HARVARD UNIVERSITY, COMPUTER SCIENCE DEPARTMENT

Approximate Influence Maximization as a Distributed Constraint Optimization Problem

Nathaniel Tucker & Michael Tingley

December 9, 2013

We propose a novel approach for approximately maximizing influence in the independent cascade model. The new model is based on "profit maximization" for selecting initial influencers in a graph, and frames this problem as a distributed constraint optimization problem. We further propose a reduction so that this can be used to approximately solve the canonical k-best influencers problem widely used in the literature. Finally, we analyze the performance of this algorithm on a number of different graph structures against the performance of an approximation algorithm for the k-best influencers problem.

1 INTRODUCTION

Information dissemination in social networks has been studied in order to determine the flow of information, which in turn can be used to maximize influence on the entire network with only a few initial influencing agents. It is an important problem facing todays politicians, advertisers, and anyone trying to study how information flows through networks. Traditionally literature will seek to choose the ideal k initial agents to maximize network influence, however approaches can often be an NP-hard problem.[1]

One can easily see two glaring issues with this formulation. We will account for and resolve these issues. First and most importantly, the goal of influence maximization is to choose the best k nodes. While this notion has value and makes academic sense, industry has different values. One is not trying to find the top k nodes, but rather concerned with profit. This is where our notion of a profit maximizing influence maximization comes into play. Second, and stated above, the problem itself is NP-hard, which suggests that it is computationally intractable. We address these two problems with our new proposal.

We consider a variant of this problem, in which the goal is to maximize the profit induced from influencing a number of nodes. Intuitively, the profit is the value earned from influencing nodes minus the cost associated with choosing the initial influencers. Profit is a more intuitive concept and represents the real demands of advertisers in the current market.

To accompany this paradigm shift, we use a series of novel representations and approximation techniques to better capture the dynamics of profit in an influence maximization function. We use a Distributed Constraint

[1]Assuming certain information flow models, Independent Cascade Model being one of them

Network to model the information flow for an influence maximization problem. The graph then undergoes local transformations and local optimization, allowing the entire process to be distributed.

Not only does our approach capture the intuitive notion of profit in influence maximization, but also it realizes this notion in such a way that allows the entire process to be distributed. This creates a more intuitive and computationally tractable framework for influence maximization. That is the motivation behind the paper.

2 Background and Related Work

We will describe our model of influence, the formulation of the problem itself and our new objective function 2.2. We will also discuss our framework for solving the problem and the approximate solvers themselves. The next section will go into the creation of our constraint network and our two created models.

2.1 Influence on Social Networks

Social networks are well-studied constructs that are often used to represent the flow and exchange of information between agents in the network. Thus it is perfectly natural to consider this flow of information, *influence*, and the potential problem, which is to maximize it. While Kleinberg was the first to propose the concept of *influence maximization*, it currently takes many forms [5].

Our version of propagation in a social network will be as follows. An *opinion* of a node is a binary function in $\{0,1\}$, and initially all nodes have opinion 0. A node is *influenced* at time step t, if at time step t, its opinion changed to 1. We will now describe three major models of influence that describe a stochastic process in which nodes are influenced to change their opinion. Our model is defined for some finite graph $G=(V,E)$ and time step $t\in\mathbb{N}$. [9]

The most cited models in literature for the *spread* of this information are:

- **The Voter Model:** In the voter model we assume the graph has self loops, and for each $v\in V$ define $N(v)$ to be the neighbors of v (including v itself) and $d(v)=|N(v)|$. In the voter model at time t, each node v adopts the opinion its neighbor had at time $t-1$ with probability $1/d(v)$.

- **Independent Cascade:** In this model every edge $(u,v)\in E$ is assigned with some weight $p_{u,v}\in[0,1]$. At time step t a node u adopts the opinion of its neighbor v with probability $p_{u,v}$ if and only if v adopted an opinion at time $t-1$.

- **Linear Threshold:** Here every edge also has a weight $p_{u,v}\in[0,1]$ and in addition every node u has some threshold $t_u\in[0,1]$ *chosen uniformly at random*. A node u is influenced at time t if at time step $t-1$, $\sum_{v\in N(u)} p_{u,v}\geq t_u$. In other words, if the total incoming weight exceeds the threshold.

Our paper will specifically focus on **Independent Cascade**. However the crux of the issue is to maximize this influence.

2.1.1 Influence Maximization

Traditional influence maximization is concerned with the question: given a graph $G=(V,E)$, an influence function $f:2^V\to\mathbb{R}_+$ and budget $k\in\mathbb{N}_+$, find a subset of nodes of size k:

$$\{v_1,\ldots v_k\}=S\subseteq V \text{ s.t. } S\in\operatorname{argmax}_{T:|T|\leq k} f(T) \tag{2.1}$$

Specifically a $1-1/e-\epsilon$ approximation has been derived for the independent cascade already, shown in Algorithm 1.[6]

However our approach will consider the question: given a graph $G=(V,E)$, an influence function $f:2^V\to\mathbb{R}_+$, and a cost function $c:V\to\mathbb{R}_+$, find a set of nodes:

$$S \subseteq V \text{ s.t. } S \in \text{argmax}_T f(T) \sum_{v \in T} c(v) \tag{2.2}$$

We are considering the set of nodes that achieve the highest *profit*, rather than just highest influence. Thus we will be seeking to maximize this objective function.

2.2 Distributed Constraint Networks

Given our model of influence spread and our objective function 2.2, we will now describe the framework for to maximize our function and the optimization tools themselves. We use constraint networks as our framework for the spread of influence. For a more detailed description of constraints see Appendix 8.1.1

2.2.1 Constraint Network

A Constraint Network is defined by the three tuple: (X, D, C), where $X = \{x_1, ...x_n\}$ is a discrete set of variables, $D = \{D^1, ...D^n\}$ is a unique set of variable domains enumerating all possible values of the corresponding variables. And $C = \{C_1, ...C_m\}$ is a set of constraints, where constraints can be of two types: hard and soft. For the purposes of our work, we will consider networks of soft constraints, and binary domains. See appendix section 8.1.2 for discussion on these assumptions.

2.2.2 Distributed Constraint Network

A Distributed Constraint Network can be formally defined by the four tuple (X, D, C, A). Where each refer previously to a Constraint Network in its simpler form, except A. This is the set of agents $A = \{A_1, ..., A_k\}$ where each agent can only control a subset $X_i \subseteq X$, and each variable is assigned to one agent (injective).

There are in particular a few more stipulations that we have for the agents. Agents can control only the variables that were assigned to them, meaning they can only see these variables and only change these variables. Furthermore, agents can only see the constraints that involve variables that they control. And finally, we will coin the term neighbors, which will be any two agents for which there exists a constraint that depends on variables that they both control. Only neighbors can communicate with each other. This is a Distributed Constraint Network. [10]

Thus our original problem of finding the best influence maximizing nodes on the graph, is translated into the framework of a distributed constraint optimization problem (DCOP).

2.3 Distributed Constraint Network Solvers

Here we will describe how we go about solving these DCOP.

In soft Constraint Networks, one faces the problem of constraint optimization problems. The goal is to find the best solution, or to optimize a global function $F(\bar{a}) = F(\bar{a}) = \sum_i F_i(\bar{a}_i)$ where $\bar{a} = \{a^1, ...a^n\}$ given that $a^j \in D^j$. Thus a general constraint optimization problem will seek to maximize:

$$\bar{a}^* = \text{argmax}_{\bar{a}} \sum_i F_i(\bar{a}_i) \tag{2.3}$$

This is generally the case. We however will be dealing with DCOP. Our setting will involve agents that control only one variable, again without loss of generality. However, we will make the assumption that agents are not self interested, and their one goal is to optimize the global function.

2.3.1 Optimal DCOP Algorithms

Finding an optimal solution for a DCOP is NP-Hard problem. This is most easily seen by reducing the problem to deciding 3-colorability of a graph. However solutions do exist. The solutions are either Search Based or Dynamic

Programing Based. An example search based algorithm would be ADOPT (Asynchronous Distributed OPTimization). And an example of a dynamic solution would be DPOP (Dynamic Programming Optimization Protocol). [1] These solutions are computationally and memory intensive, and for the purposes of this paper we will not focus on them.

2.3.2 Approximate DCOP Algorithms

Again, solving a DCOP is an NP-Hard problem, and often the worst case of the complete methods are prohibitive for practical applications. In these cases, approximate algorithms are preferred.

One often sacrifices the optimality of a given outcome for computational and memory efficiency. These will require very little local computation and communication.

There are two classes of approximate algorithms for addressing DCOP's that the paper will focus on: local Greedy algorithms and Generalized Distributed Law (GDL) based algorithms. [7]

Greedy Approximate DCOP Algorithms: In such a local Greedy algorithm, one begins with a random assignment for all variables. Then based on a series of greedy local moves will seek to optimize the global function. A local move will consist of gathering local information and then changing one or more local variable, local being defined as members of a neighborhood.

The paper focuses on two algorithms of such kind: DSA and MGM. [3]

DSA, or the Distributed Stochastic Algorithm, uses the immediate payoff for its target function, and uses the argmax function for a decision rule. As an adjustment schedule, it uses a random schedule, where each agent has some probability p of actually changing their strategy at any time step. Our implementation of DSA is seen in Algorithm 2.

MGM, or Maximum Gain Message, like DSA also uses immediate payoff and argmax function. The algorithms differ by the adjustment schedule. MGM uses a schedule that gives preference to agents that can achieve the greatest gains. MGM is shown in Algorithm 3.

GDL based Algorithms Generalized Distributive Law (GDL) is a unifying framework to preform inference on graph models. It is a generalized message passing algorithm formed from the synthesis of information theory, digital communications, signal processing, and other works. The GDL algorithm that this paper will focus on is Max-Sum.

Max-Sum: This is an iterative message passing algorithm. Agents will continually exchange messages to build up local functions that depend only on variables that they control. Again, Max-Sum can handle larger domains and more than one variable per agent, however we will only describe the binary and single variable domain. We have included a description in Algorithm 4.

The Max-Sum technique is guaranteed to solve the DCOP optimally on acyclic structures, however, if applied to general graphs there is no guarantee. However empirically, despite this lack of convergence max-sum does generally give a good approximation on cyclic graphs. [2]

3 Experimental Design

This section discusses how we have modified Distributed Constraint Networks to approximately solve influence maximization.

3.1 Specification of the Problem

This paper does not (directly) solve the original problem of influence maximization as posed by Kleinberg[5]. It differs in the following ways.

EXPECTED INFLUENCE. Given a fully connected undirected graph $G = (V, E)$, and an influence nodes exert on each other $i_{u,v}$ for all $v, u \in V$, our model assumes that regardless of the state of node w, $i_{u,v}$ will remain the same for all $v, u \neq w \in V$. We go through an example of this phenomena in the Appendix 8.4

PROFIT. In the original influence maximization problem, the task was to choose the k most influential nodes. In this problem, we are instead solving the problem of influencing such that we maximize a profit, our objective function 2.2. Intuitively, we receive value from influencing nodes, and lose value based on the cost of the nodes that we choose as initial influencers. For an example, please see Appendix 8.5

This approach is different than the one used in much of the comparative influence maximization literature. Much of the literature assumes an *a priori* budget. The goal is to choose the set of maximal influencing nodes under this budget. That problem is analogous to allocating a certain amount of money for advertisement, *and then* trying to choose agents that are effective to advertise to. The present formulation is analogous to spending and receiving money at the same time and always trying to remain in the black.

3.2 DISTRIBUTED CONSTRAINT NETWORK INTEGRATION

Now that we've explored the differences between Kleinberg's notion of influence maximization and the one being addressed in this paper, we must examine how this problem can be translated into a distributed constraint optimization problem.

In our setup, we want to model nodes as being either selected as initial influencers or not, and so the node variables will be binary. Given this, we're interested in transforming the input into a Distributed Constraint Network problem that we can approximately solve using standard techniques. There are two kinds of Distributed Constraint Networks that we can immediately adapt the influence maximization problem into: the one-step model and the multi-step model. The one-step model is a simple but computationally very tractable model that only captures a small bit of the network influence problem, while the multi-step model is a more complex model, built off of the one-step model, that much more accurately models the influence maximization problem. The kind of model that we choose determines the definition of our global and constraint functions.

Both models begin with a graph $G = (V, E)$. Every edge $(u, v) \in E$ is assigned with some weight $p_{u,v} \in [0, 1]$, which is the known influence that u and v have on each other. Each node has a cost c_v for all $v \in V$. We will take this formulation and create a constraint network.

3.2.1 ONE-STEP MODEL

The one-step model simply captures the information dissemination after one step of propagation. This model assumes that you only ever have one chance of infecting your neighbors. For this model, an edge with propagation probability $p_{u,v}$ is worth $p_{u,v}$ if it is between an activated node and an unactivated node. If it is between two activated nodes, it is worth nothing, since we choose to use these nodes as initial influencers and therefore do not receive benefit from them influencing each other. An edge between two unactivated nodes is obviously worth nothing, since we are unable to influence either of these nodes in this case.

These constraint values model the revenue of the graph, but do not yet model the cost. How do we model the cost? The key insight here is that we can introduce a self-edge on every node. This self-edge represents the cost of turning a node on. Importantly, this cost will materialize only for nodes that are on. For each of these nodes, this cost will represent the cost of turning that node on.

Thus, in this one-step model, we define two different kinds of constraint functions. For each node v with initial influencer cost c_v, the self-edge constraint function is therefore defined as follows:

$$f(v,v) = \begin{cases} \begin{array}{c|cc} & v=0 & v=1 \\ v=0 & 0 & 0 \\ v=1 & 0 & c_v \end{array} \end{cases}$$

For each neighbor u of node v with influence transmission likelihood $p_{u,v}$, we have that the edge constraint function is defined as follows:

$$f(u,v) = \begin{cases} \begin{array}{c|cc} & u=0 & u=1 \\ v=0 & 0 & p_{v,u} \\ v=1 & p_{v,u} & 0 \end{array} \end{cases}$$

Note that above, because the graph is undirected, we have used the fact that $p_{v\to u} = p_{u\to v} = p_{v,u}$.

This setup allows us to maximize the profit that we make, because the distributed constraint solver will attempt to find an optimal configuration of nodes that are on such that we maximize profit by turning on highly influential nodes that have lower cost. However, if there is a cluster of highly influential nodes, we would not want to turn all of these nodes on, as turning each successive node on will reduce the value derived from the other activated nodes connected to it.

3.2.2 Multi-step Model

The one-step model is an obvious simplification to the real problem of influence maximization. The one-step model is incapable of modeling the influence that one node has on any node other than its neighbors. The model will preform better than only using the degree of the node, but still lacks needed complexity.

In order to get around this limitation, we realize that we can introduce new edges with associated constraint functions that span beyond the immediate neighbors of a node. Intuitively, for a node $u \notin \mathcal{N}(v)$, there is some nonzero expected influence $p_{v\to u}$ that v exerts on u if and only if there exists a path in the network between v and u. We wish to introduce a new edge into the graph between v and u that exists to model the influence that v bears on u.

Unfortunately, computing the expected influence that agent v bears on agent u is NP-hard. In fact, it is even NP-hard to compute the expected influence that agent v bears on agent $w \in \mathcal{N}(a)$[4]. This is because the number of paths that exist between v and w may be exponential in the graph size when considering that certain intermediary nodes may or may not become infected by v. Thankfully, however, there exists a good stochastic approximation to determine $p_{v\to u}$, the expected influence that initial influencer v bears on any other agent u when we allow no other initial influencers. This algorithm simulates a number of independent cascade trials with v as an initial influencer, and computes the fraction of times that u is influenced. The full algorithm is presented in the appendix as Algorithm 5. As shown in Kempe 2011[4], this algorithm can approximate the actual influence of the agent within an arbitrarily small fraction of the actual value a high probability of the time.

With this, for each node, we compute the influence probability that it bears on all other nodes in the graph. For every connected component in the graph, we compute the expected influence that each node in the component exerts on each other node. For agent v, we introduce an edge to agent $u \in \text{Reachable}(v)$, with the constraint function

$$f(v,u) = \begin{cases} \begin{array}{c|cc} & u=0 & u=1 \\ v=0 & 0 & p^e_{v,u} \\ v=1 & p^e_{v,u} & 0 \end{array} \end{cases},$$

where $p^e_{v,u}$ is the expected influence approximated by the above stochastic Sample Influence algorithm, seen in Algorithm 5 in the Appendix. In the multi-step model, we again introduce the self-edges introduced in the one-step model that reflect the costs associated with selecting each agent.

Moreover, the graph in practice need not be fully complete. With an exponential decrease in influence occurring as distance increases, it would be safe to assume that we could localize this edge creation process using message passing in a distributed system. For a more in depth look into this model, see Appendix 8.7. To walk through a process of creating this graph, see Appendix 8.8.

4 Comparison with Existing Models

Before discussing the results of our algorithm, it's worth discussing how its performance can be measured against existing measures of influence maximization in the independent cascade model. Very little of the existing literature discusses this profit-based model, and much is related to the 'canonical' problem of selecting the k most influential nodes in the graph. As explained in the Introduction, there is a $\left(\frac{e-1}{2e}\right)$-approximation algorithm to this problem.

It would be convenient to be able to express the algorithm presented in this paper in terms of the 'canonical' influence maximization problem for independent cascade. In order to do this, we consider the output of the algorithm presented in this paper. For a graph with agent costs, this algorithm will return the profit maximizing set of initial influencers — that is, the largest set of influencers that we can select such that our marginal return is greater than our marginal cost (consider our objective function 2.2). Therefore, if this algorithm selects q influencers, these influencers are the q influencers that the algorithm thinks are most influential, given their costs. If all of the agent costs are identical, then this is the set of q influencers that the algorithm thinks are most influential in the graph. Therefore, if we set $k = q$ on a graph with uniform agent costs, we are answering the 'canonical' influence maximization problem of selecting the k most influential nodes in the graph.

In fact, we can answer the 'canonical' influence maximization problem for any value of k. We allow every agent cost to be α. By increasing or decreasing α, we raise or lower the cost of selecting each agent. As α increases, the number of chosen initial influencers will monotonically decrease in expectation. Therefore, by varying α, we can cause this algorithm to select any k desired number of initial influencers.

Once we've selected a value of k, it's simple to determine the relative performance of these two algorithms. Simply generate the k influencers from the approximation algorithm and from this distributed constraint optimization algorithm, and then simulate iterations of information dissemination using these two different sets of initial influencers. Averaging over the number of nodes influenced from each initial set of influencers can be used to determine which approach is more effective.

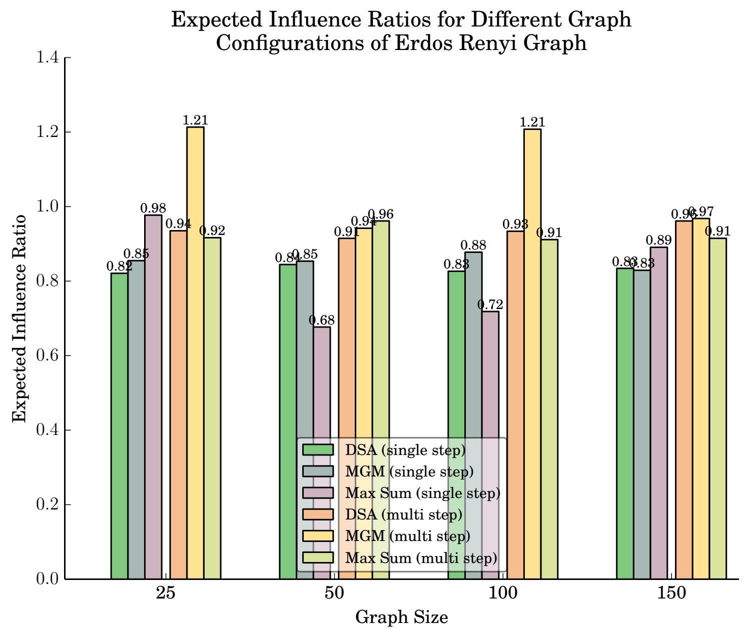
5 Results and Analysis

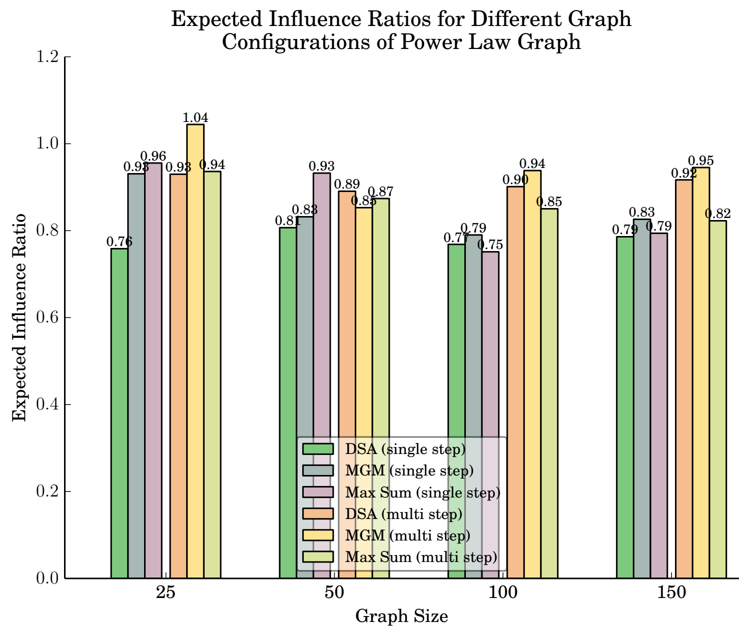
We ran our different graph algorithms on different randomly generated graphs with drastically different degree distributions. We generated results on random graphs of sizes 25, 50, 100, and 150 and one well-known real-world social network graph called the "Karate Club Graph", which had 32 nodes. We would have tried larger graphs, but simulating expected influence took a prohibitive amount of system resources for larger graphs. For each trial and each constraint solver, we used the same graph. We ran all three constraint solvers for 15 steps on each graph for five trials and collected averages of the results.

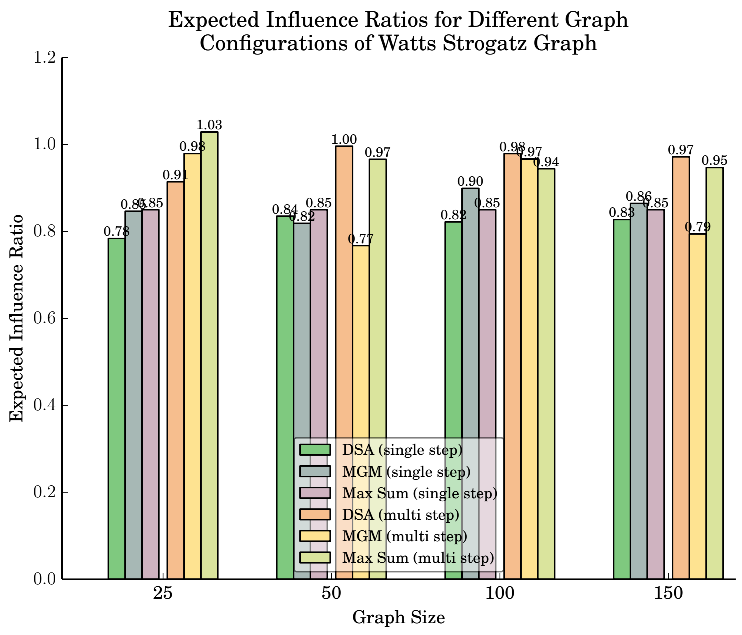
For each of these graphs, we first generate a graph and transform it using either the single- or multi-step model. We then run the Distributed Stochastic Algorithm, Max Gain Message, or Max Sum constraint solvers with sufficient uniform agent weights to select $\sqrt{n}$ initial influencers in the graph, where n is the graph size.

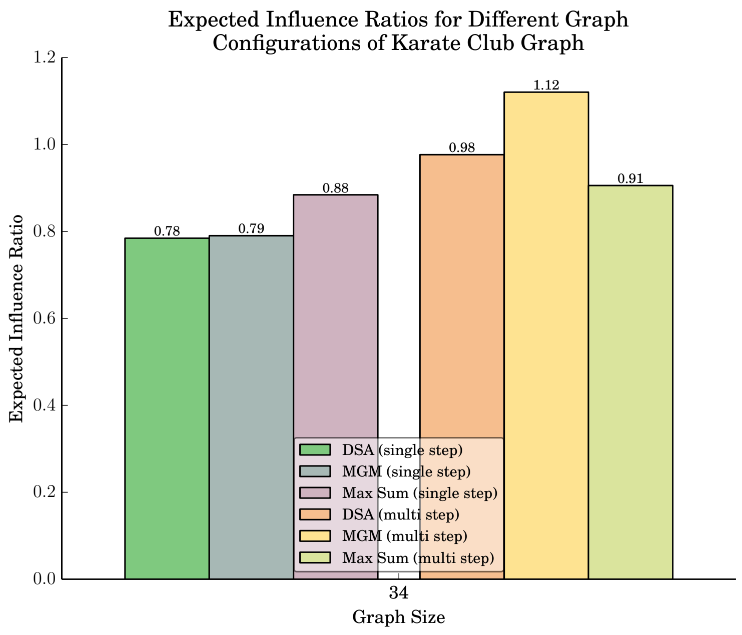
We now have the initial influencers produced from our constraint solving algorithm, called the *experimental influencers*. We also produce the *baseline influencers*, which are the set of influencers derived from the baseline approximation algorithm. With each of these two sets of influencers, we simulate the influence on the graph until the influence propagation stagnates (i.e., the influence frontier on the graph is empty) for a number of trials equal to the graph size. We take the average influence value on this graph and consider that to be the approximate influence induced on the graph from each set of influencers. We can take the ratio of the experimental influence to the baseline influence to comparatively measure the effectiveness of different algorithms. A value greater than 1 indicates that our algorithm performed better, while a value less than 1 indicates that the baseline approximation algorithm performed better.

We ran this procedure on an Erdos-Renyi graph with node degree 4, a Power Law-distributed graph, a Watts-Strogatz graph with 4 neighbors and rewiring probability of 0.1, and the Karate Club graph. The results from these approaches are displayed in the graphs below.







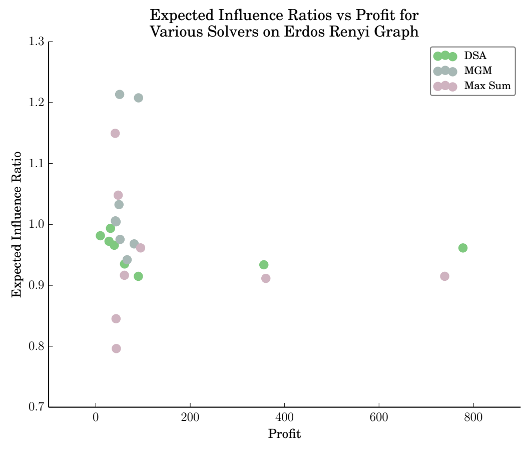


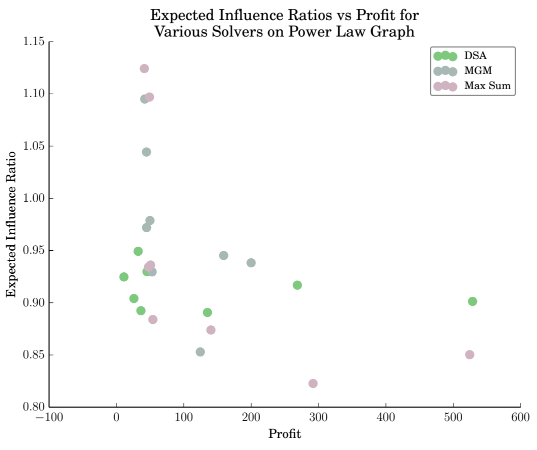
From these graphs, we observe a few interesting trends. Across all graphs, the single-step models tend to perform worse than the multi-step models. This makes sense, since the single-step models are working over more localized information than the multi-step models. We would expect that this means that the multi-step models would be better able to predict hubs of influence, whereas the single-step models are only doing a little more than choosing nodes with high degree. Nonetheless, the performance of the single-step models is not terrible, which is interesting. This is in accord with our prior observation that influence falls off exponentially with distance, and therefore the local neighborhood of influence is a good proxy for total influence. In scenarios where nailing down the high-scoring influencers isn't essential, this simpler model may be desirable because of its reduced complexity.

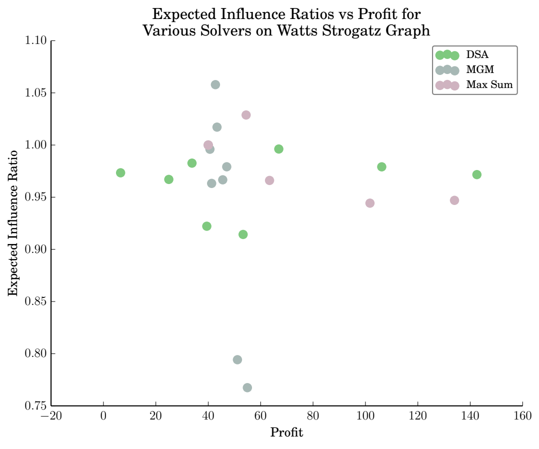
As for the different algorithm types, we observe different results across the different graph types. Overall, max sum tends to do poorly and MGM tends to perform the best. However, there are many scenarios where this phenomenon is reversed. It appears for Erdos-Renyi and Power Law graphs, MGM is a relatively safe bet, as it oftentimes is the best performer, or is close to the best performer. Comparatively, on the Watts-Strogatz model, it performs very poorly, and DSA or Max Sum may be more desirable. This is interesting, because the Watts-Strogatz model is conceptually quite different than the power law or Erdos-Renyi graphs. The Watts-Strogatz graph is constructed in a very predictable manner and, compared to the other two graphs, has relatively little randomness in its construction. While it's difficult to pin down why specifically this may lead to one algorithm outperforming another, it appears that the less structured a model is, the better that MGM performs. An interesting auxiliary observation to this is that the Karate Club graph performs best with MGM. This is likely because the Karate Club graph is a social network, and therefore is structurally similar to the Power Law graph.

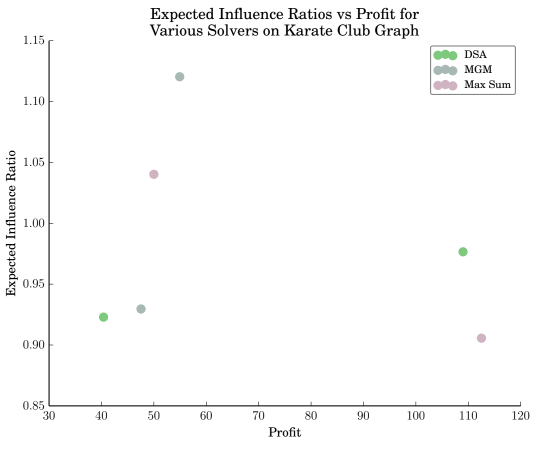
Finally, one somewhat limiting point is that most of these algorithms on all of the graphs do not perform as well as the baseline approximation algorithm. Since the baseline approximation algorithm has a tight bound, it cuts its losses, in some sense. Comparatively, we have no bounds on the performance of our algorithms, and so it's not expected that our algorithms would necessarily perform better than the baseline. It's worth noting that for the Erdos-Renyi graph type, MGM will often beat or very-closely-match the baseline, and so we believe that it would be a desirable substitute for influence computation on graphs of this type. At a higher level, it's also worth noting that most of the time, our algorithms produce results that are at least 90% as good as the baseline. Since there are computational benefits to using a distributed constraint solver, this may be a desirable approach in circumstances where a theoretical lower bound on performance is not obligated.

We have also analyzed the relationship between profit and induced influence. Note that for influence maximization, we are concerned with maximizing influence, whereas distributed constraint problems are themselves concerned with maximizing profit. This is an interesting dichotomy, since this relies on profit being a reliable proxy for influence. We have explained previously why, theoretically, this should be the case. The below graphs present data on this matter. For each of the experimentation trials used above, we were able to obtain a profit value from the graph, which is computed directly from the constraint solver, as well as the influence on the graph, which is computed from the nodes chosen by the constrain solver. We have plotted the influence versus profit below for the different trials.





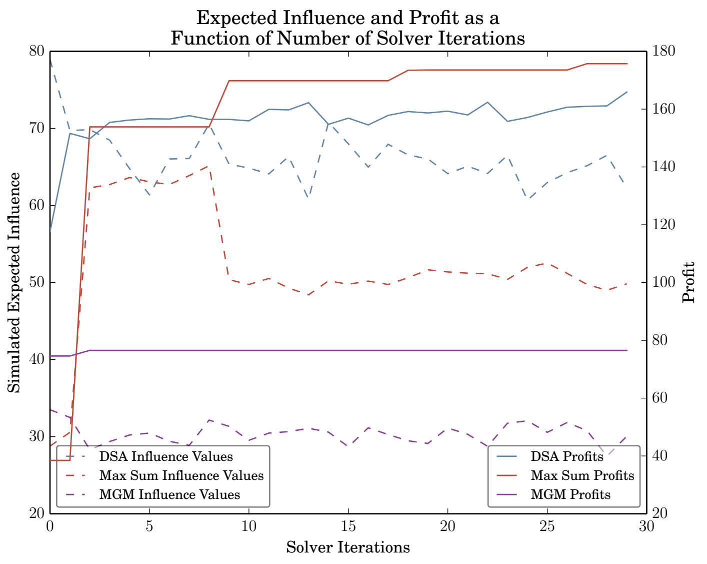




The results here are interesting. We expect profit and expected influence to be positively correlated, since higher profit should induce higher influence on a graph. This does not appear to be the case, and there instead appears to be very little correlation between these two variables. This may be due to a number of reasons, such as that different graph types may have different profit profiles.

It is important to consider how this could be fixed. Ideally, each constraint function value would be computed by simulating the influence of turning on or off a node in the graph. This would eliminate the need for the 'profit' proxy. Unfortunately, simulating graph influence to a reasonable degree of accuracy takes a prohibitively large amount of time, and so this is not possible.

We also looked at how varying the number of constraint solver iterations changed the results of our influence and profit. These results were derived from testing on a Power Law graph of size 50. The computed results are displayed in the plot below.



In this plot, the line colors represent different algorithms, solid lines represent profit at each iteration, and dashed lines represent induced profit. We note that, after the first 10 iterations, profit and influence become essentially uncorrelated. This is encouraging in that it means that our solvers will tend to converge quickly, but discouraging from a theoretical basis. This suggests that profit is a poor proxy for influence, as increasing the number of iterations will increase profit but not influence. This validates the results of the previous set of graphs in confirming that profit may be a poor proxy for influence.

6 Conclusion

We found empirically that using distributed constraint solvers are an effective substitute for the baseline approximation algorithm for approximately computing the optimal initial influencers on an independent cascade model. While our algorithms did not typically perform as good as the baseline approximation algorithm results, they were almost never lower than 90% of the baseline algorithm's results. The key difference between these algorithms, however, is that distributed constraint networks can easily be parallelized using local knowledge, allowing for efficient scalable computation of expected influence on even massive graphs. The baseline approximation algorithm, contrastingly, requires the entire graph to be replicated on a single machine, and is in general a slower algorithm.

It is concerning that the target of our distributed constraint solvers, the profit, does not appear to be strongly correlated with the overall target of our algorithm, the influence. This is likely due to the stochastic nature of the constraint solvers and specific features of different graphs. However, this appears to have relatively little impact on the quality of the algorithm.

7 Future Work

The most pressing of future works is running this in a distributed way. All steps in the process: transformation of the graph (1) and running distributed constraint solvers on the graph (2), can be done in a distributed manner. Thus the next stage in development is testing this claim. We will develop a parralelization of our process and test its tractablility on larger graphs: of sizes one million or more.

With this, we can show that the comprimised preformance can be justified with increased computational tractability.

8 Appendix

8.1 Constraint Network Elaborations

8.1.1 Constraints

Constraints are elements of situations that limit available options and solutions to problems. Constraints by limiting the possible range of solutions, thereby encode information into a system. Thus allowing computationally intractable problems at times to be efficiently solved.

In these problems a set of agents will work together using only local information and message passing in order to jointly achieve a globally optimal solution.

The most natural way to see an example of this behaviour is to consider a scheduling problem, where agents must agree on the time of one or various meetings.

Therefore one can see the global problem of scheduling a company's meeting for the week, the local problem of when to schedule a department's meeting (represented as a neighborhood), and of course the individual problem.

The paper, however, will focus on a very limited domain of constraints, specifically stocastic optimization on the most generalizable form of Multi-Agent Constraint Optimization Problems: Distributed Constraint Optimization Problems. Each of these choices sacrifices optimality, and gains computational efficiency. [8]

8.1.2 Soft and Hard Constraints

Hard constraints enumerate all valid joint assignments of all variables in the scope of the constraint. A soft constraint, C_i^s is a functions F_i, that is defined on the variables $S_i \subseteq X$ which comprise the scope of the function. Thus we can write $F_i = D_i^1 \times \ldots \times D_i^s \to \mathbb{R}$ where $s = |S_i|$. Thus one can imagine how we can encode hard constraints in soft constraints, simply by imposing higher costs to certain functions. Thus we will simply care about soft constraints.

8.1.3 Domains in Constraint Networks

As for domains, while we can represent any number of domains in our network, we will focus on binary domains and thus binary constraint networks. The is fully generalizable seeing that every constraint network can be mapped to a binary contraint network.

8.2 Independent Cascade Approximation

Algorithm 1 Independent Cascade Approximation

```
1: $S \leftarrow \emptyset$
2: for $i \in \{1, 2, \ldots, k\}$ do
3:     $v_i \leftarrow$ the node with the “approximately maximal marginal influence” (within a factor of $1-\epsilon$, achieved with probability $1-\gamma$), as determined by using a polynomial time sampling procedure done in Algorithm 5
4:     $S \leftarrow S \cup \{v_i\}$
5: end for
```

8.3 Approximate DCOP Algorithms

Algorithm 2 DSA: Distributed Stochastic Algorithm

```
1: Initialize each variable, $x_i \in X$ to random $d \in D_i$
2: for Each Activation Step do
3:     for Each Agent $A_i$ do
4:         Choose activation probability $p_i \in [0,1]$
5:         Generate some random number $r_i \in [0,1)$
6:         if $r_i < p_i$ then
7:             Choose new value $a_i$ such that local gain is maximized
8:             $x_i \leftarrow a_i$
9:         end if
10:    end for
11: end for
```

Algorithm 3 MGM: Maximum Gain Message Algorithm

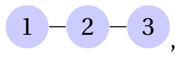
```
1: Initialize each variable, $x_i \in X$ to random $d \in D_i$
2: for Each Activation Step do
3:     for Each Agent $A_i$ do
4:         Collects Gains and Values from neighbors
5:         Choose a values $a_i^*$ such that local gain, $g_i^*$, based on current neighbor values is maximized
6:         if $g_i^*$ is greatest in neighborhood then
7:             $x_i \leftarrow a_i^*$
8:         end if
9:     end for
10: end for
```

Algorithm 4 Max-Sum

```
1: Initialize each variable, $x_i \in X$ to random $d \in D_i$
2: Initialize all messages, $m_{i \to j}$, to constant functions
3: for Each Activation Step do
4:     for Each Agent $A_i$ do
5:         for Each Agent $A_j \neq A_i$ do
6:             Compose a message:
                   $$m_{i \to j}(x_j) = \alpha_{ij} + \max_{x_i} \left( F_{ij}(x_i, x_j) + \sum_{k \in N(i) \neq j} m_{k \to i}(x_i) \right)$$
   Where $\alpha_{ij}$ is a normalization constant, $N$ is the neighbor function.
7:         end for
8:         Compute that local function $z_i(x_i)$:
                   $$z_i(x_i) = \sum_{k \in N(i)} m_{k \to i}(x_i)$$
9:         Find the argument max value given the local function:
                   $$x_i^* = arg \max_{x_i} z_i(x_i)$$
10:    end for
11: end for
```

8.4 Expected Influence

In order to make this problem compatible with Distributed Constraint Networks, we are considering the specific problem of maximizing the expected influence. The general problem posed in Kleinberg involved maximizing influence in the sense that if we have a graph that looks like the following,

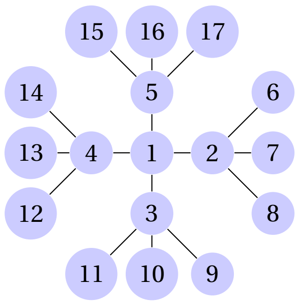


then if we choose to use agent 1 as an influencer and fail to influence agent 2, then we would be unable to influence agent 3. In our formulation of the problem, we alleviate this constraint, and say that 3 can be influenced by 1 with diminishing returns. For concreteness, let's say that each edge will materialize with probability 1/2 if one of their vertices is influenced. The revenue in the above model from choosing 1 as an influencer is $\frac{1}{2}v_2 + \frac{1}{4}v_3$ (where v_i is the revenue from the ith node). Note that maximizing this value is the same as maximizing the expected influence, which is what most modern literature is concerned with, and therefore we feel that this formulation of the problem it not a particularly unfulfilling simplification.

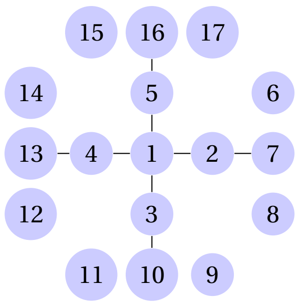
In this formulation, once an agent has been chosen as an initial influencer, revenue is never derived from it. In this way, choosing nodes reduces the amount of revenue available in the graph, and therefore this influence function is submodular.

8.5 Profit Maximization

This is much more easily shown by example than explanation. Consider that we have the following graph.



Let's assume that each node has a cost that is 1 times its number of edges. (Values in this problem are always normalized such that the value of a node that would certainly be influenced is 1.) What would the profit be from using agent 1 as an initial influencer? It would cost 4, but would earn $4 \cdot \frac{1}{2} + 12 \cdot \frac{1}{4} = 5$ (assuming that we did not choose any other agents as initial influencers), and so we would choose it. Now let's consider this modified version of the above graph.



In this case, choosing agent 1 as an initial influencer would only net a revenue of $4 \cdot \frac{1}{2} + 4 \cdot \frac{1}{4} = 3$, an so we would not choose node 1. Instead, we may consider choosing node 4, which has a cost of 2 but a revenue of $2 \cdot \frac{1}{2} + 3 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} = 2.125$.

Note that no value is obtained from "influencing" the initial influencer nodes.

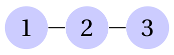
8.6 Stochastic Influence Approximation

Algorithm 5 Sample Influence

```
1: Input: Graph $G = (V, E)$, with edge probabilities $\{p_{u,v}\}_{(u,v)\in E}$, target node $v$, and sample limit $m \in \mathbb{N}$
2: for $i \in \{1, 2, \ldots, m\}$ do
3:     $E' \leftarrow \emptyset$
4:     for $e \in E$ do
5:         Add $e$ to $E'$ with probability $p_{u,v}$
6:     end for
7:     $r_i \leftarrow$ the number of nodes reachable from $v$ in $G' = (V, E')$
8: end for
9: Return $\frac{1}{m}\sum_{i=1}^{m} r_i$
```

8.7 Multi Step Model

Note that this model assumes that each node influences the graph as an initial influencer independently of the other initial influencers in the graph. Clearly this is not desirable. For instance, consider the following graph:



This algorithm will introduce an edge from agent 1 to agent 3 that represents the nonzero bidirectional influence that these nodes exert on one another. However, if we choose node 2 as an initial infleuncer, then nodes 1 and 3 exert no influence on one another. Unfortunately, this is a restriction of Distributed Constraint Networks: they are unable to model constraint functions that are dependent on values other than the nodes that they are connecting. However, since influence decays at an exponential rate in the distance from the influencer, we believe the empirical consequence of this shortcoming to be small.

8.8 Sample Transformation

First we will visualize a transformation of the graph. Consider the below graph **??**:

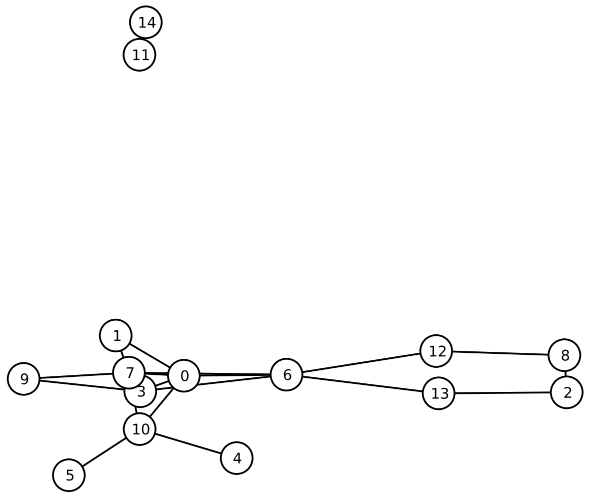


Figure 8.1: Original Powerlaw Graph

It is a power law graph (it could be any graph) with weighted edges and costs for each node encoded on the graph itself. Our first step in our process is to make the graph complete. Below is the complete graph.

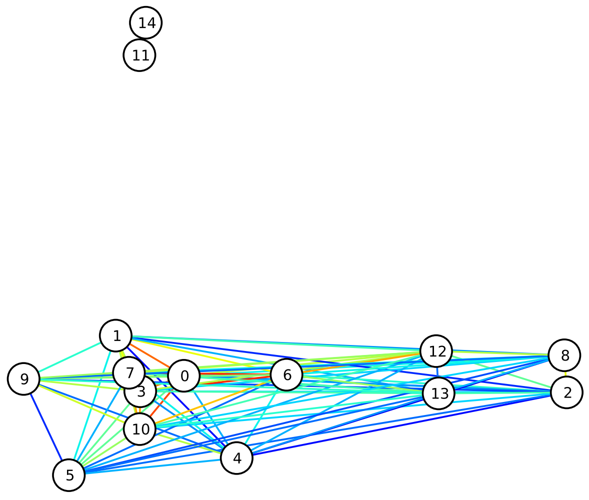


Figure 8.2: Complete Powerlaw Graph

Here there are edges between each node (as long as they were connected on the original graph). The edges are still weighted and we have shown the new weights with color. The heavier the weights the redder, and the lighter the weights the bluer. We will use this graph to get a feel for how influence moves across the graph. We will make this graph into a constraint network (not visualized) and use constraint solvers to find the optimal nodes in the graph. Below is the DSA graph.

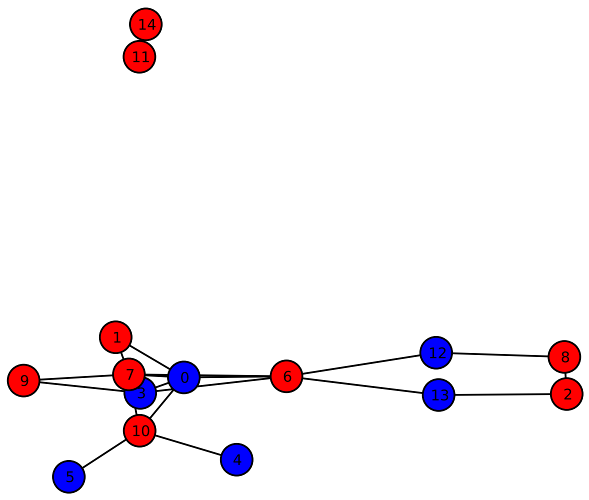


Figure 8.3: DSA sovled graph

This is the first solver. This is DSA (Distributed Stochastic Algorithm). The blue nodes are selected and the red are not. There are a lot of very cool properties that we have mentioned in the paper and will fully flush out later. But this is fully generalizable to any graph, edge weight, and cost for nodes. This instead of having a flimsy assumption of wanted to convert all nodes on the graph, instead wants to make a profit (make the greatest profit).

REFERENCES

[1] A. Chechetka and K. Sycara. No-commitment branch and bound search for distributed constraint optimization. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1427–1429. ACM, 2006.

[2] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 639–646. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[3] R. Junges and A. L. Bazzan. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 599–606. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[4] D. Kempe. Structure and dynamics of information in networks. In *Lecture Notes*, 2011.

[5] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD ’03, pages 137–146, New York, NY, USA, 2003. ACM.

[6] D. Kempe, J. Kleinberg, and É. Tardos. Influential nodes in a diffusion model for social networks. In *Automata, languages and programming*, pages 1127–1138. Springer, 2005.

[7] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 133–140. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[8] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1):149–180, 2005.

[9] Y. Singer. How to win friends and influence people, truthfully: influence maximization mechanisms for social networks. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 733–742. ACM, 2012.

[10] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering, IEEE Transactions on*, 10(5):673–685, 1998.