Extended Framework for Target Oriented Network Intelligence Collection

Liron Samama-Kachko **Rami Puzis** **Roni Stern** **Ariel Felner**
Information Systems Engineering Department, Ben Gurion University, Beer Sheva, Israel

Abstract

The Target Oriented Network Intelligence Collection (TONIC) problem is the problem of finding profiles in a social network that contain publicly available information about a given *target* profile via automated crawling. Such profiles are called *leads*. Leads can be found by crawling the network using the profiles' friend lists (immediate neighborhood) in order to decide which profile will be crawled next.

Assuming that leads tend to cluster together, prior work limited the search for new leads only to immediate neighbors of the leads previously found. In this paper we relax this limitation, and extend the scope of the search to a wider neighborhood, including the possibility of crawling to non-leads, i.e., profiles that have no publicly available information about the target. We propose a set of heuristics that guide this search. Experimental results show that with the new setting more leads can be found and leads are found faster. In addition, we perform a cost benefit analysis of the search, weighing the reward of finding leads with the costs of the search.

Introduction

Online Social Networks (OSNs) are an increasing part of our lives. They are used to connect friends, share information and interests, play games and more. The increasing use of OSNs results in an abundance of personal data available online and accessible to third parties. This data can be useful in many ways. OSN researchers use it to study the global community and online relationships amongst people. Commercial companies utilize it to recommend relevant commodities and entertainment to users to improve their browsing experience. The data also helps law enforcement investigations and provides information about criminals. This paper addresses the problem of searching for data about a given OSN profile denoted as *target*.

Information contained in a profile as well as its *list of friends* (LOF) can be acquired by querying an application program interface (API) provided by the OSN or by scraping the OSN web pages associated with the profile.[1]

Some information about *target* can be directly obtained from its public OSN profile but this information may be insufficient. Moreover, depending on privacy settings, the content of *target*'s profile including its LOF may be inaccessible to third parties. In such cases, much more information about *target* can be obtained by crawling profiles of *target*'s friends and other related profiles (Altshuler et al. 2013). However, since the *target*'s profile is inaccessible, it's LOF is unknown thus turning the problem of finding the *target*'s friends to a challenge.

We denote by *lead* a profile that contains information (e.g., photos, comments, events, etc.) related to *target*. Finding leads in an OSN is not a trivial task. An OSN crawler that searches for leads can be a very effective solution. However, most OSNs limit the number of queries that can be executed by a crawler and the large OSNs charge for excess queries. Thus, crawlers should be designed with care and reduce overcharges by focusing the search on those part of the OSNs which are likely to contain leads.

To this end, Stern et al. (2013) defined the Target Oriented Network Intelligence Collection (TONIC) problem – the problem of finding leads in an OSN while aiming to minimize the incurred costs. The output of a TONIC solver is a list of leads. Information about the target can be extracted from theses leads using standard techniques (Chang et al. 2006; Tang et al. 2010; Pawlas, Domanski, and Domanska 2012). The information extraction phase is beyond the scope of this paper and is a large field of research of its own.

Stern et al. (2013) modeled TONIC as a search problem in an unknown graph and proposed several heuristics for a search-based solver. Others used TONIC as a benchmark to evaluate the volatile multi-arm bandit (VMAB) model, showing comparable results to Stern et al., even without a sophisticated heuristic (Bnaya et al. 2013). All previous work on TONIC relied on the natural clustering of OSNs, assuming that leads are clustered together in tightly connected group of OSN friends and that focusing solely on leads would be sufficient for finding information about *target*. Thus, in order to avoid unnecessary waste of resources on exploration of non-leads they restricted the crawling to the immediate neighborhood of known leads.

However, most people have more than one tightly connected group of OSN friends e.g., family, co-workers, and college friends. These groups (sometimes called social circles) may not share any common member except the target. Thus, shortsighted crawling decision, as done by previous TONIC solvers, may result in exploring a single social cir-

[1] User license agreement of many OSNs prohibit access of automatic crawlers to the profile pages. For such cases explicit written permission of the OSN service owner is required.

cle leaving others under-explored. In this paper we remedy this problem and propose an Extended TONIC Framework (ETF) in which exploration of non-leads is allowed, while still focusing on profiles related to the *target*. While a non-lead does not provide information about the *target* during the information extraction phase, it has potential to reveal more social circles. Additional social circles will in turn enable finding more leads and as a consequence gathering more information about the *target*.

Nevertheless, a crawler cannot be allowed to acquire non-leads unrestrictedly. Traditional crawling methods such as forest fire (Leskovec, Kleinberg, and Faloutsos 2007) or snowball (Lee, Kim, and Jeong 2006) may obtain many OSN profiles with few leads among them. To this end, we propose several heuristics for guiding ETF intelligently. ETF with the Extended Bayesian Promising (EBysP) heuristic proposed in this paper finds leads faster and eventually finds many more leads. For example, 50% and 60% of leads are found 2 and 4 times faster than using the previous approaches.

Additionally, we evaluate the cost versus the benefit of using the TONIC heuristics. We conclude that ETF with EBysP performs best for short searches where the reward for finding a lead is high.

Problem description

The basic entity in TONIC is the *profile* of a person in a specific OSN (e.g., Facebook or Google+). Profiles are connected to each other via a "friendship" relation. The *list of friends* of a given profile p is denoted by LOF(p). Given a *target* profile, a *lead* is a profile that contains information about the target. Practically there could be many ways to determine who is a lead. For example, a profile p can be regarded as a lead if (1) the target is contained in $LOF(p)$, or (2) p contains a post by the target, or (3) the target is tagged in one of the photos shown in p. Other criteria for determining whether a profile is a *lead*, e.g., using sophisticated IE techniques, are also possible.

In this work we assume that two high-level OSN queries can be applied on a profile p:

1) *IsLead(p)*: This is a binary query that returns whether p is a lead or not. In our experiments we used a simple *IsLead(p)* function which returns $True$ if $target \in LOF(p)$ and $False$ otherwise. The assumption is that if p is a friend of *target* then it probably contains other information on $target$. Nevertheless, our algorithms are applicable across different implementations and definitions of *IsLead()*.

2) *Acquire(p)*: This is a heavier query which extracts substantial information from p. In particular, it returns pointers to new profiles found in $LOF(p)$.

Given a set of profiles to start with, one can search the OSN for leads by calling the *IsLead()* and *Acquire()* queries. The input to TONIC is a *target* and a set of *initial leads*. The initial leads are profiles were found manually, using previous knowledge about *target*. These initial leads provide a starting point for the automated phase of the search. The output of TONIC is a collection of leads, found and acquired using the *IsLead()* and *Acquire()* queries.

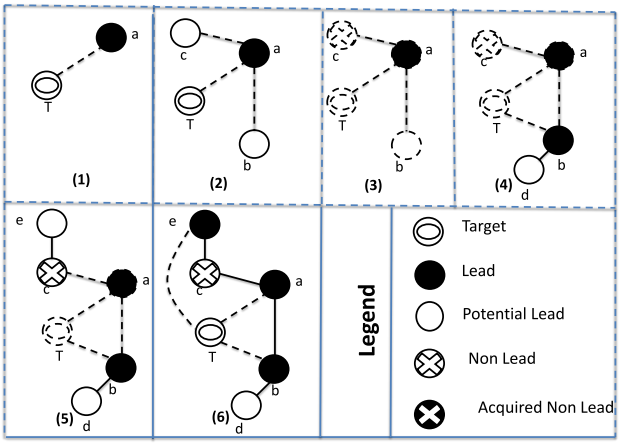

Figure 1: An example of a search for leads. Edges denote a friendship relation. Dotted edges denote friends of *target*.

We consider three types of profiles: *leads*, *non-leads*, and *potential leads*. Leads and non-leads are profiles that *IsLead()* identified as being or not being leads, respectively. Potential leads are known profiles that were not (yet) evaluated by *IsLead()*. When a profile is acquired, its LOF is extracted, possibly revealing new potential leads. Figure 1 shows an example of several steps of this process. There is a single initial lead a connected to *target* (step 1). Acquiring a reveals that its LOF contains two potential leads b and c (step 2). Performing *IsLead(c)* reveals that c is a non-lead (step 3) and *IsLead(b)* reveals that b is a lead. *Acquire(b)* returns its LOF, which contains a new potential lead d (step 4). This search process can continue, finding and acquiring more and more leads.

Performing *IsLead()* and *Acquire()* queries incur costs in terms of both computational resources and network activity. In addition, most OSN services limit automatic web scraping attempts as well as massive exploitation of their API. We denote by cost(*Acquire()*) and cost(*IsLead()*) the costs of *Acquire()* and *IsLead()*, respectively. Usually, cost(*IsLead()*)<cost(*Acquire()*) because *IsLead()* only checks the internal page of the profile while *Acquire()* needs to pull out other pages and URL addresses of the friends. The TONIC problem is defined as follows:

Definition 1 (The TONIC Problem) *Given a target profile, a set of initial leads and a budget b, the TONIC problem is to find as many leads as possible without exhausting the budget b.*

Other variants of TONIC can also be formulated. For example, finding a fixed number of leads while minimizing the total cost of the executed queries, or an "anytime" variant where an end-user may stop the search unexpectedly.

Definition 1 slightly differs from previous definitions of TONIC (Stern et al. 2013; Bnaya et al. 2013). They did not distinguish between the *IsLead()* and *Acquire()* queries, assuming a unified *profile acquisition* query. Applying that unified query to a potential lead p first performs *IsLead(p)*.

Algorithm 1: BFS for TONIC (BTF and ETF(n))

Input: *target* the target profile
Input: b, the max total queries allowed
Input: $InitialLeads$, the set of initial leads
Output: L, the leads found

1 $L \leftarrow InitialLeads, PL \leftarrow \emptyset, NL \leftarrow \emptyset,$
$CKG \leftarrow L \cup NL \cup PL$
2 **foreach** *l in InitialLeads* **do**
3 $LOF \leftarrow$ *Acquire(l)*
4 Add LOF to $OPEN$ and to PL
5 **while** $OPEN \neq \emptyset$ *AND* $b > 0$ **do**
6 $best \leftarrow$ ChooseBest($OPEN$)
7 $bestislead \leftarrow$*IsLead(best)*
8 $b \leftarrow b - cost(isLead())$
9 $PL \leftarrow PL \setminus \{best\}$
10 **if** $best \in PL \wedge bestislead$ **then**
11 Add $best$ to L and CKG
12 $LOF \leftarrow$ *Acquire(best)*
13 $b \leftarrow b - cost(acquire())$
14 Add $LOF \setminus CKG$ to $OPEN$, PL, CKG
E 15 Add $LOF \cap NL$ to $OPEN$
E 16 **else if** $best \in PL \wedge \neg bestislead$ **then**
/* best is a new non lead */
E 17 Add $best$ to NL and CKG
E 18 Add $best$ to $OPEN$
E 19 **else if** $best \notin PL \wedge d_L(best) \leq n$ **then**
/* best is a known non-lead */
E 20 $LOF \leftarrow$*Acquire(best)*
E 21 $b \leftarrow b - cost(acquire())$
E 22 Add $LOF \setminus CKG$ to $OPEN$, PL, CKG
E 23 Add $LOF \cap NL$ to $OPEN$
24 **return** L

If p is a lead it is acquired, otherwise it is discarded. While the unified profile acquisition query is sufficient to describe the basic TONIC framework (BTF) described next, the separation of *IsLead()* and *Acquire()* is required to describe the extended TONIC framework (ETF) described later.

Previous work: Basic TONIC Framework

TONIC can be modeled as a *search problem in unknown graphs*. In such problems the underlying graph is only partially known and is further explored by expanding nodes using an external *expansion* query where the task is to minimize the total cost of these queries (Stern, Kalech, and Felner 2012). In TONIC, the searched graph $G = (V, E)$ is the searched OSN where vertices V represent profiles and edges E represent the friendship relationships, i.e. $(v_1, v_2) \in E$ iff $v_2 \in LOF(v_1)$. The expansion action in TONIC corresponds to applying the *IsLead(p)* query followed by *Acquire(p)* if p is a lead. Otherwise, p is discarded.

Search problems in unknown graphs can be solved with a best-first search (BFS) approach. The Basic TONIC Framework (BTF) follows this modeling, and implements a BFS for TONIC as described in Algorithm 1 (Stern et al. 2013). Lines with a preceding *E* (15-23) are to be ignored for now; they are only relevant for the *Extended TONIC Framework* (ETF) introduced later.

BTF maintains the following entities:
(1) OPEN, which consists of all the current potential leads.
(2) L, NL and PL, which are the sets of leads, non-leads and potential leads found so far, respectively.
(3) The *currently known subgraph* of the OSN, denoted $CKG = (V', E')$, which contains all known profiles $V' = L \cup NL \cup PL$ (see line 1), and edges of profiles that were already expanded $E' = \{(u, v) \in E | u \in L\}$.

In every iteration, a single profile ($best$) is chosen from OPEN (line 6). After $best$ is chosen, *IsLead(best)* query is performed and $best$ is removed from PL and from OPEN. If $best$ is a lead then its LOF is obtained by the *Acquire(best)* query and all newly discovered potential leads are added to OPEN and to PL.

Every query decrements the budget b by the respective cost of the query (lines 8 and 13). This process continues until the budget b was exhausted by the queries or until OPEN is empty, meaning that all reachable profiles have been acquired. The efficiency of BTF depends on the method of choosing $best$ from OPEN.

Heuristics for BTF

The $best$ profile chosen for expansion (line 6) is determined by analyzing the CKG, which contains all the expanded nodes and their neighbors. The following two heuristics for choosing $best$ were proved effective for BTF (Stern et al. 2013).

Known Degree (KD). The KD heuristic chooses to expand the potential lead with the highest degree. Since the degree in G of a potential lead is only known after it is acquired, KD uses instead the degree of that potential lead in CKG. The intuition behind this is that highly connected nodes in proximity of the *target* have higher chances to contain some information about the *target*.

Bayesian Promising (BysP). This heuristic estimates the probability that a potential lead will turn out to be a lead. This is done as follows. For every lead $m \in L$ we partition its LOF to leads, non-leads and potential leads, denoted as $L(m)$, $NL(m)$ and $PL(m)$, respectively. The *promising factor* of a profile m, denoted by $pf(m)$ is the fraction of lead friends out of known friends $\left(\frac{L(m)}{L(m)+NL(m)}\right)$.[2] $pf(m)$ is intended to estimate the probability that a potential lead in m's LOF is a lead. A given potential lead may be connected to many leads, each having a different $pf(\cdot)$. The $BysP(\cdot)$ aggregates the promising factors based on a Naïve Bayes approach to aggregate probabilities as follows:

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

The BysP heuristic chooses to expand the potential lead pl with the highest $BysP(pl)$. BysP was shown to outperform all other heuristics in BTF (Stern et al. 2013).[3].

[2] If $L(m) + NL(m) = 0$ then $pf(m)$ is set to 0.5.

[3] Bnaya et al.(2013) proposed a very sophisticated algorithm which showed very marginal improvement over BysP so we do not

Regardless of the heuristic used, non-leads are never expanded in BTF. If *IsLead(p)* finds out that p is a non-lead, p is discarded. This may preclude reaching some of the leads. Consider the example in Figure 1. Node e is a lead, but it will never be acquired by BTF because its only neighbor c was found to be a non-lead and was discarded. The Extended TONIC Framework (ETF) described next overcomes this limitation by considering to expand some of the non-leads in order to find more leads.

Extended TONIC Framework

ETF is a generalization of BTF that allows acquisition of non leads. By extending the range of reachable profiles, ETF finds new leads that were unreachable by BTF. For example, in Figure 1 the profile e belongs to a different social circle than a and would not be identified by BTF because c is a non-lead. ETF assumes that more leads are reachable through the extended social circles of *target* (e.g. friends of friends) and thus allows expanding non-leads.

Similar to any unconstrained crawling, ETF can potentially span to the entire OSN resulting in unreasonable waste of resources. We, therefore, make some restrictions on EFT and do not allow exploring the OSN farther than a few hops away from the *target*. This restriction has several advantages. First, it provides a clear stopping criteria to the search that may occur before reaching the budget b. Second, it may focus the search towards areas of the OSN where leads are more likely to be found, "protecting" to some extent the search from misleading heuristics.

Since *target* is not part of the CKG, we apply the restriction on ETF based on known leads as a reference point. Let $d_L(p)$ be the shortest distance in the CKG between a profile p and the closest acquired lead. ETF(n) is a TONIC framework, where a profile p may only be acquired if $d_L(p) \leq n$. Otherwise, it is discarded. BTF is, therefore, ETF(0), a special case where the acquisition of profiles is limited to leads only. We use the term *tier* n to denote the set of profiles that may be acquired by ETF(n).

The pseudocode of ETF(n) is also given in Algorithm 1 with several modifications (lines 15-23 preceded with *E*). The key difference between BTF and ETF is how non-leads are handled. In BTF, a potential lead found to be a non-lead is discarded. In ETF(n), discovered non-leads are reinserted into OPEN (lines 15, 18, and 23), to be later considered for expansion if their $d_L(\cdot) \leq n$ (lines 19–23). Note that $d_L(\cdot)$ can change as the search progresses. A non-lead p that was discarded in line 19 since $d_L(p) > n$ can be reinserted into the OPEN (line 15) when a shorter path connecting it to a lead is discovered.

Figure 1 shows an example of 6 iterations of ETF. Steps 1-4 are similar to BTF. In step 5, node c, which is a non-lead, is expanded and $e \in LOF(c)$ is added to OPEN (Line 18). Then, in step 6, *IsLead(e)* is performed and e is found to be a lead. Recall that e could not be found with BTF.

The average distance between profiles in online OSN is between four and five (Ugander et al. 2011) making $ETF(4) \approx ETF(\infty)$ since there would rarely be more than four nodes between any two profiles and in particular more than four non-leads between any two leads. Table 1 shows the percentage of leads at a given tier (out of the entire set of leads) for a given number of initial leads. Even with 5 initial leads, the marginal contribution of moving from tier zero to one is not negligible (35% more leads are found for one initial lead and 20% for 5 initial leads). By contrast, even with a single initial lead, the marginal contribution of any additional tier above 1 is negligible. We thus limit our experimental results below to tier 1 (ETF(1)) only. The numbers converge to 89% as there were around 11% of leads which were unreachable by this set of initial leads.

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 1: % of reachable leads from different number initial leads

As we report below, besides the advantages that many more leads are now available by ETF(1), these leads are found much faster than BTF.

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 heuristics for ETF.

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 2 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 found by *IsLead()* up to that point.[4] As can be seen, EFIFO discovers leads slower than BysP but eventually reaches more leads. 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 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.

consider it here

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

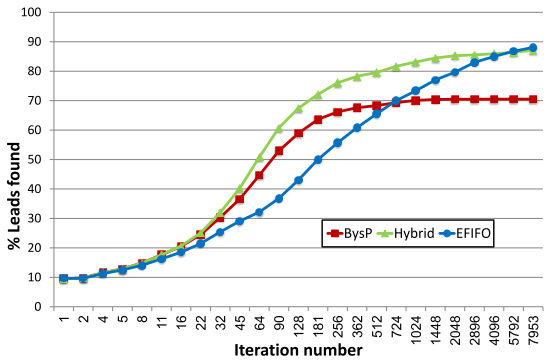

Figure 2: BysP(0), EFIFO(1) and the hybrid heuristic. The x-axis is the is the number of *IsLead()* calls. The y-axis is the number of leads found by *IsLead()* up to that point.

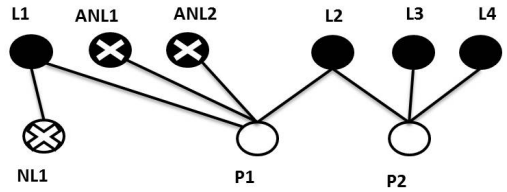

Figure 3: Example for the KD and KDL heuristics.

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. Empirical evaluation (see Figure 2) shows that *hybrid* switches heuristics when EFIFO starts to outperform BysP thus performing as the best of the two throughout the search.

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 3 depicts a CKG that demonstrates the difference between KD and KDL. The legend for this figure is the same legend shown in Figure 1. 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 $KDL(NL1) = 1$.

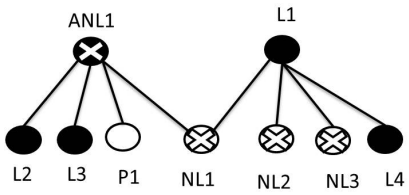

Figure 4: Example for the EBysP heuristic

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 5 explained in the experimental section).

EBysP BysP was the best performing heuristic for BTF (Stern et al. 2013). BysP chooses to expand the potential lead $pl \in OPEN$ having the highest $BysP(pl) = 1 - \prod_{m \in L(pl)}(1 - pf(m))$. A small modification is required in order to apply it to ETF as follows:

$$EBysP(p) = 1 - \prod_{m \in L(p) \cup NL(p)} (1 - pf(m))$$

The key difference between $BysP(p)$ and $EBysP(p)$ is that $BysP(p)$ only aggregates the $pf(\cdot)$ values of the leads that are adjacent to p ($L(p)$) while $EBysP(p)$ aggregates over all the acquired neighbors of p ($L(p) \cup NL(p)$). We use EBysP to denote the ETF heuristic that expand the profile p with the highest $EBysP(\cdot)$ in $OPEN$, regardless if p is a potential lead or a non lead.

To illustrate EBysP, consider Figure 4. $OPEN$ 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.

Friends Measure (FM) The TONIC problem bears some resemblance to the *link prediction* problem (Liben-Nowell and Kleinberg 2007), 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*.

In particular, we propose an ETF heuristic that is based on the *Friends Measure*, which 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 (Fire et al. 2013). Formally, the friends

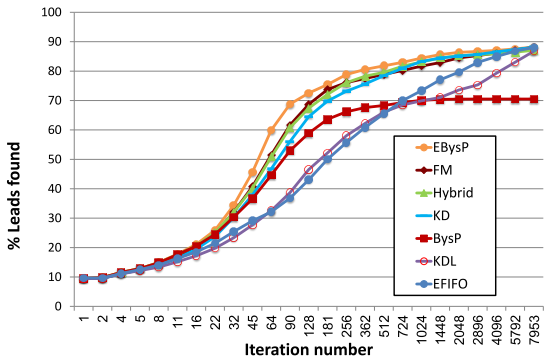

Figure 5: % of leads found vs. iteration number

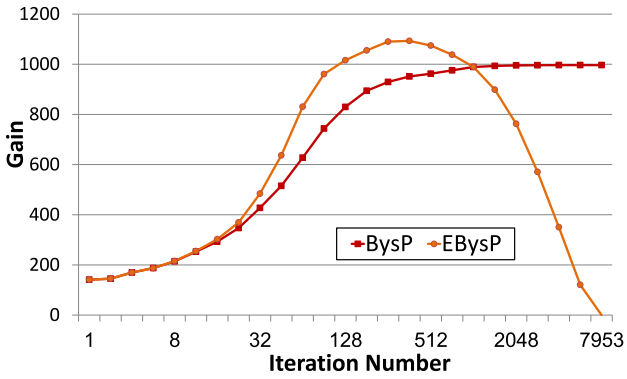

Figure 6: The net gain by BysP and EBysP when cost(*Acquire()*)=1 and $R_{lead} = 40$.

measure (fm) between profiles t and p is:

$$fm(t,p) = \sum_{x \in L} \sum_{y|(p,y) \in E} \delta(x,y)$$

where L is the set of leads and $\delta(x,y)$ is defined as:

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

The ETF heuristic that is based on the Friend Measure is called FM and chooses as $best$ the candidate p with the highest $fm(target, p)$. Experimentally, we show later (see Figure 5) that fm performs reasonably well as a TONIC heuristic.

Experimental Evaluation

We evaluated the performance of ETF with the proposed heuristics on Google+, one of the largest social networks, having more than 540M registered users and 300M users that are active monthly (according to Wikipedia). The exact definition of a *lead* in our experiments is a profile that is a friend of $target$. Thus, *IsLead(p)* returns $True$ if $target \in LOF(P)$ and $False$ otherwise. In every experiment a single Google+ profile was set as *target* and three random neighbors of *target* as the initial leads. We used the benchmark set of targets and initial leads used by Stern et al. (2013). The target degrees range from 30 to 86 while the number of vertices and edges considered in ETF(1) range from 146 to 4088 and from 104 to 4776 respectively. The full data set was made available by Fire et al. (2013) and contains 211K Google+ profiles with 1.5M links between them.

Performance of ETF heuristics

Figure 5 compares the number of leads found (the y-axis) by each of the proposed ETF heuristics as a function of the total number of nodes expanded. For reference, we also present the results for BysP (which is a 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,the intelligent heuristics and in particular the EBysP heuristic, uses the topological information obtained to better guide the search, thus performing better finding more leads faster. In particular, EBysP substantially outperforms the previous state-of-the-art BysP throughout the search. For example, after 45 node expansions, 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 the early stages of the search. At the end of the search all heuristics converge at the point in which no more leads can be found.

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 comparable to the performance of the more sophisticated heuristics EBysP and FM and much better than the performance of KDL which is more focused toward leads.

Cost-Benefit analysis of TONIC heuristics

The task in TONIC according to Definition 1 is to maximize the number of leads found within a given budget that is decremented when applying the OSN queries of *IsLead()* and *Acquire()*. Consider an application of TONIC where the monetary reward of finding a lead (R_{Lead}) (and passing it to the information extraction phase for further analysis) is known. In some cases the cost of *Acquire()* (cost(*Acquire()*) can also be estimated in monetary terms allowing measuring the net gain of the search process throughout the search. We assume that cost(*IsLead()*) is negligible compared to cost(*Acquire()*), as *Acquire()* is often a heavier operation. We define the *net gain* (NG) of a search as the total reward from finding leads minus the total cost of all *Acquire()* queries.

During late stages of the search the frequency of leads decreases as can be seen from the slopes of all heuristics in Figure 5. Therefore, the *gain* may drop to a point where the search process is not worthwhile.

Figure 6 demonstrates this, showing the net gain (y-axis)

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

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

as the search progresses (x-axis) 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 allow to acquire only 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 6 the acquisition of non leads by EBysP results in much more frequent discovery of leads at the first stages of the search and overall higher gain in comparison with 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 $R_{lead} \geq 5$. This is reasonable because larger R_{lead} makes the long term investment of EBysP in acquiring of non leads worthwhile.

To gain a deeper understanding of the gains of BysP and EBysP throughout the search, Figure 7 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 6. 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()*).

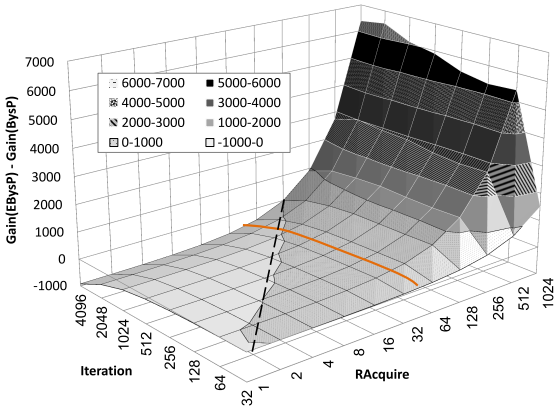

Figure 7: The difference in net gain between BysP(0) and BysP(1)

Conclusions and Future Work

We presented ETF, a new framework for solving TONIC which generalizes BTF by allowing non leads to be acquired. We demonstrated that this facilitates finding more leads. Then, we presented several heuristics for guiding ETF towards finding leads faster. Empirical evaluation on Google+ showed that (1) using ETF results in substantially more leads than BTF, and (2) ETF with the EBysP heuristic finds leads faster than BTF with BysP.

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.

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.

An important future work is to evaluate the proposed heuristics on several OSNs, such as Facebook and Twitter, and studying relations between various OSN properties and effectiveness of ETF heuristics. Even inside a specific OSN, it may be the case that specific heuristics work better in different parts of the network. A preliminary study of this was done by Stern et al. (2013) for BTF.

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 man-

ually, 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 (Klerks 2001). 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.

Acknowledgments

This research was supported by the Ministry of Science and Technology, Israel and The Israel Science Foundation (ISF) under grant #417/13 to Ariel Felner.

References

Altshuler, Y.; Aharony, N.; Elovici, Y.; Pentland, A.; and Cebrian, M. 2013. Stealing reality: When criminals become data scientists (or vice versa). In Altshuler, Y.; Elovici, Y.; Cremers, A. B.; Aharony, N.; and Pentland, A., eds., *Security and Privacy in Social Networks*. Springer New York. 133–151.

Bnaya, Z.; Puzis, R.; Stern, R.; and Felner, A. 2013. Social network search as a volatile multi-armed bandit problem. *ASE Human* 2(2):pp–84.

Chang, C.; Kayed, M.; Girgis, M.; Shaalan, K.; et al. 2006. A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering* 18(10):1411.

Fire, M.; Tenenboim-Chekina, L.; Puzis, R.; Lesser, O.; Rokach, L.; and Elovici, Y. 2013. Computationally efficient link prediction in a variety of social networks. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5(1):10.

Klerks, P. 2001. The network paradigm applied to criminal organizations: Theoretical nitpicking or a relevant doctrine for investigators? recent developments in the netherlands. *Connections* 24(3):53–65.

Lee, S. H.; Kim, P.-J.; and Jeong, H. 2006. Statistical properties of sampled networks. *Physical Review E* 73(1):016102.

Leskovec, J.; Kleinberg, J.; and Faloutsos, C. 2007. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1(1):2.

Liben-Nowell, D., and Kleinberg, J. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58(7):1019–1031.

Pawlas, P.; Domanski, A.; and Domanska, J. 2012. Universal web pages content parser. 291:130–138.

Stern, R.; Samama, L.; Puzis, R.; Felner, A.; Bnaya, Z.; and Beja, T. 2013. Target oriented network intelligence collection for the social web. *AAAI AIW*.

Stern, R.; Kalech, M.; and Felner, A. 2012. Finding patterns in an unknown graph. *AI Commun.* 25(3):229–256.

Tang, J.; Yao, L.; Zhang, D.; and Zhang, J. 2010. A combination approach to web user profiling. *ACM Trans. Knowl. Discov. Data* 5(1):2:1–2:44.

Ugander, J.; Karrer, B.; Backstrom, L.; and Marlow, C. 2011. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*.