.1756039

Jure Leskovec and Christos Faloutsos. 2006. Sampling from Large Graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '06)*. 631–636. DOI:http://dx.doi.org/10.1145/1150402.1150479

Rong-Hua Li, Jeffrey Xu Yu, Rui Mao, and Tan Jin. 2014. Efficient and accurate query evaluation on uncertain graphs via recursive stratified sampling. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE '14)*. 892–903. DOI:http://dx.doi.org/10.1109/ICDE.2014.6816709

David Liben-Nowell and Jon Kleinberg. 2003. The Link Prediction Problem for Social Networks. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '03)*. 556–559. DOI:http://dx.doi.org/10.1145/956863.956972

Fredrik Liljeros, Christofer R. Edling, Luis A. Nunes Amaral, H Eugene Stanley, and Yvonne Åberg. 2001. The web of human sexual contacts. *Nature* 411, 6840 (June 2001), 907–908. DOI:http://dx.doi.org/10.1038/35082140

Lin Liu, Ruoming Jin, Charu C. Aggarwal, and Yelong Shen. 2012. Reliable clustering on uncertain graphs. In *Proceedings of the IEEE International Conference on Data Mining (ICDM '12)*. 459–468. DOI:http://dx.doi.org/10.1109/ICDM.2012.11

Yang-Yu Liu, Jean-Jacques Slotine, and Albert-Laszlo Barabasi. 2011. Controllability of complex networks. *Nature* 473, 7346 (May 2011), 167–173. DOI:http://dx.doi.org/10.1038/nature10011

László Lovász. 1970. Subgraphs with prescribed valencies. *Journal of Combinatorial Theory* 8, 4 (1970), 391–416. DOI:http://dx.doi.org/10.1016/S0021-9800(70)80033-3

Priya Mahadevan, Dmitri Krioukov, Kevin Fall, and Amin Vahdat. 2006. Systematic Topology Analysis and Generation Using Degree Correlations. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '06)*. 135–146. DOI:http://dx.doi.org/10.1145/1159913.1159930

Miller McPherson, Lynn Smith-Lovin, and James M. Cook. 2001. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology* 27, 1 (Aug. 2001), 415–444. DOI:http://dx.doi.org/10.1146/annurev.soc.27.1.415

Julián Mestre. 2006. Greedy in Approximation Algorithms. In *Algorithms ESA*. Vol. 4168. Springer, 528–539. DOI:http://dx.doi.org/10.1007/11841036_48

Silvio Micali and Vijay V. Vazirani. 1980. An O($\sqrt{|V|} \cdot |E|$) algorithm for finding maximum matching in general graphs. In *Proceedings of the IEEE Annual Symposium on Foundations of Computer Science (FOCS '80)*. 17–27. DOI:http://dx.doi.org/10.1109/SFCS.1980.12

Daniele Micciancio. 2001. The Hardness of the Closest Vector Problem with Preprocessing. *IEEE Transactions on Information Theory* 47, 3 (March 2001), 1212–1215. DOI:http://dx.doi.org/10.1109/18.915688

Milena Mihail and Nisheeth K Vishnoi. 2002. *Position Paper, Approximate and Randomized Algorithms for Communication Networks (ARACNE '02)* 142 (2002), 1–11.

Dov Monderer and Lloyd S. Shapley. 1996. Potential games. *Games and Economic Behavior* 14, 1 (May 1996), 124–143. DOI:http://dx.doi.org/10.1006/game.1996.0044

Walaa Eldin Moustafa, Angelika Kimmig, Amol Deshpande, and Lise Getoor. 2014. Subgraph Pattern Matching over Uncertain Graphs with Identity Linkage Uncertainty. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE '14)*. 904–915. DOI:http://dx.doi.org/10.1109/ICDE.2014.6816710

Jaroslav Nešetřil and Svatopluk Poljak. 1985. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae* 26, 2 (Jan. 1985), 415–419.

Panos Parchas, Francesco Gullo, Dimitris Papadias, and Franceseco Bonchi. 2014. The Pursuit of a Good Possible World: Extracting Representative Instances of Uncertain Graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. 967–978. DOI:http://dx.doi.org/10.1145/2588555.2593668

Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. 2010. K-nearest Neighbors in Uncertain Graphs. *Proceedings of the VLDB Endowment* 3, 1-2 (Sept. 2010), 997–1008. DOI:http://dx.doi.org/10.14778/1920841.1920967

Gerardo Rubino. 1998. Network reliability evaluation. *State-of-the art in performance modeling and simulation* (March 1998), 275–302. DOI:http://dx.doi.org/10.1002/wics.81

Petteri Sevon, Lauri Eronen, Petteri Hintsanen, Kimmo Kulovesi, and Hannu Toivonen. 2006. Link Discovery in Graphs Derived from Biological Databases. In *Proceedings of the Springer International Conference on Data Integration in the Life Sciences (DILS'06)*. 35–49. DOI:http://dx.doi.org/10.1007/11799511_5

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2005. *Introduction to Data Mining*. Addison-Wesley Longman Publishing Co., Inc.

Hongsuda Tangmunarunkit, Ramesh Govindan, Sugih Jamin, Scott Shenker, and Walter Willinger. 2002. Network Topology Generators: Degree-based vs. Structural. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '02)*. 147–159. DOI:http://dx.doi.org/10.1145/633025.633040

Ali Tizghadam and Alberto Leon-Garcia. 2010. Betweenness centrality and resistance distance in communication networks. *IEEE Network* 24, 6 (Nov.-Dec. 2010), 10–16. DOI:http://dx.doi.org/10.1109/MNET.2010.5634437

Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM Journal of Computing* 8, 3 (1979), 410–421. DOI:http://dx.doi.org/10.1137/0208032

Christian von Mering, Martijn A. Huynen, Daniel Jaeggi, Steffen Schmidt, Peer Bork, and Berend Snel. 2003. STRING: a database of predicted functional associations between proteins. *Nucleic Acids Research* 31, 1 (Jan. 2003), 258–261. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC165481/

Ye Yuan, Lei Chen, and Guoren Wang. 2010. Efficiently Answering Probability Threshold-Based Shortest Path Queries over Uncertain Graphs. In *Database Systems for Advanced Applications*. Vol. 5981. Springer, 155–170. DOI:http://dx.doi.org/10.1007/978-3-642-12026-8_14

Ye Yuan, Guoren Wang, Lei Chen, and Haixun Wang. 2012. Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases. *Proceedings of the VLDB Endowment* 5, 9 (May 2012), 800–811. DOI:http://dx.doi.org/10.14778/2311906.2311908

Ye Yuan, Guoren Wang, Haixun Wang, and Lei Chen. 2011. Efficient Subgraph Search over Large Uncertain Graphs. *Proceedings of the VLDB Endowment* 4, 11 (Jan. 2011), 876–886.

Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. 2010a. Finding top-k Maximal Cliques in an Uncertain Graph. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE '10)*. 649–652. DOI:http://dx.doi.org/10.1109/ICDE.2010.5447891

Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. 2010b. Mining Frequent Subgraph Patterns from Uncertain Graph Data. *IEEE Transactions on Knowledge and Data Engineering* 22, 9 (May 2010), 1203–1218. DOI:http://dx.doi.org/10.1109/TKDE.2010.80