1 Abstract

Roni says: ***I rewrote some parts of the abstract, and taken a lot from our recent SoCS paper.***

The Target Oriented Network Intelligence Collection (TONIC) problem is the problem of finding profiles in a social network that contain information about a given target via automated crawling. Such profiles are called *leads*. Two best-first search frameworks are proposed for solving TONIC and several heuristics are proposed for each framework. Based on the assumption that leads tend to cluster together, the first proposed framework, called the Basic TONIC Framework (BTF), limits the search for new leads only to immediate neighbors of the leads previously found. The second proposed framework, called the Extended TONIC Framework (ETF), relaxes this limitation and extend the scope of the search to a wider neighborhood, including the possibility of crawling to profiles that are not leads. We analyze experimentally the pros and cons of the two frameworks on two large social networks: DBLP co-authorship graph and the popular Google+ social network. The results show that ETF can reach more leads and in some cases even faster, but BTF might be preferred in cases where the reward of finding a lead is moderate and the time to available budget for finding leads is large. Our results also provide insight into the maximal distance that exists between two leads of the same target. **Roni says:** ***Does the last sentence make sense? I mean to hint here towards the "tiers" table***

2 Introduction

Web-based *online social networks* (OSNs) such as Facebook, Twitter, and Google+ are a part of everyday life for many people around the world. These OSNs are a source of personal information about their users and even contain sensitive commercial or security related information. Commercial companies, government agencies and even individual people often utilize this abundance of data to extract information about a given person of interest. For example, it is common practice for most commercial companies to inspect the Facebook and LinkedIn profiles of candidate employees in order to extract information about past projects, colleagues and managers. Government agencies may also explore OSNs looking for information on terrorists and organized crime.

Information about a given person of interest, denoted as the *target*, can be automatically collected from its OSN profile using information extraction (IE) techniques [**?**]. However, awareness to privacy issues and to data leakage cause many users to tighten their OSN privacy settings limiting access to their profiles. This paper addresses the case where the target's profile is inaccessible to third parties and therefore information about the target cannot be collected from its profile.

In such cases, an alternative way to collect information about the target is through other profiles such as the target's OSN friends. **Roni says:** ***English question: you can either use "target" as a name, and then you can say "... as target's OSN friends" or you can use "target" as noun, in which case it should be "... as THE target's OSN friends". Both are OK, but need to be consistent.*** While the target can easily modify its own privacy settings, it is prohibitively

challenging to cause other OSN profiles to conceal information about the target that they may expose. For example, a OSN profile of a friend of the target may contain photos of the target, public posts made by the target or posts about the target.

Since the target's profile itself is inaccessible, collecting information about the target consists of two tasks. The first task is to find OSN profiles that potentially contain information about the target. Such profiles are called *leads*. The second task is to analyze these leads and extract information about the target, if such information exists. The second task, referred to in this paper as *profile acquisition*, can be performed using standard IE and web scrapping techniques [**?**, **?**, **?**]. In this paper we focus on optimizing the first task, i.e., finding leads while minimizing web crawling costs. We call this problem the Target Oriented Network Intelligence Collection (TONIC).

We solve TONIC with Artificial Intelligence techniques, formalizing it as a heuristic search problem and using a best-first search approach. Two frameworks are proposed for solving TONIC, the Basic TONIC Framework (BTF) and the Extended TONIC Framework (ETF). BTF focuses the search on known leads and their neighbors, while ETF also guides the search through the extended social cycles of a target, by allowing exploration of profiles that are not leads (called "non-leads"). As a generic middle ground between BTF and ETF, we proposed ETF(n), in which non-leads are only explored if they are n steps from a known leads. Thus BTF=ETF(0) and ETF= ETF(∞). Experimentally, we evaluate the extent to which non-leads located farther from the target enable finding new leads, showing that ETF(n) for $n > 1$ is not worthwhile.

Several intelligent heuristics are given for both BTF and ETF. These heuristics analyze the currently known subgraph of the OSN to guide the

search. Importantly, all investigated heuristics are based solely on the topology of the OSN being crawled, and are thus orthogonal to the details of the profile acquisitions. The proposed heuristics and frameworks are evaluated on two large social networks: the DBLP co-authorship graph and the popular Google+ OSN. Results show that ETF(1) is able to finds more leads than BTF and can even find them faster using the presented heuristics.

Finally, we analyze the tradeoff between the cost of searching and the benefit of finding more leads. The results of this analysis is that ETF(1), with the proposed heuristics, is better than BTF for short searches where the reward for finding a lead is high; while BTF, with a proper heuristic, is better suited for long term search process with moderate rewards.

This paper contains material from two previously published conference papers [**?**, **?**]. In addition to providing a comprehensive and unified view, this paper also contains additional experiments on the DBLP co-authorship network and deeper analysis of **Roni: todo: any thoughts what to write here? says: .**

3 Problem Definition

The basic entity in TONIC is the *profile*, which is associated with a specific OSN (e.g., Facebook or Google+). In a given TONIC problem, a specific single profile is defined to be the target profile, simply denoted as $target$. Profiles are connected to each other via a "friendship" relation. The *list of "friends"* (LOF) of a given profile p is denoted by LOF(p).

Roni says: ***As we get grief for this every time we submit this work, I now propose the simplest approach, which also fits what we did in our experiments. The original is commented here***

Definition 1 (Lead and Non-Lead) *A profile p is called a* lead *if $p \in LOF(target)$ and a* non-lead *otherwise.*

The above definition of a lead is a simple way to capture the notion of a profile that is likely to contain information about target. The methods proposed in this work can also apply for more sophisticated definitions of a "lead", e.g., checking if the profile and target are tagged in the same photo or commented on each other statuses.

Information about a profile can be extracted from an OSN in several ways.

- **Scrapping.** Parsing the web pages that are exposed by the OSN services, such as the time-line in Facebook.
- **OSN Application programming interface (API).** APIs exposed by the OSN provide convenient way to collect information about profiles, including their LOF. A prominent example of OSN API is the Facebook Query Language (FQL).

In general, IE from OSN profiles is an active field of research [**?**, **?**, inter alia]. In this work we assume that the IE aspects are encapsulated in two possible OSN queries:

- ***IsLead().*** This query checks if a given profile is a lead or a non-lead.
- ***Acquire().*** This query extracts publicly available data from the OSN about a given profile, including its LOF.

Before applying an *IsLead()* query to a profile p, the problem solver may not know if p is a lead or a non-lead. We refer to such profiles as *potential leads*. Once a profile is acquired (i.e,. an *Acquire()* query was applied to it),

its LOF is extracted. These profiles may either be previously known leads or non-leads, or newly discovered potential leads.

Roni says: ***Consider adding here the life cycle of a profile.***

The task in a TONIC problem is to find *leads* by searching through the OSN graph using the above *IsLead()* and *Acquire()* queries. The input to TONIC is *target* and a set of *initial leads*. These are profiles that are known a-priori to be leads based on previous knowledge about the target. Such a-priori knowledge can, for example, be found manually using traditional intelligence techniques. Since they are very hard to obtain, we assume that there are only a few initial leads per target. The initial leads provide a starting point for the search. The expected output of TONIC is a set of leads discovered during the search.

Executing the *Acquire()* and *IsLead()* queries incurs costs in terms of both computational resources and network activity. In addition, most OSN services limit automatic web scrapping attempts as well as massive exploitation of their API. In this work, we assume that the cost of the *IsLead()* and *Acquire()* queries are equal for all profiles, and that a profile can only be acquired if an *IsLead()* query was applied to it first.

The TONIC problem is defined as follows:

Definition 2 (The TONIC Problem) *Given a profile target, a set of initial leads and a budget b, the TONIC problem is to find and acquire maximum leads while performing at most b queries.*

Other variants of the TONIC problem can also be formulated. For example, finding a fixed number of leads with minimum acquisitions, or an "anytime" variant, where an end-user may stop the search for leads unexpectedly. The algorithm and heuristics proposed in this paper are expected

to be effective for all these variants.

4 TONIC As Search in an Unknown Graph Problem

We model TONIC as a search problem in an unknown graph [**?**]. These are graph search problems where *expanding a node* requires some external action, and the goal is to minimize the number of node expansions (as oppose to runtime). The searched graph $G = (V, E)$ is the topology of the OSN, i.e., V is a set of OSN profiles and E is a set of link such that $(v_1, v_2) \in E$ if $v_1 \in LOF(v_2)$. *Expanding* a node corresponds to acquiring a profile, which reveals its LOF. Following standard search terminology, the profiles in the discovered part of the OSN can be partitioned to:

- **Expanded:** profiles that were acquired
- **Generated:** potential leads

Previous work on searching in an unknown graph [**?**] proposed a best-first search (BFS) approach. Algorithm 20 outlines a similar search framework specifically designed for TONIC.

Algorithm 20 maintains the following data structures:

1. L, NL and PL, which are the sets of leads, non leads and potential leads found so far, respectively.
2. The *currently known subgraph* of G (denoted CKG), containing all nodes (profiles) and edges (friendship relations between profiles) found so far (line 1)

Algorithm 1: BFS for TONIC

Input: *target* the target profile

Input: b, the max total queries allowed

Input: *InitialLeads*, the set of initial leads

Output: L, the leads found

$L \leftarrow InitialLeads, PL \leftarrow \emptyset, NL \leftarrow \emptyset$, $CKG \leftarrow$ the subgraph of G induced by L

foreach *l in InitialLeads* **do**
    LOF $\leftarrow$ *Acquire(l)*
    Add LOF to $OPEN$ and to PL

while $OPEN \neq \emptyset$ *AND* $b > 0$ **do**
    best $\leftarrow$ ChooseBest($OPEN$)
    if *best is a potential lead* **then**
        bestislead $\leftarrow$*IsLead(best)* `/* Dec(b) */`
        $PL \leftarrow PL \setminus \{best\}$
        if *bestislead= True* **then**
            Add *best* to L
            LOF $\leftarrow$ *Acquire(best)* `/* Dec(b) */`
            Update CGK and PL with found edges and potential leads
        else
            Add *best* to NL
            Reinsert *best* to OPEN
    else
        LOF $\leftarrow$*Acquire(best)* `/* Dec(b) */`
        Update CGK and PL with found edges and potential leads

return L

3. $OPEN$, a list of nodes representing profiles considered for either an *IsLead()* or an *Acquire()* query.

Initially, all the initial leads are expanded and OPEN is seeded with the potential leads from their LOFs. In every iteration, a single profile (*best*) is chosen from OPEN (line 6).

If *best* is a potential lead, then an *IsLead(best)* query is performed (line 8) and *best* and *best* is removed from PL. If *best* is found to be a lead, then it is immediately acquired, as finding leads is always desirable (line 12). CKG is updated with the newly found edges and nodes and $OPEN$ is updated with newly found potential leads (line 13). Otherwise, it is reinserted to $OPEN$ to be considered for acquisition at a later stage (line 16). Thus, $OPEN$ may contain potential leads that are considered for an *IsLead()* query, or non-leads considered for an *Acquire()* query. If *best* is a non-lead, then it is acquired (line 18), updating CKG and $OPEN$ with the resulting information (line 19). After the budget of allowed queries is exhausted, or all the nodes in the OSN has been acquired, the set of found leads is returned (line 20).

Figure 1 presents an example execution of Algorithm 1. The target is marked by T, and there is one initial leads a. After acquiring a we reveal b and c as potential leads. In the next step, the *IsLead()* is performed on c revealing that it is a non lead. Then, the *IsLead()* is performed on b revealing that it is a lead. b is then immediately acquired, discovering the potential lead d. **Roni says:** ***Maybe we want to add here also a stage where a non-lead is acquired. Not critical***

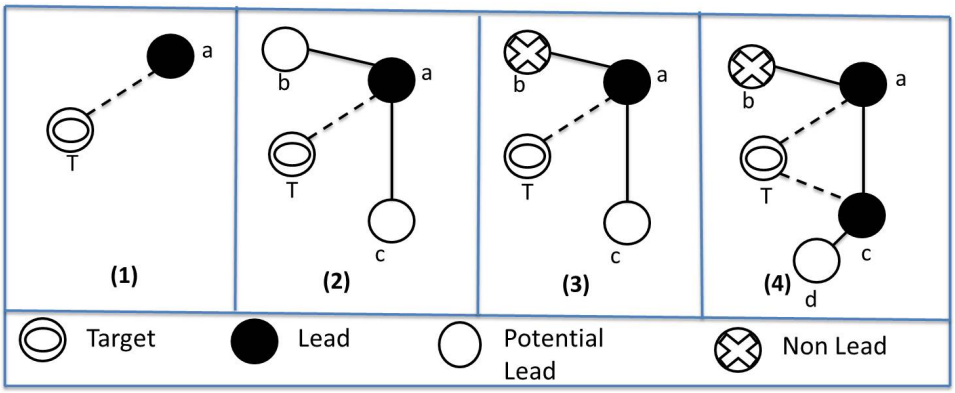

Figure 1: An example of a run of Algorithm 1.

5 The Basic TONIC Framework (BTF)

One clear limitation of Algorithm 1 is that in the worst case, it will acquire all the profiles in the OSN. This can be a serious limitation, as popular OSNs are very large. Moreover, the set of possible profiles to acquire may grow to be very large, and include many profiles that are very unlikely to be leads. Previous work showed that OSNs and social networks in general follow the homophily principle, which means that friends tend to exhibit similar attributes [**?**, **?**, **?**]. Thus, it follows that that friends of leads (i.e., profiles that appear in leads' LOFs), are more likely to be leads than randomly selected OSN profiles.

The Basic TONIC Framework (BTF) builds on this understanding, and focuses the search for leads by only applying the *Acquire()* query to profiles that are known to be leads. BTF simplifies Algorithm 1, as profiles that are found to be non-lead are discarded and are not re-inserted into $OPEN$ (line 16). As a result, $OPEN$ only contains potential leads and lines 18-19 are redundant. Also, a single *expand* action can be defined, which performs

IsLead() on *best* and acquires if it is a lead. **Roni says:** ***Last sentence is helpful or not? what motivated me in adding this is that we earlier say that expand=acquire. Thoughts? we can write everything below without mentioning the term "expand" but I'm not sure if this won't hurt readability***

5.1 BTF Heuristics

Key to the efficiency of BTF is to choose intelligently which potential lead to expand in every iteration of Algorithm 1 (line 6). Next, we present several heuristics designed for this purpose. We refer to these heuristics as *BTF heuristics*. As a baseline BTF heuristic, consider expanding profiles in a *first-in-first-out* (FIFO) manner. This means that the neighbors of the initial leads are chosen first, then their neighbors and hence forth.

Figure 2 illustrates such an expansion order. T is the target profile and there is a single initial lead a. After profile a is acquired, two profiles are discovered, b and c. Then, they are expanded according to the order of their insertion into $OPEN$.

Another baseline is to randomly choose which potential lead to expand next. These baselines are denoted *FIFO* and *RND*. Both baselines do not consider the CKG (the currently known subgraph of the OSN). As the search progresses, the CKG grows, containing more information about the searched OSN. The following heuristics are based on analyzing the CKG in different ways.

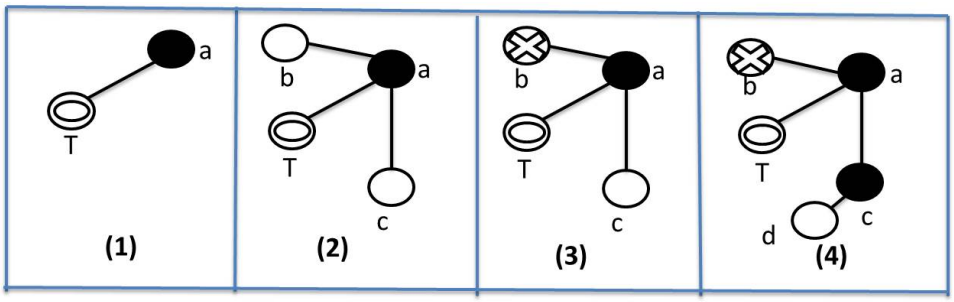

Figure 2: An example the basic FIFO heuristic.

5.1.1 Clustering Coefficient

Nodes in OSNs tend to form tightly connected clusters in which most peoples are friends of each other. This phenomenon is quantified using the notion of *local clustering coefficient* [**?**]. The local clustering coefficient of a node in a graph is the density of edges between its neighbors. Formally, it is the proportion of links between the nodes within its neighborhood divided by the number of links that could possibly exist between them. **Roni says:** ***Maybe the next sentence and formula is redundant. Thoughts?*** Let $CC(n_i)$ be the local clustering coefficient of node n_i, and let N_i be the neighborhood of n_i, then

$$CC(n_i) = \frac{2|\{e_{jk} : n_j, n_k \in N_i, e_{jk} \in E\}|}{|N_i|(|N_i| - 1)}.$$

In TONIC, a profile that is connected to a cluster of leads is likely to be part of that cluster and thus likely to also be a lead. An intuitive example of such a case is a small university department where a member of that department is the target. It seems reasonable that members of that department form a dense cluster in the OSN and are likely to be leads. Therefore, as more leads are found in that cluster, profiles in it will have higher local clustering coefficient and it would be worthwhile to expand them

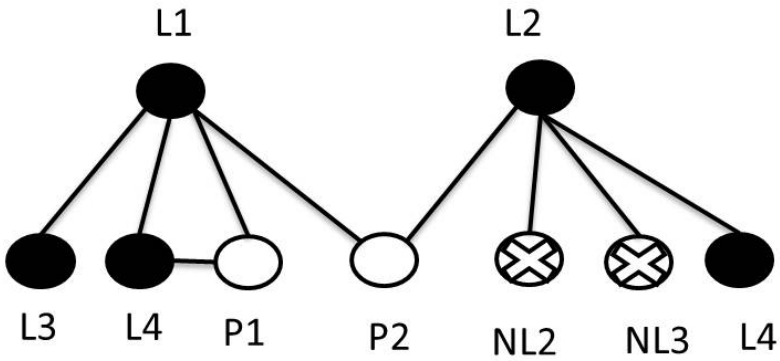

Figure 3: Example of the CC Heuristic.

because they would be more likely to be leads. **Roni says:** ***Not sure if this is convincing***

Building on this intuition we propose a heuristic that is based on computing the local clustering coefficient (CC) of each of the potential leads and choosing to expand the potential lead with the highest CC. More formally, let $L(pl)$ be the set of leads that are friends of the potential lead pl in the CKG. The CC of pl in the CKG is the number of links between $L(pl)$ divided by the number of possible links among $L(pl)$:

$$CC(pl) = \frac{2 \cdot |\{(u,v) \in E_{CKG} | u,v \in L(pl)\}|}{|L(pl)| \cdot (|L(pl)| - 1)}$$

In BTF, the neighborhood of a potential lead consists of only the previously acquired leads, and thus $L(pl) = N(pl)$ and the above formula is exactly the standard local clustering coefficient given earlier. The *CC heuristic* is the heuristic that chooses to expand the potential lead with the highest CC.

An example of CC heuristic is provided in Figure 3. White nodes are potential leads, black nodes are leads, and white nodes with X are non-leads. $CC(P1) = 1$, since the number of links between $L(P1)$ is 1 (the link between

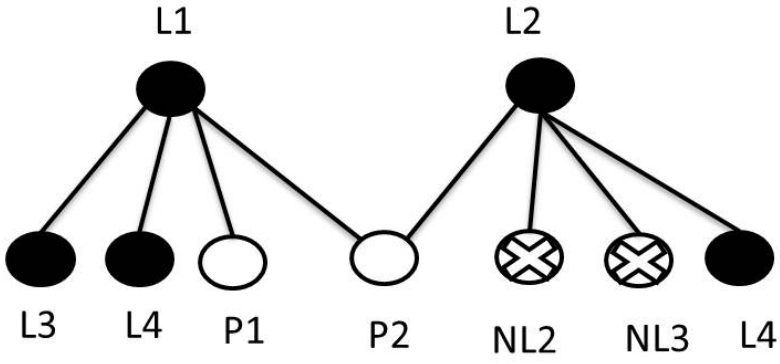

Figure 4: Example of the KD Heuristic.

L1 and L4) out of one possible link among $L(P1)$. $CC(P2) = 0$, since there are no links between $L(P2)$. Thus, the CC heuristic will choose to expand node $P1$ before node $P2$.

5.1.2 Degree of Potential Leads

The CC heuristic ignores the number of friends that a potential lead has. In fact, potential leads with a single friend will have the highest CC score (one). This is a disadvantage of CC, because it has been shown that degree of nodes in a social networks exhibits a power-law distribution [**?**], and previous work has shown that an effective strategy for searching in such graphs is to direct the search towards nodes with a high degree [**?**]. **Roni says:** ***Rami – should the relevant network characteristic here be "preferential attachment" and not "power law"?*** The intuition behind this is that high degree nodes are connected to many other nodes, and thus are better sources for searching than low degree nodes.

Identifying the potential lead with the highest degree, or is connected to the most leads, is not possible in TONIC, since the problem solver does not

know the true degree of potential leads before they are acquired. However, the degree of a potential lead in the CKG *is* known. This is called the *known degree* (KD) of a node and the corresponding heuristic, denoted as the *KD heuristic*, expands nodes in the order of their KD. This heuristic was previously used to find cliques in unknown graphs [**?**]. In BTF, the KD of a potential lead pl is the number of acquired leads that are friends of pl (denoted earlier as $L(pl)$), since only leads are acquired. An example of KD is provided in Figure 4. $KD(P1) = 1$ (connected to $L1$) while $KD(P2) = 2$ (connected to $L1$ and $L2$). Thus, the KD heuristic will choose to expand node $P2$ before node $P1$.

5.1.3 Promising-Leads Heuristic

The KD heuristic expands potential leads according to the number of acquired leads that are connected to them. This is reasonable if all leads have an equivalent effect on the likelihood that a potential lead connected to them is a lead. Consider the example presented in Fig 5. P1 and P2 are potential leads, both connected to one lead. P1 is connected to a lead with 3 lead friends while P2 is connected to a lead with 3 non lead friends. We believe that P1 is more likely to be a lead than P2 since it is connected to a more "promising" lead. Following, we explore an alternative approach that considers not just the amount of leads that a potential lead is connected to, as the KD heuristic, but also how "promising" these leads are **Roni says:** ***Minor: in Latex, instead of using "bla bla", use "bla bla". It looks differently in the output and some reviewers are nudnikim. Please fix throughout the paper.***,**Liron says:** ***Done*** in the sense that potential leads connected to them are more likely to be leads.

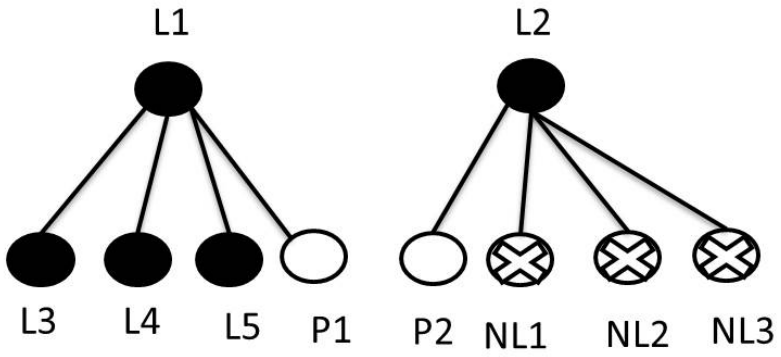

Figure 5: Motivation for the Promising heuristic.

The first step in creating such a heuristic is to define a measure of how "promising" a lead is. An ideal "promising" measure for a lead m would be the probability that a randomly drawn generated neighbor of m is a lead. This is the ratio of potential leads connected to m that are leads. We denote this ideal promising measure as $pm^*(m)$.

Unfortunately, $pm^*(m)$ cannot be known before *all* neighbors of m are acquired. As a practical alternative, we consider the ratio of leads among the *expanded* neighbors of m. Formally, we divided the LOF of an expanded lead m into three sets: leads, not leads and potential leads, denoted as $L(m)$, $NL(m)$ and $PL(m)$, respectively. The promising measure we propose, called the *promising factor* and denoted by $pf()$, is computed by $pf(m) = \frac{L(m)}{L(m)+NL(m)}$.[1]

[1]Initially, it is possible that $L(m) + NL(m) = 0$, making $pf(m)$ undefined. To avoid this, we set $pf(m) = 0.5$ in this case.

5.1.4 Aggregating Promising Leads

If every potential lead was connected to a single expanded lead, then a straightforward TONIC heuristic that considers the promising factor would expand the potential lead that is a friend of the expanded lead with the highest promising factor. However, a potential lead may be connected to a set of expanded leads, each having a different promising factor.

Two simple ways to aggregate the promising factors of the leads is to take their maximum or their average. We call the corresponding TONIC heuristics $MaxP$ and $AvgP$, respectively. Formally, $MaxP$ chooses to expand the potential lead pl that maximizes $\max_{m \in L(pl)} pf(m)$, while $AvgP$ chooses to expand the potential lead pl that maximizes $\frac{1}{|L(pl)|} \sum_{m \in L(pl)} pf(m)$. As an example of these aggregation methods, consider again the graph in Figure 6. There are two potential leads, $P1$ and $P2$. The only lead connected to $P1$ is $L1$, which has $pf(L1) = \frac{3}{3} = 1$. $PL2$ is connected to two leads, $L2$ and $L3$, with $pf(L2) = pf(L3) = \frac{1}{3}$. According to the $MaxP$ heuristic, $MaxP(P1) = 1$ and $MaxP(P1) = \frac{1}{3}$. Thus, P1 will be expanded first. According to the $AvgP$ heuristic, $AvgP(P1) = 1$ and $AvgP(P1) = \frac{1}{3}$. Thus, in the $AvgP$ P1 will also be expanded first.

$MaxP$ only considers the lead that is most promising, and ignores all the other leads in $L(pl)$. $AvgP$ takes into consideration all the leads in $L(pl)$ but may diminish the effect of a very promising lead in $L(pl)$, if $L(pl)$ contains other less promising leads. Next, we consider a more sophisticated way to aggregate the promising factors of the leads.

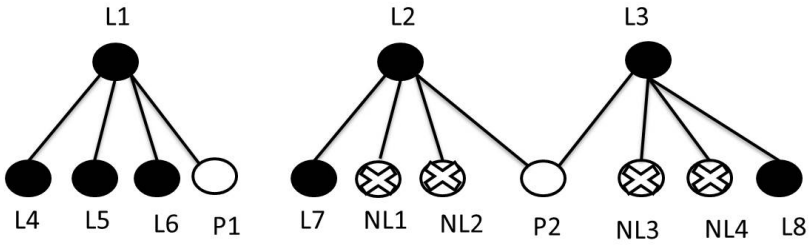

Figure 6: Example of the Promising Heuristic.

5.1.5 Bayesian Aggregation

The promising factor $pf(m)$ is designed to estimate $pm^*(m)$, which is the probability that a potential lead connected to m is a lead. We therefore propose another way to aggregate the promising factors that is based on a Naïve Bayes approach to aggregate probabilities.

$$BysP(pl) = 1 - \prod_{m \in L(pl)} (1 - pf(m))$$

The TONIC heuristic that chooses to expand the potential lead pl with the highest $BysP(pl)$ is denoted as the *Bayesian Promising* heuristic, or simply $BysP$. $BysP$ has the following desirable attributes. Unlike the $AvgP$, discovering a new lead m that is connected to pl is guaranteed to increase (or at least not decrease) $BysP(pl)$, since $pf(m) \leq 1$. Unlike $MaxP$, any change in the promising factor of each of the leads in $L(pl)$ affects the $BysP(pl)$: it will increase or decrease according to the increase or decrease of the promising factor of the leads in $L(pl)$.

As an example of the Bayesian Promising heuristic, consider again the graph in Figure **??**. As mentioned earlier, there are two potential leads, $P1$ and $P2$. The only lead connected to $P1$ is $L1$, which has $pf(L1) = \frac{3}{3} = 1$. $PL2$ is connected to two leads, $L2$ and $L3$, with $pf(L2) = pf(L3) =$

$\frac{1}{3}$. Thus $BysP(P1) = 1 - (1 - pf(L1)) = 1$. $BysP(P2) = 1 - (1 - pf(L2))(1 - pf(L3)) = \frac{5}{9}$. Therefore, according to the $BysP$ heuristic $P1$ will be expanded first.

5.1.6 Friends Measure (FM)

The TONIC problem bears some resemblance to the *link prediction* problem [**?**], where the goal is to predict whether two profiles are connected. Link prediction algorithms return the likelihood of a link to exist between two profiles. This suggests the possibility of employing a link prediction algorithms for TONIC, by ranking nodes in $OPEN$ according to the likelihood of a link to exist between a node and *target*.[2] In fact, a notion very similar to KD was extensively used in link-prediction research, where it is known as the common-friend concept [**?**]. Simply put, we expect a potential lead that has many friends that are leads to also be a lead.

The *Friends Measure* is a very successful link prediction method that estimates the likelihood of a connection between two profiles by counting the number of common friends and the number of links between the friends of the two profiles [**?**]. Formally, the friends measure (fm) between profiles two profiles (u and v) is defined as follows:

$$fm(u,v) = \sum_{x \in N(u)} \sum_{y \in N(v)} \delta(x,y)$$

$$\delta(x,y) = \begin{cases} 1 & \text{if } x = y \text{ or } (x,y) \in E \text{ or } (y,x) \in E \\ 0 & \text{Otherwise} \end{cases} \quad (1)$$

[2] A more comprehensive discussion on the relation between link prediction and TONIC is given in Section 11.

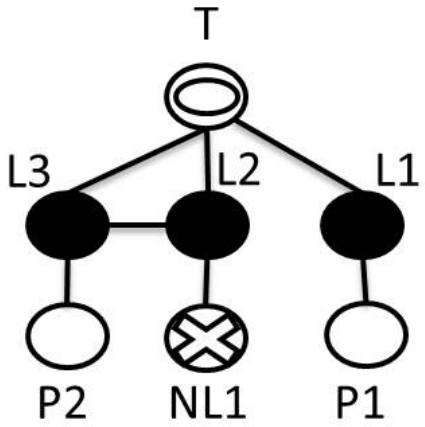

Figure 7: Example of the FM Heuristic.

In BTF, we are interested to predict if a given potential lead pl has a link to $target$ (i.e., to predict if pl is a lead). Computing $fm(pl, target)$ required $N(pl)$ and $N(target)$. $N(pl)$ is not known before pl is acquired. Instead, we used the neighbors of pl in the CKG. As only leads are acquired in BTF, this is exactly $L(pl)$. $N(target)$ is also unknown, otherwise we would know all the leads. Instead, we used the set of known leads (L). The resulting Friends Measure of pl and $target$ is:

$$fm(pl, target) = \sum_{x \in L} \sum_{y | y \in L(pl)} \delta(x, y)$$

Roni says: ***Liron/Rami – please verify the above*** The BTF heuristic that chooses to expand the potential lead pl the highest $fm(pl, target)$ is called *FM*.

Figure 7 provides an example of the FM heuristic. In Figure 7, there are two potential leads P1 and P2. P1 has neighborhood of P1 contains one mutual friend with the target T (L1) and therefore $fm(P1, T) = 1$.

In contrast, P2 also have one mutual friend with the target (L3), however the target's neighborhood and P2's neighborhood's contain and additional mutual friend(L2) and therefore $fm(P2, T) = 2$ Therefore, the FM heuristic will choose to expand node $P2$ before node $P1$.

5.2 Experimental Results

Next, we evaluate the performance of the different BTF heuristics on the Google+ social network. Google+ is one of the largest social networks, having more than 540M registered users and 300M users that are active monthly (according to Wikipedia).

5.2.1 Data Collection

The data set used in our experiments was obtained from the Google+ network and included 211K profiles with 1.5M links between them. This data set was collected by Fire et. al [**?**] and made available at `http://proj.ise.bgu.ac.il/sns/datasets.html`. From this data set we randomly selected a set of 100 profiles having at least 30 friends. These profiles were used as the targets in our experiments.

Since in BTF only potential leads can be acquired, the CKG for each target can contain at most the set of leads and their neighbors throughout the search. We call these set of profiles the *relevant neighborhood* of the target. In our data set, the size of relevant neighborhood ranged from 233 to more than 4,000 profiles. Figure 8 presents the relevant neighborhood of one of the targets in our data set.

The search for leads for each target was executed using BTF using the following heuristics: RND, FIFO, CC, KD, AvgP, MaxP, BysP, and FM.

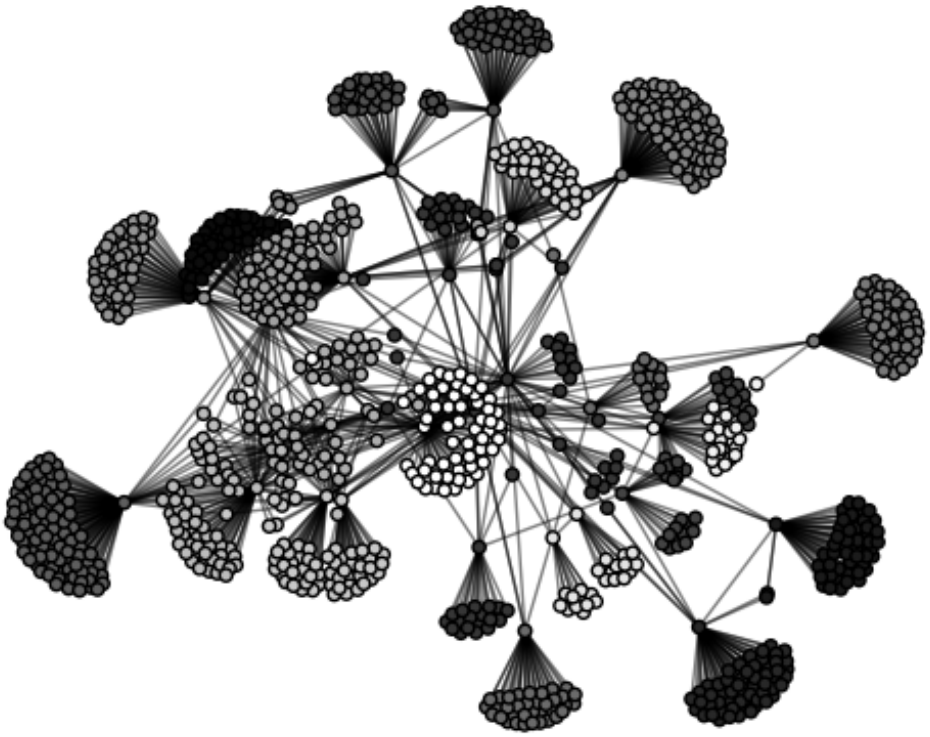

Figure 8: Sample network of the relevant neighborhood of a target.

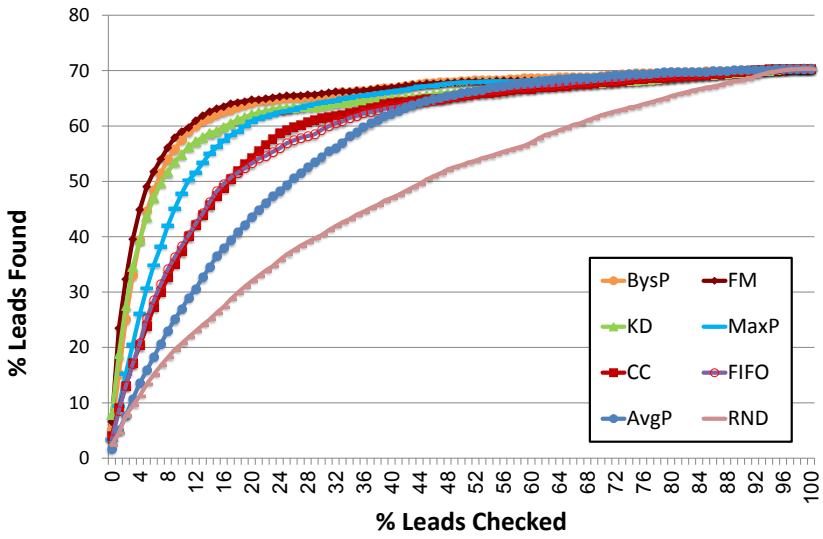

Figure 9: % of leads found vs. % of potential leads checked.

Three friends of every target were randomly chosen as the initial leads. The search continued until *all* potential leads were expanded.

Figure 9 shows the following analysis. The x-axis represents the percent of BTF iterations preformed, where each iteration represents an *IsLead()* query and if necessary, an *Acquire()* query as well.The y-axis represents the fraction of leads found out of all the possible leads, averaged over all the 100 targets in our data set. Note that since BTF limits the search to only acquire leads, not all leads are reachable from the initial leads.

The result in Figure 9 show that that KD, FM and BysP are able to find more leads earlier during the search process and in general outperform all other heuristics. In particular, BysP and FM dominate all other heuristics, and BysP has a slight advantage of FM. **Liron says:** ***Should i still say the***

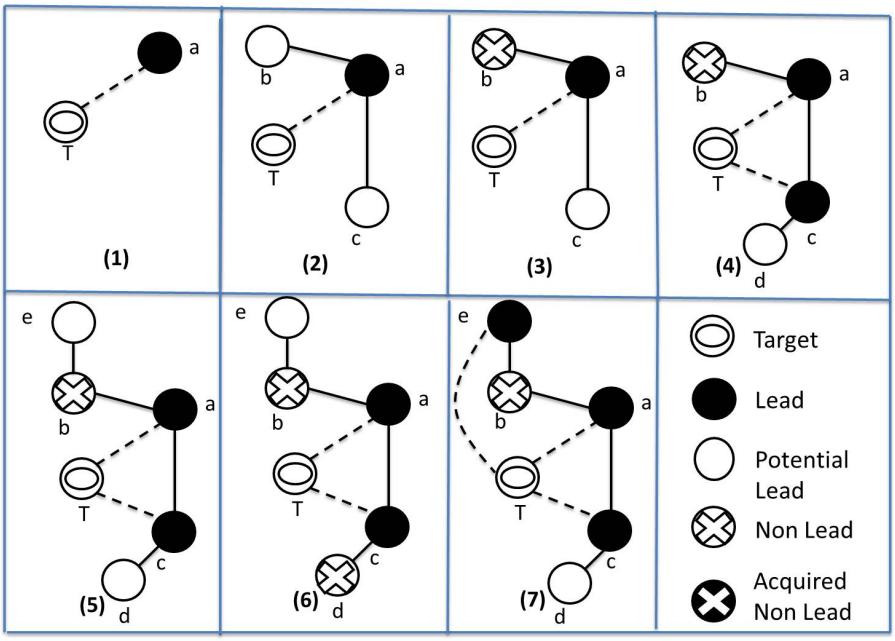

Figure 10: An example of a search for leads.

bysp dominates all other heuristics? bysp and fm are extremely close (avg 64.433 vs 64.292 respectively **Roni says:** ***Rewrote to say that BysP and FM are the winners. Liron – can you check statistical significance in each time step? the numbers above - 64.433 and 64.292 are averages over the entire x-axis, where they looks pretty much the same, but maybe in some time steps the difference is statistically significant***

6 Extended TONIC Framework (ETF)

As seen in the results above, BTF may miss 30% of the leads. In this section, we relax the BTF requirement of acquiring only leads, and present the Extended TONIC Framework (ETF), where non-leads can also be acquired.

tiers	0	1	2	3
% reachable leads	70.51%	88.28%	88.28%	88.60%

Table 1: % of reachable leads from different tiers leads

This enlarges the scope of the search to a larger set of profiles around known leads.

To prevent the search from scattering, ETF limits the non leads that may be acquired to profiles that are at most n edges in the CKG from a known lead, where n is a parameter. ETF(n) denotes ETF with this parameter. Thus, BTF is actually ETF(0), since only leads are acquired, while running Algorithm 1 unrestricted is in fact ETF(∞).

For a given target and initial leads, ETF partitions the OSN to *tiers*, where tier n is the profiles that can be acquired by ETF(n). As n increases, the number of profiles in the corresponding tier grows, and the search will be able to reach more leads. However, larger tiers have also more non-leads, causing potential waste in querying them. To provide insight into how to choose n intelligently, Table 1 shows the percentage of leads that exists in each tier. As can be seen, the exploration of tier 1 profiles can increase the number of leads found substantially compared to tier 0, i.e., in ETF we can find more leads than in BTF. However there is not much difference between the number of leads found in higher tiers (tier 2, 3, and 4). Therefore we focus in the rest of this paper on ETF(1).

A key difference between BTF and ETF is handling of non-leads which can now be added to $OPEN$. In BTF, a potential lead found to be a non-lead is discarded. In ETF(n), discovered non-leads are re-inserted into $OPEN$, to be later considered for expansion if their are not too far from a

known lead. Note that the distance of non-leads to known leads can change as the search progresses and new leads are discovered.

Roni says: ***Figure fig:etf-process-and-legend does not exist*** Figure **??** shows an execution of ETF. Stages 1-4 are similar to the BTF process and were presented in section 4. In stage 5 the non lead c is acquired revealing its LOF ($\{e\}$). *IsLead(e)* is then performed, revealing that it as a lead. This search process can continue, finding and acquiring more and more leads.

As in BTF, the main challenge in ETF(1) is to choose which node to query next in each iteration. Next, we propose several ETF heuristics for this purpose.

6.1 ETF Heuristics

Since the amount of reachable profiles (and reachable non-leads) with ETF is much larger than with BTF, ETF can potentially perform worse than BTF. Thus, the benefit of ETF depends on having an effective heuristic for choosing the best node to expand. In this section we describe several ETF heuristics.

6.1.1 EFIFO

This simple baseline heuristic chooses *best* for expansion in a first-in-first-out (FIFO) order. If a potential lead was chosen as *best* and discovered as a non lead, it is removed from the OPEN and reinserted as non lead at the end of OPEN. Figure **??** illustrates such an expansion order. Figure 11 shows the performance of EFIFO versus that of BysP, the best BTF heuristic. The x-axis is the number of *IsLead()* calls. The y-axis is the number of leads

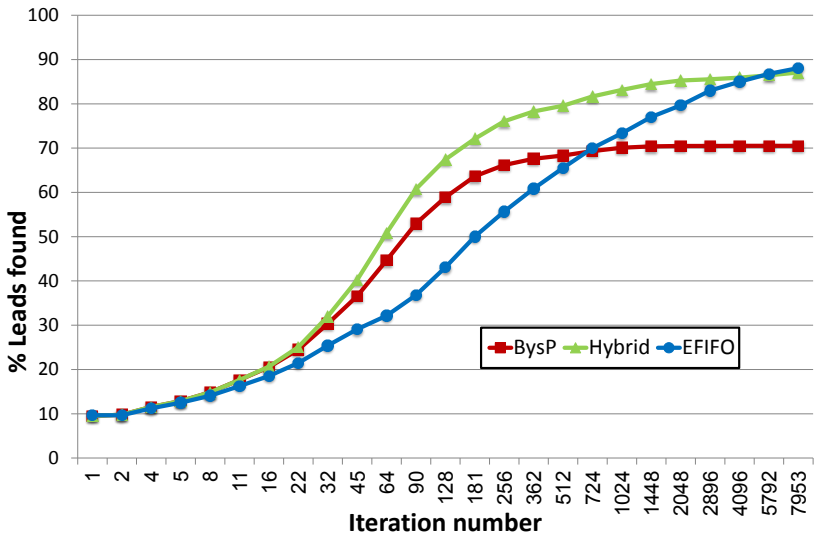

Figure 11: BysP(0), EFIFO(1) and the hybrid heuristic. The x-axis is the is the number of *IsLead()* calls. The y-axis is the number of leads found by *IsLead()* up to that point.

found by *IsLead()* up to that point.[3] As can be seen, EFIFO discovers leads slower than BysP but eventually reaches more leads. 11.

6.1.2 Hybrid Heuristic

To enjoy the complementary benefits of BysP, which finds leads fasts by effectively focusing on clusters of leads, and the extended reachability of EFIFO, we propose an adaptive *hybrid* heuristic that starts the search with BysP and eventually switches to EFIFO. Ideally, we would like to switch as soon as BysP exhausted the set of leads it can reach. To determine the

[3]The exact setting of this experiment is provided below in the experimental section.

exact switching point we define a bound U which determines the number of *IsLead(·)* queries allowed since the last lead was found. If U unsuccessful *IsLead(·)* queries were done by BysP, the *hybrid* heuristic assumes that BysP has discovered the set of leads it can reach and switches to EFIFO.

We have tried many values for U and have found that the best option is to increment U dynamically according to the following assignment schedule. U is initially set to a some constant. Upon acquisition of a lead (with BysP), U is reset to be the number of *IsLead(·)* queries done so far. For example, if a lead was found in the third *IsLead(())* operation, U is now set to 3, meaning that unless a lead is found in the next 3 *IsLead( )*operations, the BysP heuristic will be switched to FIFO. Empirical evaluation (see Figure 11) shows that *hybrid* switches heuristics when EFIFO starts to outperform BysP thus performing as the best of the two throughout the search.

6.1.3 Known Degree Variants

KD, described above for BTF, expands the potential lead with the highest degree in the CKG. Next we discuss how to adapt it to ETF. Let $KD(p)$ be the degree of p in the CKG, and let $KDL(p)$ be the number of leads adjacent to p in the CKG. Since only leads are acquired in BTF, there is at least one lead in every edge of the CKG. Consequently, in BTF $KD(p) = KDL(p)$ for every potential lead. In ETF, $KD(p)$ and $KDL(p)$ can be different, as a potential lead may be connected to leads and to non-leads. This results in two possible ETF heuristics, KD and KDL, each expanding the node (either potential lead or non-lead) in $OPEN$ with the highest $KD(\cdot)$ and $KDL(\cdot)$, respectively.

Figure 12 depicts a CKG that demonstrates the difference between KD

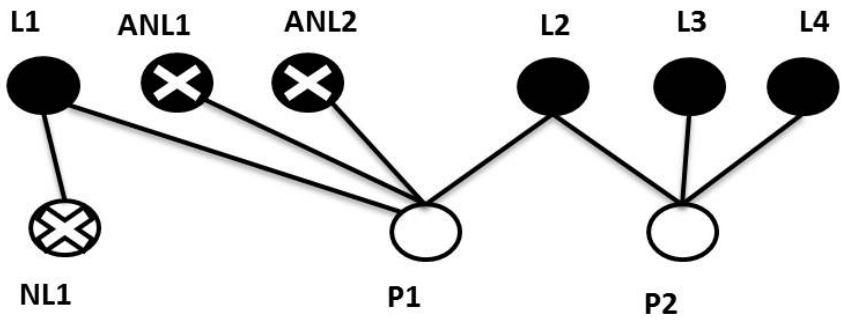

Figure 12: Example for the KD and KDL heuristics.

and KDL. The legend for this figure is the same legend shown in Figure **??**. There are three profiles that can be expanded: NL1, P1 and P2. KD will expand P1 since $KD(P1) = 4$, $KD(NL1) = 1$, and $KD(P2) = 3$, while KDL will expand P2 since $KDL(P2) = 3$, $KDL(P1) = 2$, and $KDL(NL1) = 1$.

The intuition behind KD and KDL differ. KD is based on the assumption that leads tend to cluster together, and thus a profile with many adjacent leads suggests that this profile is itself lead or is adjacent to many other leads. KDL is based on the assumption that a profile with a high degree in the CKG has a high degree in the underlying graph (the OSN), and thus expanding it would result in finding many other profiles, some of which would be leads. Our experiments showed that KD performs significantly better than KDL in ETF(1) (as shown in Figure 15 explained in the experimental section).

6.1.4 BysP and FM for ETF

BysP and FM was the best performing BTF heuristics (see Section 5.2), and they can be easily adapted for ETF. The key difference between BTF and ETF is that non-leads can be acquired, and thus while in BTF neighbors of profiles that are considered for expansion (either *IsLead()* or *Acquire()*)

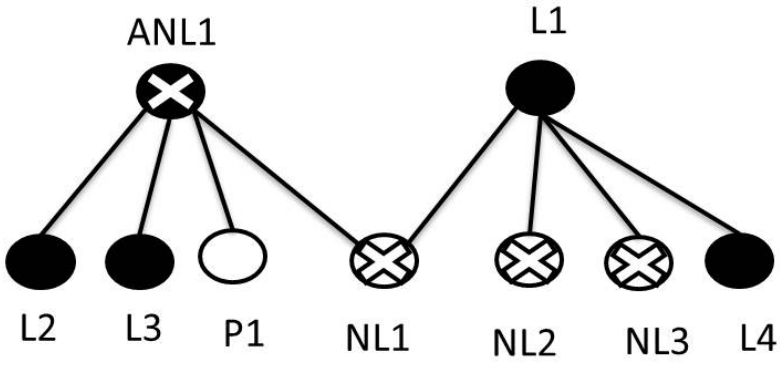

Figure 13: Example of the EBysP heuristic

were leads, in ETF the neighborhood of a profile that was not acquired yet may also contain non-leads. This requires slight modifications to the BysP and FM heuristic computation.

For BysP, the BysP score ($BysP(p)$) of a profile p in ETF would aggregate the promising factor of all its neighbors, regardless if they are leads or not. For FM, the Friends Measure would count links between all neighbors of p, non-leads included, and all the known leads. To distinguish between BysP and FM for BTF and for ETF, we denote the latter EBysP and EFM, respectively.

Note that both EBysP and EFM expand the profile with the highest score in $OPEN$ (each according to its scoring function), regardless if it is a potential lead or a non-lead. If that profile is a potential lead, then an *IsLead()* query is applied to it and it is acquired if it is found to be a lead. If the best profile is a non-lead, then it is acquired.

To illustrate EBysP, consider Figure 13. $OPEN$ contains P1, NL1, NL2, and NL3. These profiles are connected to two acquired profiles, ANL1 and L1, having $pf(\cdot)$ values of $\frac{2}{3}$ and $\frac{1}{4}$, respectively. As a result, P1, NL1, NL2,

and NL have $EBysP$ values of $\frac{2}{3}$, $\frac{9}{12}$, $\frac{1}{4}$, and $\frac{1}{4}$, respectively, and therefore EBysP expands NL1.

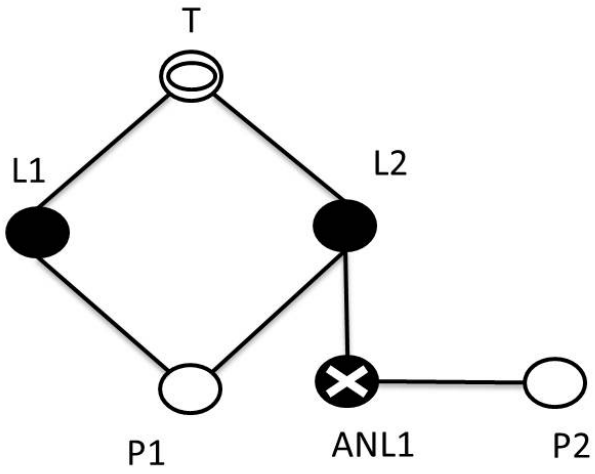

Figure 14: Example of the EFM heuristic.

Figure 14 demonstrates EFM. There are two potential leads P1 and P2. According to the EFM definition presented, $fm(T, P1) = 2$ and $fm(T, P2) = 1$ (ANL1 is in P2's neighborhood and it is friends with L2 which is in the target's neighborhood). Thus, P1 will be acquired prior to P2.

Roni says: ***Need to consider removing both examples. Or, even better, have these examples where we have BysP and EBysP to show the difference, and the same for FM and EFM. Thoughts?***

6.2 Experimental Results

In order to evaluated the performance of ETF(1) with the proposed heuristics we used the same benchmark set of targets and initial leads as was used in the BTF experiments earlier (Section 5.2). However now, the relevant neighborhood of each target is larger since it contains an additional tier.

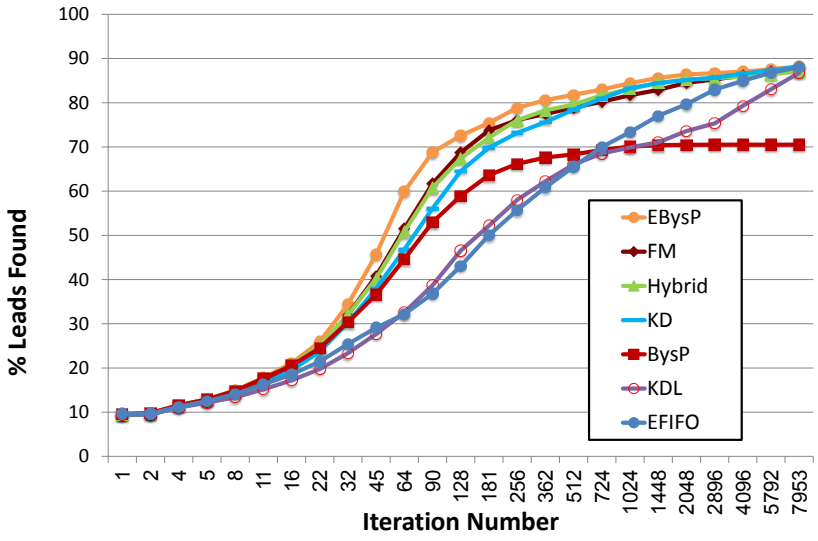

Figure 15: % of leads found vs. iteration number

Figure 16 presents the relevant neighborhood of one of the targets in our data set. **Roni says:** ***Liron. What would be really cool is to show for the same profile how its relevant neighborhood looks in tier 0 and tier 1. Can you do this easily?***

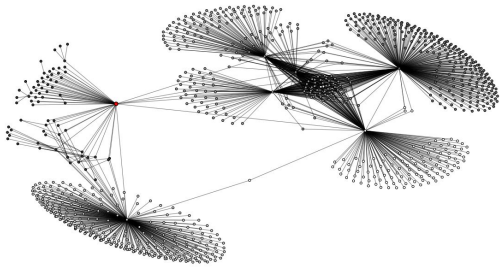

Figure 16: Sample network of the relevant neighborhood of a target

Figure 15 compares the percentage of leads found (the y-axis) out of all the possible leads by each of the proposed ETF heuristics as a function of the iteration percentage (the x-axis). For reference, we also present the results for BysP (which was the best BTF heuristic). At the very beginning of the search, all heuristics perform similarly because the CKG is too small to be informative and the search is almost blind. As the search progresses and the CKG grows, EBysP quickly gets better than other heuristics, finding more leads faster. In particular, EBysP substantially outperforms BysP throughout the search. For example, after 0.8% of the iterations, BysP found slightly more than 40% of the leads, while EBysP found approximately 60%. This result demonstrates that intelligent acquisition of non-leads not only enables reaching more leads eventually (as shown in Table 1), but also significantly speeds up the acquisition of leads during early stages of the search. **Roni says:** ***A more sensible x-axis for this figure would be***

nice. It is not clear if it is logarithmic, of some predefined steps with no logic behind it.

All studied heuristics are computationally efficient as they analyze the neighborhood of the evaluated profile p only up to two hops away. KD is the simplest heuristic that only considers the connectivity of p. Surprisingly, its performance is not a lot worse than the performance of the more sophisticated heuristics EBysP and FM and much better than the performance of KDL which is more focused toward leads.

7 Effects of Network Topology Characteristics

Next we investigate the network topology characteristics that influence the performance of the proposed TONIC heuristics. For that purpose, we will examine the performance of both BTF and ETF heuristics on the DBLP network [**?**] as an additional OSN to the Google+ network previously discussed.

We have taken 100 different targets with more than 30 friends and set 3 random friends as initial leads (same as we did for the Google+ network). An example of tier(1) of one of the targets in DBLP is presented in Figure 17. The search was preformed on four heuristics; The BTF and ETF baselines (FIFO and EFIFO) and the dominating heuristics (BysP and EBysP). We have also limited the run to 10K iteration, thus also simulating the limitation in the number of queries that exist in part of the OSNs.

The results are presented in Figure 18. An interesting observation that can be made from the figure is that the search is divided into three stages. (1) The beginning of the search in which the ETF heuristics preform similar to the BTF heuristic. (2) The second stage is the stage starting from the

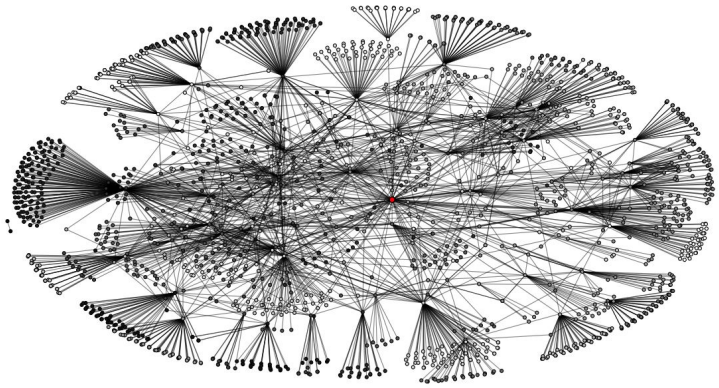

Figure 17: Sample network of the relevant neighborhood of a target in DBLP network number

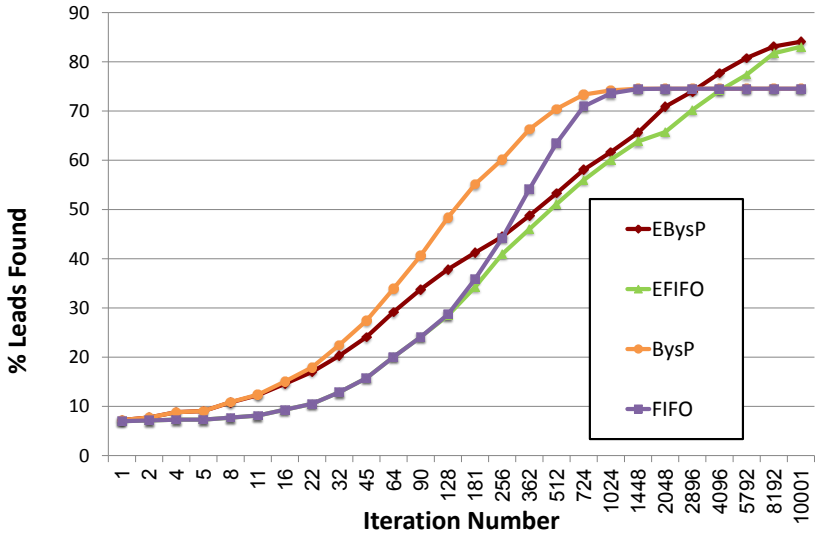

Figure 18: % of leads found vs. iteration in DBLP network number

point in which the ETF and BTF heuristics cease to preform similarly and the BTF heuristics outperform the ETF heuristics. (3) The last stage is the stage in which the BTF heuristics have already found all of their reachable leads and the ETF heuristic continue to find new leads.

At the first stage, the CKG is relatively small and the information obtained on the OSN topology is still insufficient to cause different choices of *best* profiles between the BTF and ETF heuristics.

In the third stage, we can see that the ETF heuristics continue to find leads after the BTF heuristics already found all of their reachable leads. Meaning that the use of ETF heuristics finds more leads than BTF, as can be expected from the Google+ results.

The second stage is the most interesting stage. In this stage, EBysP does not outperform BysP as expected from the Google+ results previously shown.

To understand the differences between the networks, we have experimented with various topology attributes including the *clustering coefficient* (CC) of each target, the size of the *relem"vant neighborhood* of the target, the number of connected components in the graph excluding the target and the density of this neighborhood. The topology characteristic that has the most distinct impact on the proposed heuristics was the number of connected components in the graph(excluding the target).

We have checked the number of connected components in ETF(0)(BTF) and in ETF(1). As expected we have seen a decrease in the number of connected components between ETF(0) and ETF(1) both in Google+ and in DBLP. This decrease is caused since the additional tier in EFT(1) contain profiles that connect several connected components in ETF(0). Meaning, ETF(1) contains non leads that connect several connected components. We

noticed that the decrese was different between the Google+ and the DBLP networks. In Google+ the decreses was larger than in DBLP (40% decrease from 15.94 to 9.3 connected components in Google+ as oppose to 30% decrease from 6.29 to 4.38 connected components in DBLP).

The non leads in ETF(1) can connect not only existing connected components, that were reachable in ETF(0) (from different initial leads), but also new connected components that were not reachable in ETF(0) and might contain leads. In fact, since ETF find more leads than BTF, as can be seen in the third stage, we know that new connected components containing leads were added to the graph in ETF(1) and are now reachable.

Due to this decrease, ETF in general and EbysP in particular have the advantage of searching for leads in a larger variety of connected components. Meaning, that instead of focusing the search in the connected component that existed in ETF(0), the ETF heuristics can search for leads in the additional connected component if a lead is more likely to be found there. This advantage was the reason that EbysP preformed better than BysP even in the early stages of the search in the Google+ network.

However, in DBLP, it is seems that EbysP does not have this advantage. In order to understand why, we have checked the number of reachable leads that were added into the graph in ETF(1). In Google+, 16.53% new leads in average were added as oppose to 11.52% new leads in DBLP. We have also checked the likelihood to find the new added lead by calculating the number of added reachable leads divided by all reachable profiles. The likelihood to find an added lead in Google+ is 0.01 as oppose to 0.007 in DBLP. Meaning that less leads were added in DBLP and that the likelihood of finding them is smaller than in Google+. This relatively small improvement in the number of reachable leads in DBLP, resulting from the relatively small

decrease in the number of connected components, reduces the advantage that the ETF heuristics and EbysP in particular have, since the number of connected components to consider in ETF(1) is relatively similar to the number of connected components in ETF(0). In addition, the difference in the percentage of reachable leads between ETF(0) and ETF(1) is much lowe in DBLP than in Google+. I Google+ the difference in the precentage of leads found is 17.5%(from 70% to 88% reachable leads) as oppose to only 9% (from 75% to 84% reachable leads). From this we conclude that in networks in which BTF finds a relatively large percentage of leads and the improvement in the percentage of reachable leads between ETF(0) and ETF(1) is relatively low, such as DBLP, we would prefer to use BTF. By contrast, in networks such as Google+, in which BTF finds relatively lower percentage of leads and the improvement in the percentage of reachable leads between ETF(0) and ETF(1) is relatively high, we would prefer to use ETF.

8 Cost-Benefit Analysis

The task in TONIC according to Definition 2 is to maximize the number of leads found within a given budget. Next, we consider a different objective for TONIC, where the reward of finding leads is weighted against the cost of finding spent in finding them.

Specifically, consider an application of TONIC where the monetary reward of finding a lead (and passing it to the information extraction phase for further analysis) is known and denoted by R_{Lead} . In addition, the cost of *Acquire()* and *IsLead()* is also estimated in monetary terms, denoted by cost(*Acquire()*) and cost(*IsLead()*), respectively. This allows measuring the

net gain (rewards minus costs) throughout the search. For the following analysis, we assume that the cost of *IsLead()* is negligible compared to the cost of *Acquire()*. Thus, we compute the *net gain* (denoted NG) of a search as the total reward from finding leads minus the total cost of all *Acquire()* queries. **Roni says:** ***Minor: consider replacing*** NG ***with*** $NetGain$***. Why not be clear and relax the cognitive burden of the reader?***

The net gain is related to the steepness of the slopes in Figure 15. A steep slope corresponds to finding many leads with few queries, which means high net gain, while a flat line means costs of queries are spent but no leads are found, which means decreased net gain. A trend that is very clear in Figure 15 is that during late stages of the search the frequency of leads decreases and the slopes become flatter. Therefore, the net gain may drop to a point where the search process is not worthwhile.

Figure 19 demonstrates this, showing the net gain (y-axis) as the search progresses (x-axis, denoted number of *Acquire()* queries executed) for BysP and EBysP. These heuristics were chosen because BysP is the best heuristic for BTF(=ETF(0)) and EBysP is the best heuristic for ETF(1). First, observe that the gain of BysP does not decrease throughout the search. This is because BTF only acquires leads, and thus whenever a cost is spent on *Acquire()* it is immediately followed by a reward (of passing this profile to the information extraction phase). Thus, the gain in BTF cannot decrease unless either cost(*Acquire()*)$> R_{lead}$ or cost(*IsLead()*) is not negligible.

ETF(1) allows acquiring non-leads. When a non lead is acquired, its acquisition is not followed by immediate reward. However, the acquisition of non leads can be viewed as a long-term investment, leading to higher rewards and gain in the future. Indeed, as shown in Figure 19 the acquisition of non leads by EBysP results in much more frequent discovery of

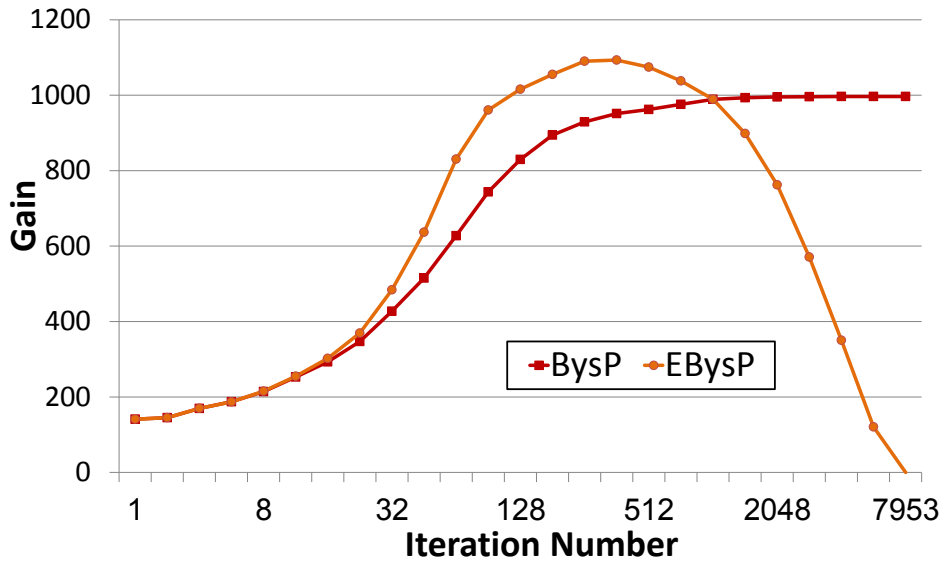

Figure 19: The net gain by BysP and EBysP when R_{lead} costs 40 times more than cost(*Acquire()*).

leads at the first stages of the search and overall higher gain compared to BysP. However, when leads are exhausted, EBysP looses the previously accumulated reward on useless exploration of the network and may eventually reach negative gain. In order to gain the most from EBysP one needs to determine when the search process should be halted. While we leave development of sophisticated stop conditions for future work, we illustrate next the potential gain of having such a mechanism.

Table 2 shows the maximal gain achievable for BysP and EBysP, for different values of R_{lead} and assuming that cost(*Acquire()*)=1. According to Table 2 BysP reaches a maximal gain higher than EBysP when $R_{lead} \leq 3$, and this reverses when $R_{lead} \geq 4$. This is reasonable because larger R_{lead} makes the long term investment of EBysP in acquiring of non leads worthwhile.

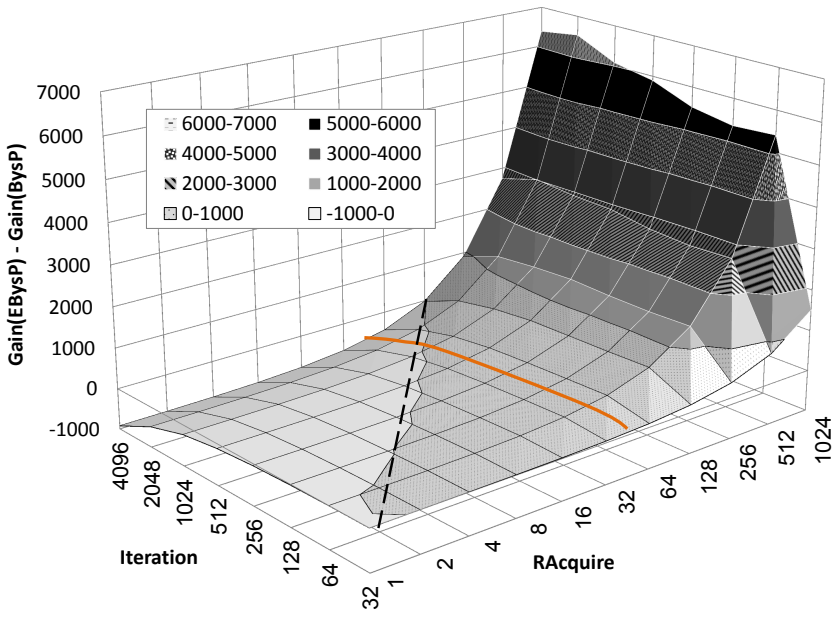

Figure 20: The difference in net gain between BysP(0) and BysP(1)

Reward	BysP	EBysP
100	2529.45	**3100.89**
10	229.95	**250.55**
5	102.20	**104.96**
4	76.65	**76.71**
3	**51.10**	49.17
2	**25.55**	22.50
1	0	0

Table 2: Maximal gain for different $R_{Acquire}$ values

To gain a deeper understanding of the gains of BysP and EBysP throughout the search, Figure 20 shows the difference between the gains of EBysP and BysP (vertical-axis $\uparrow$) as a function of R_{lead} (horizontal-axis $\nearrow$) and the number of iterations (depth-axis $\nwarrow$). For example, the solid line at $R_{lead} = 40$ represents the difference between gains of EBysP and BysP as depicted in Figure 19. The dashed line represents the relation between R_{lead} and iterations where the gains of both heuristics are equal. EBysP benefits from higher rewards and suffers for longer executions. Thus, the choice of between BysP and EBysP could be determined upfront if one knows the number of iterations the search will be run, R_{lead}, and cost(*Acquire()*).

9 TONIC As An Information Retrieval Task

An alternative cost-benefit analysis, often used in Information Retrieval (IR) and other fields, is the well-known receiver operating characteristic (ROC) curve [**?**]. The ROC curve measures the true positive rate (TPR) as a function of the false positive rate (FPR). TONIC can also be viewed as

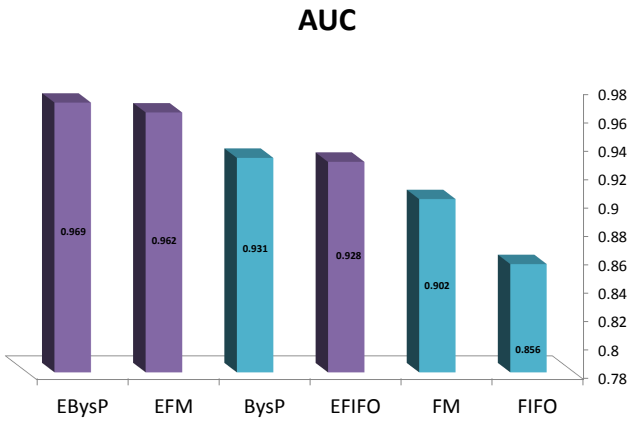

Figure 21: Average AUC

an IR task: the *query* is the target and the set of initial leads, and the *response* to that query is a sequence of profiles acquired up to any specific point during the search. Thus, for TONIC the TPR is the ratio of leads found (the ratio of profiles that the heuristic chose as likely to be leads that were in fact leads) and the FPR is the ratio of non-leads found (the ratio of profiles that the heuristic chose as likely to be leads that were discovered as non leads). In TONIC, most potential leads are not leads, and thus in general the number of non-leads found is very similar to the total number of *IsLead()* preformed. Thus, the ROC curve for TONIC greatly resembles the cost-benefit curve shown in Figure 9 and Figure 15 (this was also observed empirically).

We considered two known IR metrics: the area under the ROC curve (AUC) and the discounted cumulative gain (DCG) [**?**]. AUC is simply the are under the ROC curve described above. DCG provides an alternative

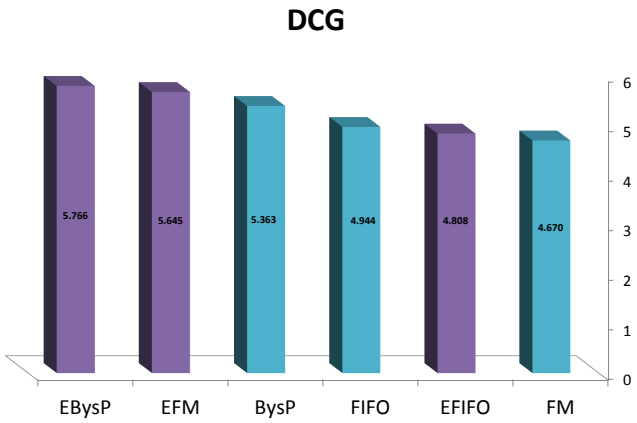

Figure 22: Average DCG

view to AUC, giving higher value to finding leads earlier in the search. It is computed by $DCG = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{log_2 i}$, where rel_i in TONIC is one if the i^{th} profile that the *IsLead()* query was preformed on is a lead or zero otherwise. DCG has special importance in TONIC, as every *IsLead()* query increases the network footprint of the intelligence collection and the possibility that the OSN service will block future acquisitions. Therefore, one can argue that it is important to spot the leads as early in the search process as possible, supporting the DCG metric.

Figure 21 shows the average AUC of the algorithms: FIFO, EFIFO, FM , EFM, BysP and EBysP. The BTF heuristics (run under BTF) are colored in blue and the ETF heuristics (run under ETF(1)) are colored in purple. AUC results show the same trends as shown in Figure 15 and Figure 9, as could be expected. However it is interesting to see that despite the fact that EFIFO eventually finds more leads then BysP, it's average AUC is still

lower, since it preforms worse during most of the search.

Figure 22 shows the average DCG of the algorithms mentioned before.In the DCG analysis, EFIFO also preforms worse than BysP, however it is interesting to see that according to the DCG analysis EFIFO even preforms worse than FIFO. This is due to the fact that FIFO find leads in earlier iterations , since its search is less scattered, which gives it an advantage over EFIFO.

Note that EBysP is in general robust across both AUC and DCG and that both FM and BysP had shown improvement both in AUC and in DCG when implemented in ETF (EFM, EbysP).

10 Effects of the Initial Leads

TONIC receives as input a number of initial leads from which the search begins. As described earlier, obtaining these initial leads may be very costly. However, the set of leads reachable increases as more initial leads are given, thus potentially leading to more leads found and higher net gain. Next, we examine the effect of the number of initial leads on the percentage of reachable leads. Note that since the initial leads are hard to obtain, we kept the number of initial leads in our analysis relatively small.

Table 3 shows the percentage of reachable leads in each tier (tiers 0-3) as a function of the number of initial leads. As can be seen from the table in all tiers, the use of only one initial lead can produce worse results than the use of two or more initial leads. The average difference between the use of one initial lead and the use of two initial leads across all tiers is 4.63%. As the number of initial leads increases, the difference between the percentage of reachable leads decreases. For example the average difference between two

tiers	0	1	2	3
1 initial lead	61.34%	82.78%	82.78%	83.10%
2 initial leads	67.94%	86.75%	86.75%	87.07%
3 initial leads	70.51%	88.28%	88.28%	88.60%
4 initial leads	72.19%	88.52%	88.52%	88.84%
5 initial leads	73.81%	88.82%	88.82%	89.11%

Table 3: % of reachable leads from different number initial leads

and three initial leads is 1.79% and the difference between three and four initial leads is less than 1% (0.6%). As can also be seen, the difference in the percentage of reachable leads between the different initial leads number is higher in BTF(=ETF(0)) than in ETF. Thus, the impact of adding initial leads is more pronounced in BTF (tier 0), while it is relatively small for ETF(1), ETF(2), and ETF(3). For example, the results suggest that unlike BTF, for ETF(1) or higher there is no need for more than three initial leads, as the added % of leads does not increase significantly.

Roni says: ***We will probably be asked how the number of initial leads impacted the actual search, not just the number of reachable leads***

11 Related Work

To our knowledge, the only previous work on TONIC was done by Bnaya et al. [**?**]. They proposed a general technique, based on a unique variant of the Multi-Arm Bandit problem, for intelligent application of social network queries. Their technique was applied to TONIC and to the problem of

finding communities in an OSN. Importantly, their goal was to propose a search method that is not based on a domain-specific heuristic such as the BTF and ETF heuristics proposed in their work. Experimentally, their results were comparable to BysP, and they did not consider ETF at all.

Analyzing and searching the social web was previously addressed in the contexts of profile attribute prediction, link prediction, and searching in an unknown graph. Next, we briefly review work in these related problems and discuss their connection to TONIC.

Roni says: ***I changed "profile features" to "profile attributes" to follow Misolve's terminology***

11.1 Profile Attribute Prediction

The task in *profile attribute prediction* is to predict attributes of profiles (e.g., college graduation year and major) in a social network, based on the topological structure of the social network and a limited set of profiles for whom (at least some of) the relevant attributes are known [**?**, **?**, **?**, **?**]. **Roni says:** ***Rami - would be good if you could check if there are more recent work to add***

Mislove et al. [**?**] observed that profiles with similar attributes are grouped into *communities*, which are highly connected subgraphs of the social network. Consequently, profile attributes can be predicted by identifying the *communities* that a given profile is a member of, and predicting that all profiles in the same community have the same attribute value (using majority voting). Mislove et al. [**?**] also proposed the following iterative method to identify a community. First, a set of acquired profiles with the same attribute are grouped together. Then, in every iteration a profile is added to

the community. Selecting which profiles to add to the community was done according to a special utility function they called *normalized conductance*. This process stops when there is no partial profile that increases this utility function (with respect to the previous partial profile that have been added). All profiles found in this process are regarded as being part of the same community and are predicted to share the same attributes. Experiments on two datasets extracted from Facebook showed that this approach enabled predicting with high accuracy attributes such as: collage admission year, graduation year, dormitories and major.

Profile attribute prediction was done by considering features from a combination of networks. Specifically, the smartphone data set of "friends and family" [**?**] was used to predict various attributes of profiles[**?**]. This data set contains phone calls, text messages and available bluetooth connections. Fire et. al. [**?**] investigated two approaches to predict attributes of a known profile:(a) machine learning methods and (b) topology based methods. The machine learning methods use known attributes of a profile to predict missing attribute of the same profile. Topology based methods use the OSN topology to predict missing attributes of a profile by considering attributes of other profiles. Specifically, several social graphs were generated based on user interactions and then graph theory algorithms were applied to predict attributes. **Roni says:** ***Last sentence very vague. Ideas on how to improve it?***

The above methods were used on on various forms of interactions between the members in the "friends and family" dataset, such as identified bluetooth devices, phone calls, and text message. For each interaction form, a social graph was built and different types of attributes were predicted. Bluetooth data, for example, was able to predict a profile's "significant

other" and ethnicity with accuracy of 65% and 60% respectively. Higher accuracy predictions for a range of attributes (e.g., having children and age group) were obtained by considering also internet usage, alarm usage, and usage of location-based services. Fire et. al. [**?**] also showed that the incremental contribution of training data can be fitted to a Gompertz function. The same data set ("Friends and Family") have also been used to predict daily user activity [**?**], infer relationships [**?**] and predict application installation [**?**].

Roni says: ***Paragraph below can be omitted in my view for the journal version. Thoughts?*** Note that the described above work have used information from several networks to predict profile attributes, while in our work we only considered access to a single networks. Barabasi [**?**] have proposed a possible approach to overcome this, by inferring new networks from an existing one. For example, generate a bipartite graph connecting people to groups, based on shared attributes, e.g., all the people that play Tennis (this is similar to the notion of *community* described above). From this bipartite graph, we can infer two new weighted graphs: 1) a graph of people where people are nodes, two nodes have an edge between them if they are related to the same group in the original graph, and 2) a dual graph, of groups that are connected if there is a people in both groups. The weight of the edges in these graph correspond to the number of common groups/peoples the two peoples/groups have together.

TONIC, is conceptually different than all the work on profile attribute prediction. First, in TONIC, the network topology is not known, and is only discovered by costly OSN queries. Second, the task in TONIC is to find profiles related to the target – leads – and not to predict attributes of a given profile. Third, in profile attribute prediction, the task is to estimate

with high accuracy profile attributes while in TONIC an *IsLead()* is required to *verify* that a profile is indeed a lead.

11.1.1 Link Prediction

In general, the *link prediction* problem is the problem of predicting if a link exists between two profiles. There are two variants of the link prediction problem, which we call *static link prediction* and *dynamic link prediction*. In static link prediction, the task is to infer which links are likely to exist among known members, given a partially known social network. In dynamic link prediction, the network dynamics over time are considered and the task is to infer which links will be formed among existing known members, given a history of which links were form and broken in the past.

Recent work [**?**] on static link prediction used machine learning classifier for link prediction on five social network datasets: Facebook, Flickr, YouTube, Academia.edu, and TheMarker. The features used in the learning process were subgraph-based features, edge-based features, and for directed graphs also features based on the number of Strongly Connected Components(SCC) and Weakly Connected Components (WCC). As an additional feature, they introduced the *friends-measure* described earlier in this paper (see Section 5.1.6). These features was used by an ensemble of machine learning algorithms, resulting in a very high accuracy link prediction algorithms.

Liben-Nowell and Kleinberg [**?**] studies the dynamic link prediction problem, predicting which links will be formed in the coauthorship *arxiv.org* network. Their method predicted which link will be formed next by computing a similarity measure between every two nodes in the graph, and

predicting that the pair of nodes with the highest similarly will form a link next. Several similarity measures were proposed and evaluated, such as the "small-world" measure, which finds the shortest path between two nodes, and more sophisticated measures that consider the neighborhoods of the evaluated nodes. It was shown that two nodes are more likely to form a link if their neighborhoods have a large overlap. A more refined similarity measure is Jaccard's coefficient. Jaccard's coefficient of two nodes is the ratio between the intersection of the nodes' neighborhoods and the union of their neighborhoods. *Preferential attachment* is another similarity measure used to predict link formation that is based on the assumption that network growth is proportional to the degree of the node, such that it is more likely that a link will be formed between two nodes with already a lot of links. **Roni says:** ***Need to add a reference for this.***

Other similarly measures used for link prediction analyze the set of paths between a given pair of nodes. One example that was shown to be very successful is the Katz measure [**?**], which sums all of the paths between a pair of nodes weighted by their length. While successful, the Katz measure is hard to compute. Other, easier to compute path-based methods exist. For example *"hitting time"* is defined as the number of steps it takes to reach a node starting from another using random walk. There are also some meta-approaches that can be used on top of any the above methods. The *Low-rank approximation* reduces the adjacency matrix to a *k-rank* matrix and thus reduces some of the noise on the score measures. *Unseen bigram* enhances existing measures by computing the similarly between two nodes x and y by aggregating the score measures with y assigned for the k nodes with the best score with x. **Roni says:** ***Liron - last sentence is incomprehensible (commented is the original which is also incomprehensible)***

As discussed earlier in this paper, TONIC is reminiscent of the link prediction problem. The are, however, several key differences. First, in TONIC a link between a profile and the target needs to be verified by an OSN query (*IsLead()*), while link prediction algorithms only predict a link, but do not verify it. The scientific question is thus different: in TONIC, we ask how to know which profile to query so as to minimize query costs, while in link prediction the question is how to predict links with high accuracy and recall. A second key difference between TONIC and link prediction is that TONIC operates in an initially unknown OSN graph, while in link prediction algorithms all the nodes in the OSN and a large part of the network topology is given as input.

11.2 Search in The Web

In TONIC, we need to explore the social web in an intelligent way in order to extract information about the target. This can be viewed as a special case of *intelligent crawling* [**?**, **?**, **?**]. Intelligent crawling use data collected during the crawl by nodes (web pages) that were already expanded (i.e., downloaded) to decide intelligently which node to expand next.

Many attributes can be taken into consideration while implementing an intelligent crawler. Example of such attributes that have shown to be helpful is the URLs of the the generated but not-yet expanded nodes, the different kinds of links encountered (e.g., link on a picture, link on text) and the number of siblings (i.e., links on the same web page) of a node which have already been crawled. Naturally, a combination of such features yields the most efficient crawling [**?**]. It was also shown that reusing the learned information (as a starting point for the next crawl) gives better results as

well as performing an initial sampling of the random web pages [**?**].

Crawling can be done by several agents in parallel. A dynamic search method that have been proposed in such a setting is called *the Fish Search* [**?**]. The idea behind Fish Search is to simulate the search to a school of fish: When food (relevant information) is found, fish (search agents) reproduce and continue looking for food (other relevant information), but in case that food is not found (no relevant information) or when the water is polluted (poor bandwidth), they die (stop looking for other relevant information). There have been several improvements to Fish Search, called Shark Search [**?**] and Improved Shark Search [**?**], where additional features were considered, such as topic description, textual relevance, URL and link relevance. Although the purpose of the Improved Shark Search is different than our purpose, we may still use similar features to our benefit.

Recently, there have been preliminary work that addressed searching in the social web as an AI search problem [**?**]. Several search algorithms are considered for the problem of finding the shortest path in the social web between two profiles. Since we do not want to find the shortest path to the target, but rather gather information on the target, this work is not directly related to TONIC. However, an interesting aspect in this paper was the average shortest path that has been found between two users in a social web, which was found to be 3.43. This is shorter than the traditional claim that the average number of steps between any two people on Earth is six [**?**], which is the well-known, six degree of separation. This knowledge may be especially important in TONIC, to limit the depth of the web crawling.

A key difference between intelligent crawling and TONIC is that the task of most prior work on intelligent crawling is to uncover large portions of a OSN. In TONIC, however, we wish to retrieve information about a

specific target and avoid further crawling. This requires different problem formulation and heuristics.

11.3 The Web As An Unknown Graph

Searching in a social web is a special case of search problem in an *unknown graph*. A search problem in an unknown graph is a search problem where the structure of the searched graph is not known a-priori, and exploring vertices and edges requires a different type of resource, that is, neither CPU nor memory. When searching the social web, profiles and friendship relations are the vertices and edges of the searched graph, respectively. Accessing a vertex requires acquiring the corresponding profile, which requires sending and receiving network packets (e.g., HTTP request/response).

The main challenge when searching in an unknown graph is to solve the problem while minimizing the exploration cost. This corresponds to the task in TONIC, in which we want to acquire profiles such that the most information about the target will be found. Previous work on searching in an unknown graph have discussed solving the shortest path problem [**?**, **?**], the k-clique problem [**?**, **?**] and finding specific patterns in the unknown graph [**?**].

The TONIC problem is different, in that we do not search a specific pattern in the unknown graph (the social web), but we wish to gather the most information on a given target. In addition most heuristic search research deals with finding a path from an initial node to a predefined goal node. In contrast, the goal of TONIC is to find maximum number of leads while applying at most b acquisitions. While less common, such *utility* oriented search algorithms have been previously discussed in the literature [**?**, **?**].

12 Conclusion and future work

Roni says: ***Due to incomplete results, conclusions are not done yet***

This paper addressed the Target Oriented Network Intelligence Collection (TONIC) problem, in which the task is to find profiles in an OSN that contain information about a given target profile. Beyond academic interest, TONIC is an integral part of commercial and governmental applications. TONIC was formalized as a search problem in an unknown graph and two frameworks for solving it were proposed, BTF and ETF. BTF focuses the search by only acquiring leads. ETF generalizes BTF by allowing non leads to be acquired, as long as there exists a known path from them to a lead that is at most n hops long, where n is a parameter. As n grows, more leads are reachable but the search may become too costly and unfocused. Empirical results suggests that $n = 1$ serves as a valid middle ground.

For both BTF and ETF(1), we present several heursitics for guiding the search. These heuristics are evaluated experimentally on a real OSN. Evaluation results show that (1) in general, heuristics that incorporate the promising measure are more robust to topological properties of the target and (2) in particular, the Bayesian Promising (BysP and EBysP) heuristics significantly outperforms other heuristics. (3) using ETF results in substantially more leads than BTF, and (4) ETF with the EBysP heuristic finds leads faster than BTF with BysP. We have also examined two social networks: Google+ and dblp. **Liron says:** ***after the dblp is done, i need to talk about it here*****Roni says:** ***Yes, need to be discussed.***

There are many exciting directions for future work. One of them is to create heuristics that consider a profile's textual data (hobbies, demographic

information, etc.) and not just the topology of the CKG as the current heuristics do. One other direction is to utilize the power of machine learning to predict which potential leads will turn to be leads de facto. ML models can be trained using both data from past executions on various targets and data on profiles acquired during investigation of current target.

Another direction is to consider leads with different rewards, where finding a lead that provides more information about *target* is preferred. This raises a complex task of how to quantify information. Furthermore, information found from one lead can affect the value of information from other leads, as some information may overlap. In addition, we assumed that *IsLead()* and *Acquire()* are always applicable. Future work can consider network errors that may cause queries to fail with some probability, introducing uncertainty.

13 Ethical Aspects in TONIC

As a final note, we would like to raise the ethical issues concerning the investigation of efficient TONIC solvers. It is general knowledge that people and organization use OSNs and other publicly available sources to gather information about specific entities (e.g., people, organizations, and other social groups). Social networks are also useful for monitoring the affiliates of known criminals. This often occurs manually, for example, before hiring a person. Although this information is publicly available, some ethical concerns may be raised.

We emphasize that the use of an efficient TONIC solver can be done for many good purposes. It can serve as an helpful tool for law enforcement agencies, For example, imagine a search for information about a paedophile in an OSN. In fact, the social network paradigm has been successfully used to

investigate organised crime in the Netherlands [**?**]. Alternatively, an efficient TONIC solver can be used as a tool for preserving privacy, by allowing a person to find how much publicly available information exists about him/her in a given OSN and change his/her privacy setting accordingly.

14 Acknowledgments

This research was supported by the Ministry of Science and Technology, Israel and the Office of the Chief Scientist.