World Wide Web (2019) 22:1447–1480
https://doi.org/10.1007/s11280-018-0648-0

Target oriented network intelligence collection: effective exploration of social networks

Rami Puzis[1] · Liron Kachko[1] · Barak Hagbi[1] · Roni Stern[1] · Ariel Felner[1]

Received: 7 May 2017 / Revised: 10 June 2018 / Accepted: 1 November 2018 /
Published online: 15 November 2018

Abstract
Target Oriented Network Intelligence Collection (TONIC) is a crawling process whose goal is to find social network profiles that contain information about a given target. Such profiles are called *leads* and the TONIC problem is how to minimize crawling costs incurred while finding them. We model this problem as a search problem in an unknown graph and present a best-first search approach for solving it. Three key challenges are (1) which profiles to consider crawling to, (2) how to prioritize the crawling order, and (3) when additional crawling is not worthwhile. For the first challenge, we propose two frameworks: the Restricted TONIC Framework (RTF), that restricts the search to immediate neighbors of previously found leads, and the Extended TONIC Framework (ETF), that extends the scope of the search to a wider neighborhood. Guidelines for when to choose which framework are provided. For the second challenge, we propose a set of effective topology-based heuristics that guide the search towards profiles that are more likely to be leads. For the third challenge, we propose to use data collected in previously executed crawls to learn when additional crawling is expected to be useful.

Keywords Artificial intelligence · Heuristic search · Online social networks

✉ Roni Stern
sternron@post.bgu.ac.il

Rami Puzis
puzis@bgu.ac.il

Liron Kachko
ssliron@gmail.com

Barak Hagbi
barak80@gmail.com

Ariel Felner
felner@bgu.ac.il

[1] Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Be'er Sheva, Israel

1 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.

The process of collecting the information available in an OSN about a given person or group of interest can be automated using information extraction (IE) techniques [13]. We refer to such a person or group of interest as the *target*. Awareness to privacy issues and to data leakage causes many users to tighten their OSN privacy settings limiting access to their profiles. Consequently, the target's profile is inaccessible to third parties and information about the target cannot be collected from its profile. Furthermore, there might be scenarios where the target does not even have an OSN profile, but information about the target is still available in the OSN.

In such cases, an alternative way to collect information about the target is through information available in other OSN profiles. For example, information about the target can also be collected from profiles that contain photos of the target, public posts made by the target or posts made by other profiles about the target. If the target has a profile in the OSN, then a likely source of information about the target is in the profiles of its acquaintances, where acquaintances can be defined using the OSN *friendship relationship*. It is prohibitively challenging to conceal information about the target that other profiles expose. A profile that exposes information about the target is called a *lead*.

In this paper we address the problem of finding as many leads as possible while minimizing the number of inspected profiles. We call this problem the *Target Oriented Network Intelligence Collection (TONIC)*. In our model, extraction and analysis of relevant information from the OSN is encapsulated in two operators: *IsLead* which is a light query that determines whether a profile is a *lead* or not, and *profile acquisition* which is a heavy operation that fully extracts and observes the information of a given profile.

These operators can be carried out by utilizing web scrapping or the OSN application programming interfaces (API) followed by standard IE techniques [13, 43, 51].

We solve TONIC with Artificial Intelligence techniques, formalizing it as a heuristic search problem. Profiles and friendship relations form the vertices and edges of a graph, respectively, and we use a best-first search approach to search this graph [44, 52]. We visit profiles in a best-first order according to heuristic functions that we develop in this paper and apply the *IsLead* and *profile acquisition* operators on the *best* profile in an intelligent manner.

Two frameworks are proposed for solving TONIC, the Restricted TONIC Framework (RTF) and the Extended TONIC Framework (ETF). RTF restricts the search to known leads and their neighbors. Therefore, in RTF profiles that are found to be non-leads are omitted from deeper exploration. The rationale behind restricting acquisition of non-leads is that the cost of *profile acquisition* can be quite high and it may not result in the discovery of new leads. By contrast, ETF does not restrict acquisition of non-leads and allows the search to continue through the extended social cycles of a target. As a generic middle ground between RTF and ETF, we proposed ETF(n), in which non-leads are only explored if they are at most n steps away from a known lead. Thus RTF=ETF(0) and ETF= ETF(∞).

Several intelligent heuristics are proposed for both RTF and ETF. These heuristics guide the search by analyzing the currently known subgraph of the OSN. Importantly, all our investigated heuristics are based solely on the topology of the OSN being crawled, and are thus orthogonal to the details of the *IsLead* and *profile acquisition* operators.

In some problem settings, it is possible to reach a stage where performing additional crawling is not worthwhile and even wasteful. This occurs when the expected reward from obtaining new leads is smaller than the expected crawling costs of getting to them. For problem settings when this can occur, we propose two learning-based stopping conditions that use data about past crawls to decide when to stop searching for more leads and halt the current crawling process.

The proposed heuristics and frameworks are experimentally evaluated on three OSNs: Google+, Pokec [49], and LiveJournal [6, 35] (an online community that allows member to maintain journals, blogs, and define friendship between profiles). Results show that each framework performs best under different circumstances. We found that in general RTF and $ETF(1)$ are worthwhile, while ETF(n) for $n > 1$ does not contribute significantly to the number of leads found. We further analyze the tradeoff between the cost of searching and the benefit of finding more leads. The results of this analysis show that $ETF(1)$, with the proposed heuristics, is better than RTF for short searches and when the reward for finding a lead is high; while RTF, with a proper heuristic, is better suited for a long term search process with moderate and low rewards.

This paper contains a comprehensive and unifying presentation of material from two previously published conference papers [45, 48]. This paper also contains several significant improvements and novel contributions over these conference papers, including:

1) Two algorithms for learning when to stop crawling in order to maximize net gain (Section 7.2).
2) Added reliability to our experimental results through the evaluation on two additional OSNs (Section 9).
3 An additional, novel, form of analysis for TONIC, in which TONIC is modeled as an Information Retrieval task and the experimental results are analyzed accordingly (Section 8).
4) A significantly more comprehensive related work.
5) Clean description of the proposed algorithm, including additional examples and supporting figures.

The paper is organized as follows. In Section 2 we formally define TONIC and relevant terminology. Section 3 contains a review of related works. Section 4 explains how TONIC can be formalized as a search problem in an unknown graph, and outlines the algorithmic framework we use. Section 5 describes the Restricted TONIC Framework (RTF), RTF heuristics, and presents experimental results to evaluate their performance. Section 6 does the same for the Extended TONIC Framework (ETF) – present it along with ETF heuristics and evaluates them. The trade-off between RTF and ETF is discussed in Section 7 in terms of cost and benefit of the respective search processes. In Section 8 we present an alternative view of TONIC as an Information Retrieval task. In Section 9 we provide additional support for our conclusions by reporting on additional experimental results performed on other OSNs. Then, we summarize the main findings and outline the next steps in Section 10, and discuss ethical aspects of various TONIC applications in Section 11.

2 The TONIC problem

In this section we formally define the general TONIC problem and its variants as evaluated in our experiments.

Definition 1 (profiles) The basic entity in TONIC is a *profile*, which is associated with a specific OSN (e.g., Google+). Profiles are connected to each other via a "friendship" relation. The *list of "friends"* (LOF) of a given profile p is denoted by $LOF(p)$.

TONIC is defined with respect to a *target* OSN profile. The *target* profile can be associated, for example, with a particular person, group of people, or legal entity.

Definition 2 (Lead and Non-Lead) A profile p is called a *lead* if it has the target in its LOF. Otherwise, it is called a *non-lead*.

TONIC is motivated by situations in which one wants to find information about the target. Since leads are more likely to have such information, the process we envision is to first find profiles that are leads, and then extract valuable information from them if such exists. The information extraction is out of the scope of this paper, since deciding which information about the target is valuable is application dependent and it may require intervention of a human analyst. Thus, in TONIC we focus on finding leads (i.e., friends of the target).

Information in general and the LOF in particular can be extracted from an OSN profile in several ways. For example, by scrapping: parsing the web pages that are exposed by the OSN services, such as the time-line in Facebook; or via the OSN APIs. Some OSN APIs provide convenient way to collect information about profiles, including their LOF. A prominent example of OSN API is the Facebook Query Language (FQL). Information extraction from OSN profiles is an active field of research [36, 37, 43, 51, inter alia], and plays an important role in web tracking [10, 22].

In this work we assume that the information extraction aspects are encapsulated in two possible OSN queries:

- ***IsLead(v).*** This is a binary query which checks whether the given profile v is a lead or a non-lead. In our implementation *IsLead(v)* checks whether the target is in the LOF of v:
$$IsLead(v) = \begin{cases} True & target \in LOF(v) \\ False & \text{Otherwise} \end{cases} \tag{1}$$
- ***Acquire(v).*** This query downloads all publicly available data from the OSN about the given profile v, including its LOF as well as all the information in its profile.[1]

In our model, *IsLead()* is considered a "light weight" operator with relatively small cost, while *Acquire()* is considered a heavy operator. Consequently, our algorithms below use *IsLead()* quite often and use *Acquire()* more conservatively. In particular, in our model, *Acquire()* can be applied only to profiles that are known to be either leads or non-leads (i.e., not to potential leads), i.e., to profiles that we already performed an *Islead()* query on.

Some profiles are called *initial leads*. These profiles are known a-priori to be leads based on previous knowledge about the target. Initial leads can, for example, be found manually using traditional intelligence techniques.

For example, consider a scenario where the target is a pedophile and the police is trying to obtain leads to capture him. It is often the case that the police has additional sources of information, e.g., by applying classic detective work that allow them to identify several OSN profiles as related to the target. However, such initial leads are expected to be very hard to obtain, and we assume that there are only few initial leads for each target.

[1]Note that the acquire action does not include sophisticated information extraction methods: it simply downloads all data and extracts the LOF. As mentioned above, further analysis of this data may be done by a human analyst.

If a profile v is not an initial lead, then we assume that its LOF is initially unknown. Consequently, since the target's LOF is also unknown, then it is not known if v is a lead before querying the OSN (by applying *Islead(v)* or *Acquire(v)*). We refer to such profiles as *potential leads* as they might later turn into leads or into non-leads. *Islead()* is applied only to potential leads. Applying *IsLead* to a potential lead marks it as either a lead or a non-lead.[2]

Among other things, the heavy *Acquire(v)* query extracts and examines $LOF(v)$. This results in the discovery of additional, previously unknown, potential leads as well as the discovery of links to previously known profiles (which can be leads, non-leads, or potential leads). A *TONIC process* is an OSN exploration process that searches the OSN for leads using the *Islead()* and *Acquire()* queries.

A profile that is not an initial lead has the following life cycle in a TONIC process. First, it is a potential lead. Then, if an isLead() query is applied to it, it is either discovered to be a lead or a non-lead. Later, a profile (either lead or non-lead) may be acquired, revealing all its LOF.

Definition 3 (TONIC Problem) The input to TONIC is $target$ and a set of *initial leads* (denoted by L_0). The TONIC problem is to find leads by applying the TONIC process under different problem settings.

2.1 TONIC problem settings and objectives

Definition 3 defines the TONIC problem in a general way. Informally, all TONIC objectives considered in this work aim at minimizing the number of *IsLead()* and *Acquire()* queries applied while maximizing the number of leads found. Next, we define a number of objectives that will be used in the paper.

2.1.1 Anytime objective

The first objective we consider is to maximize the number of leads found until the algorithm is halted.

Definition 4 (Anytime objective) Given a set of initial leads L_0 the objective is to find as many leads as possible until the TONIC process is stopped, assuming that the TONIC process can be stopped after any number of queries (each of them is either *Islead()* or *Acquire()*).

Executing the *Acquire()* and *IsLead()* queries requires both computational resources and network activity. In addition, most OSN services limit automatic web scrapping attempts as well as massive exploitation of their API. These possible limitations on the number of executed queries are the main motivation behind the anytime objective of finding as many leads as soon as possible, i.e., with fewest possible queries.

2.1.2 Max net-gain objective

Next, we consider scenarios in which the *cost* of *Acquire()*, the *cost* of *IsLead()*, and the *reward* of finding a lead, can all be quantified, denoted by $C_{Acquire}$, C_{IsLead}, and R_{Lead} respectively.[3]Typically, the reward of finding a lead is larger than the cost of *Acquire()*,

[2]In some OSNs, profiles can block their LOF, so that it is not possible to perform the *IsLead()* query on them. For our purposes, they will be regarded as non-leads, since we cannot verify that they are leads.

[3]Sophisticated TONIC applications may assign rewards that decay with time or are dependent on the amount of information about the target that can be extracted from the lead. We focus on a simpler reward model in which the reward of finding a lead is constant.

otherwise there is no point in acquiring any lead. Additionally, it is usually the case that the cost of *Acquire()* is much larger than the cost of *Islead()* due to the storage requirements, increased interaction with the OSN, and the execution of IE tools. Taking C_{IsLead}, $C_{Acquire}$, and $R_{Acquire}$ into consideration, the second objective function we consider is as follows.

Definition 5 (Max net-gain objective) Given a set of initial leads L_0, reward R_{Lead}, C_{IsLead}, $C_{Acquire}$ the objective is to maximize the total net gain (sum of rewards minus sum of costs) of the TONIC process.

Other problem settings and objectives for TONIC can also be formulated. For example, finding a fixed number of leads with minimum acquisitions. The algorithm and heuristics proposed in this paper are expected to be effective for all similar objectives. In this paper, however, we focus on the two problem settings listed above – anytime (Definition 4) and max net-gain (Definition 5). The first part of the paper focuses on the anytime objective and Section 7 deals with the max net-gain objective.

3 Related work

Analyzing and searching the social web was previously addressed in the contexts of intelligent crawling, network completion, and general search in an unknown graph. Next, we briefly review prior efforts in these related problems and discuss their connection to TONIC.

3.1 Intelligent crawling

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 web crawling* [2, 8, 11]. There is a large body of work on intelligent web crawling, and work in this subject differ in the techniques being used, the data that is available about the crawled web, and, importantly, the purpose for which the crawling is performed.

3.1.1 Focused crawling

A very common task for web crawlers is to uncover large portions of the OSN and index them. This is fundamentally different from our problem – TONIC – in which we wish to retrieve information about a specific target and avoid further crawling. However, some prior work also considered the problem of *focused crawling*, where one wants to find all pages relevant to a specific topic [20, 40], e.g., for online query answering purposes. We provide here a non-comprehensive list of the techniques used in prior work for focused crawling and discuss their applicability to our problem.

Many attributes can be taken into consideration while implementing a focused web crawler. Example of such attributes that have shown to be helpful is the URLs of the nodes that were discovered but not visited yet, 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 that have already been crawled. Naturally, a combination of such features yields the most effective focused crawling [2]. It was also shown that reusing the learned information (as a starting point for the next crawl) as well as performing an initial sampling of random web pages are also beneficial [11].

Most focused crawlers are based on relevance of the webpage content to the topic being searched. Common measures of webpage relevance are TF/IDF, cosine similarity, and others

[20, 40]. Some works employ machine learning to classify webpages according to their content [12]. Wang et al. proposed a focused search approach that is based on the words surrounding the prioritized hyperlink [53]. Topology-wise these focused crawlers prioritize web pages using the HITS algorithm [32] and the PageRank centrality measure [3]. All approaches agree that URLs (pointers to webpages that were not crawled yet) need to be ordered intelligently in order to increase the relevance of obtained documents [16]. Importantly, in this work we do not consider the textual content of the visited profiles, and thus the approach we propose does not require preliminary content extraction (neither in form of the webpage snippet nor in form of the text surrounding the hyperlink). Content and topology analysis are orthogonal approaches that can complement each other in future developments. Thus, we could not directly use the techniques used in these works. However, several of the works mentioned above employ Bayes rules to aggregate evidence on the relevance of URLs. We use a similar approach in the $BysP$ heuristic (see Section 5) to aggregate topological evidence.

Prior work also studied how to perform focused web crawling with several agents in parallel. A dynamic search method that have been proposed in such a setting is called *Fish Search* [19]. 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). Shark Search [29] and Improved Shark Search [14] are improvements of Fish Search in which additional features were considered, such as topic description, textual relevance, URL and link relevance. In this paper we do not consider a multi-agent version of the TONIC problem, and leave it to future work.

3.1.2 Network sampling

Another similar task for an intelligent crawler is to *sample* an OSN in order to estimate its properties. For example, Gjoka et al. used variants of random walk to estimate Facebook's degree distribution and average degree [28], and Kurant et al. used a form of stratified sampling to estimate the percentage of Facebook users that attended college [34].

Crawling to obtain a representative sample of an OSN is different from our problem in that we do not aim to estimate a property of the OSN, but to find specific profiles that have a specific property – are friends of a specific target profile. Moreover, the number of profiles visited in graph sampling methods is often in the order of hundreds of thousand profiles, while in TONIC we aim at visiting the fewest possible nodes.

In general, TONIC is different from all prior work on intelligent web crawling on OSNs mainly in that we wish to retrieve information about a specific target and avoid further crawling, while only considering network topology. The only exception, to the best of our knowledge, is our prior work in which we proposed general technique for intelligent crawling that is based on a unique variant of the famous Multi-Arm Bandit problem [9]. The technique was applied to two intelligent crawling tasks: finding communities in an OSN and TONIC. Importantly, the goal of our work there was to propose a general intelligent crawling method that is not specifically designed for TONIC. Indeed, the results there show results that are comparable to the TONIC-specific BysP heuristic we present in this paper. As we show experimentally in Sections 5.2 and 6, BysP can be further improved by the heuristics and algorithms we proposed.

3.2 Network completion

Network information that was obtained by crawling is often incomplete. Two relevant problems that deal with network completion are profile's attributes prediction and link prediction. *Profile's attributes prediction* aims at predicting demographic properties or other attributes (e.g., college graduation year and major) of profiles 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 [4, 24, 41]. TONIC is reminiscent of profile attribute prediction, in the sense that the attribute that should be predicted in TONIC is whether a profile contains information about (or linked to) the target. But in TONIC we aim to find – with certainty – leads while in profile attribute prediction the task is to *estimate* the existence of various profile attributes. Nonetheless, the heuristics we propose were inspired by prior work on attribute prediction and specifically the work of Mislove et al. [41]. Mislove et al. observed that profiles with similar attributes are grouped into *communities* – highly connected subgraphs of the social network. Consequently, in order to find all profiles sharing similar attributes one can crawl the *communities* that a given profile is a member of.

Link prediction is a variant of network completion problem where given a partially known network one should determine whether or not there should be a link between two given nodes. Fire et al. used a variety of topological features to learn an accurate link prediction classifier [24, 27]. They also ignored non-topological data like posts, pictures, keywords and other content related to the social network profiles. The resulting link prediction classifier was evaluated on five social network datasets: Facebook, Flickr, YouTube, Academia.edu, and TheMarker, and was shown to be very accurate. Indeed, we have found that one of the features they proposed – the *Friends Measure* – is also an effective heuristic for prioritizing profiles that should be crawled in TONIC. See Section 5.1.6 for additional details on this measure.

Similar to profile attribute prediction, some works (e.g. [50]) focused on predicting the properties of links in *heterogeneous social networks*, i.e., networks in which there are more than one type of connection between two profiles. Unless we need to find specific kinds of leads (e.g. those that have tagged the target in a photo or commented on his posts but are not his friends on Facebook) this type of link prediction is not suitable for TONIC. However, information collected in a heterogeneous network can be used to build stronger classifiers for link prediction [18, 21]. Since different types of relationships are usually created by different social processes they carry more information than flat networks.

Li et al. [37] studied how to collect reconnaissance in a partially observed OSN with multiple crawlers. This means identifying parts of the OSN, where a benefit function associates a reward to the newly found parts of the OSN.

3.3 Heuristic search in an unknown graph

TONIC can be formalized as a *search problem in an unknown graph* (see Section 4). A search problem in an unknown graph is a search problem in which 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. The main challenge when searching in an unknown graph is to solve the problem while minimizing the exploration cost. TONIC fits into this category as we want to acquire profiles such that the most information about the target will be found while minimizing the costs of our *IsLead* and *profile acquisition* operators.

There are several previous works on developing algorithms for searching in an unknown graph. Felner et al. [23] proposed the Physical A* algorithm for solving the shortest path problem in an unknown *physical* graph. In a physical graph, exploration is done by a physical agent that needs to physically traverse the graph in order to explore new nodes. The exploration cost being minimized is the distance traveled by the agent. In contrast, we deal with exploring an OSN and exploration is done by accessing the OSN, which is not related to a physical distance measure. A setting that is more similar to searching an OSN was discussed by Stern et al. [46] that proposed algorithms for finding a k-clique in an unknown (but not physical) graph and any other specific pattern [47], where the cost of exploration was constant per node. They employed a best-first approach that is similar to the one proposed in this work and in fact we were inspired by their approach.

That being said, the problem we address in this work – TONIC – is fundamentally different. First, we do not search for a specific pattern, but for nodes with a specific property – profiles that are leads to the target. Second, Stern et al. [47] proposed heuristics to guide the search for the k-clique problem, but these heuristics do not transfer to the TONIC problem of finding leads. In this paper we present several very effective heuristics that are specifically designed for TONIC. Indeed, these heuristics exploit the fact that we are searching for leads and that the search is done over an OSN.

4 TONIC as search in an unknown graph

Algorithm 1 BFS for TONIC

```
Input: $target$ the target
Input: $L_0$, the set of initial leads
Output: $L$, the leads found
1  $L \leftarrow L_0$, $PL \leftarrow \emptyset$, $NL \leftarrow \emptyset$, $CKG \leftarrow$ the subgraph of $G$ induced by $L$
2  foreach $l$ in $L_0$ do
3      $LOF \leftarrow$ Acquire(l)
4      Add $LOF$ to $OPEN$ and to $PL$
5  while $OPEN \neq \emptyset$ do
6      $best \leftarrow$ ChooseBest($OPEN$)
7      if $best \in PL$ then
8          Remove $best$ from $PL$
9          if IsLead(best) then
10             Add $best$ to $L$
11             $LOF \leftarrow$ Acquire(best)
12             Update $CKG$, $PL$, and OPEN
13         else
14             Add $best$ to $NL$
15             Reinsert $best$ to OPEN
16     else
17         $LOF \leftarrow$Acquire(best)
18         Update $CKG$, $PL$, and OPEN
19 return $L$
```

Next, we preset a general framework for solving the TONIC problem. The fundamentals of this framework originate from prior work on solving search problems in unknown graphs (described in Section 3). The unknown graph being searched in TONIC is the topology of the OSN, denoted $G = (V, E)$, where V is a set of OSN profiles and E is a set of links such that $(v_1, v_2) \in E$ if $v_1 \in LOF(v_2)$.

A best-first search approach is taken, outlined in Algorithm 1. Throughout the search, the following data structures are maintained:

1) L, NL and PL, which are the 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 by the search process (line 1)
3) $OPEN$, a list of 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. Similarly, the CKG is initialized such that its vertices are the initial leads and their potential-leads neighbors. The edges of the CKG are the links that connect the initial leads. In every iteration, a single profile (*best*) is chosen from OPEN (line 6). The exact choice if *best* is done via the different *ChooseBest()* heuristics that we describe in later sections.

If $best$ is a potential lead, then it is removed form PL and an *IsLead($best$)* query is performed (line 9) to reveal if it is a lead or a non-lead. If $best$ is found to be a lead, then it is immediately acquired, as finding leads is always desirable (line 11). Otherwise, if *best* is found to be a non-lead, it is reinserted to $OPEN$ to be considered for acquisition at a later stage (line 15). 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 known non-lead, then it is acquired (line 17). When *best* is acquired (lines 12 and 18), CKG, PL, and $OPEN$ are updated as follows: (1) nodes and edges corresponding to the neighbors of *best* and the links to them are added to CKG, and (2) neighbors of *best* that were not previously part of CKG are added to PL and to $OPEN$. The TONIC process is halted either by the user or when all the nodes in the OSN has been acquired. When this happens the set of found leads is returned (line 19).

The pseudo encapsulates both RTF and ETF. Lines 15 to 18, which are preceded by the letter E, are executed only for the ETF framework. As detailed below, RTF will never choose a non-lead as $best$ so these lines may be deleted in an RFT implementation.

Figure 1 presents an example execution of Algorithm 1. The target is marked by T, and there is one initial lead a. After acquiring a we reveal b and c as potential leads (step 2). In the next step, *Islead()* is performed on b revealing that it is a non lead (step 3). Then, *Islead()* is performed on c revealing that it is a lead. c is then immediately acquired, discovering the potential lead d (step 4).

5 The Restricted TONIC Framework (RTF)

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

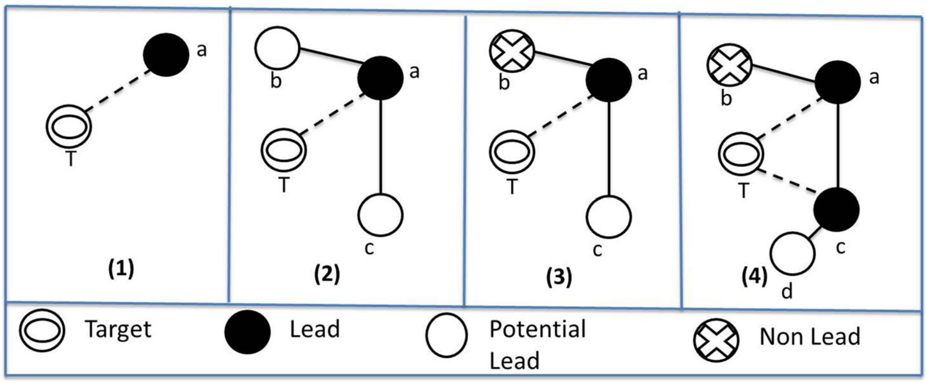


Figure 1 An example of a run of Algorithm 1. The figure shows the CKG in the different stages

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 [5, 25, 39]. Thus, it follows that friends of leads (i.e., profiles that appear in leads' LOFs), are more likely to be leads than randomly selected OSN profiles.

The Restricted TONIC Framework (RTF) builds on this understanding, and focuses the search for leads by only applying the *Acquire()* query to profiles that are known to be leads. Non-leads are never acquired. RTF may be implemented with Algorithm 1 setting the $ChooseBest$ function to never choose a non-lead. In particle, with RTF we may simplify Algorithm 1 – profiles that are found to be non-lead are discarded and are not re-inserted into $OPEN$ (line 15). As a result, $OPEN$ only contains potential leads and lines 15–18 are redundant (they are marked with E for extended framework).

Also, a single *expand* action can be defined, which performs *IsLead()* on *best* and acquires if it is a lead.

5.1 RTF heuristics

We now provide a number of heuristics for choosing which potential lead to expand next in RTF.

A Key to the efficiency of RTF 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 *RTF heuristics*. As a baseline RTF 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 1 illustrates such an expansion order. T is the target profile and there is a single initial lead a. After profile a is acquired (*Acquire(best)* in Algorithm 1 line 6), 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 the 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. By contrast, the heuristics provided next are based on analyzing the CKG in different ways.

5.1.1 Clustering coefficient heuristic

Nodes in OSNs tend to form tightly connected clusters in which most people are friends of each other. This phenomenon is quantified using the notion of *local clustering coefficient* [54].

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.

Let $CC(n_i)$ be the local clustering coefficient of node n_i, and let $N(x)$ be the neighborhood of x, then

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

A profile 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.

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 $L(pl)$:

$$CC(pl) = \frac{2 \cdot |\{(u, v) \in E_{CKG}|u, v \in L(pl)\}|}{|L(pl)| \cdot (|L(pl)| - 1)} \tag{3}$$

In RTF, the neighborhood of a potential lead consists of only the previously acquired leads. Thus $L(pl) = N(pl)$ and the above formula is exactly the 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 2. 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 L1 and L4) out of one possible

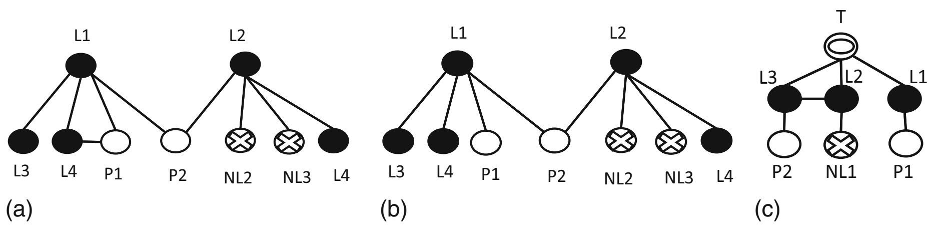


Figure 2 Examples used to illustrate the RTF heuristics. **a** is used for the Clustering Coefficient (CC) heuristic, **b** for Known Degree (KD), and **c** for Friends Measure (FM)

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 Known-degree heuristic

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 [7], 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 [1].

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 [47]. In RTF, 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 2b. $KD(P1) = 1$ (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 heuristics

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 Figure 3a. 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 in the sense that potential leads connected to them are more likely to be leads.

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, non-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 $pf(m) = \frac{L(m)}{L(m)+NL(m)}$.[4]

[4]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.

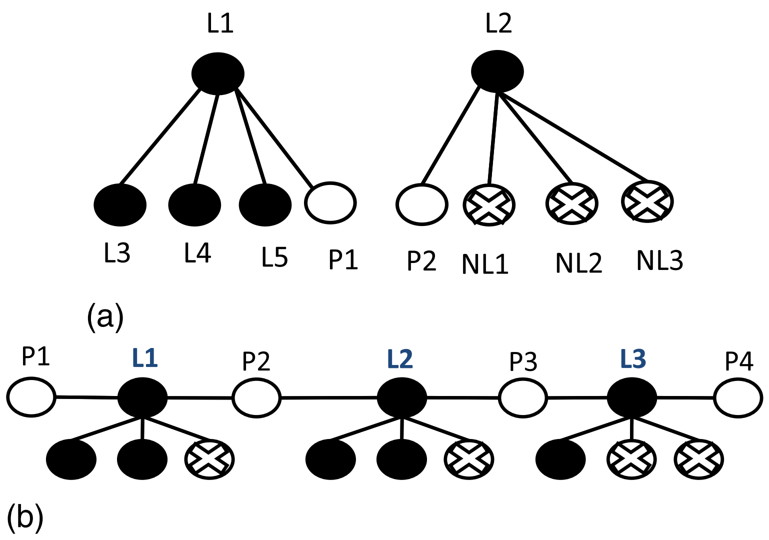


Figure 3 Examples illustrating the Promising Heuristic. **a** is used to demonstrate how to compute the Promising Factor and **b** is used to demonstrate how to aggregate promising factor values

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 3b. There are four potential leads, $P1$, $P2$, $P3$, and $P4$, and three leads $L1$, $L2$, and $L3$. The promising factor of leads $L1$, $L2$, and $L3$ is $\frac{2}{3}$, $\frac{2}{3}$, and $\frac{1}{3}$, respectively. The $MaxP$ values for potential leads $P1$, $P2$, $P3$, and $P4$, are $\frac{2}{3}$, $\frac{2}{3}$, $\frac{2}{3}$, and $\frac{1}{3}$, respectively, while the $AvgP$ values for these potential leads are $\frac{2}{3}$, $\frac{2}{3}$, $\frac{1}{2}$, and $\frac{1}{3}$. Thus, using $MaxP$ will result in choosing first one of the potential leads $P1$, $P2$, and $P3$ (but not $P4$, which has a smaller $MaxP$ of $\frac{1}{3}$). In contrast, using $AvgP$ will not allow choosing $P3$ first, and either $P1$ or $P2$ will be expanded first (since both have $AvgP$ of $\frac{2}{3}$ while $P3$ and $P4$ have an $AvgP$ of $\frac{1}{2}$ and $\frac{1}{3}$, respectively.

$MaxP$ 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.

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.

$$BysP(pl) = 1 - \prod_{m \in L(pl)} (1 - pf(m)) \tag{4}$$

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 3b.

The $BysP$ values of the four potential leads are $BysP(P1) = 1 - (1 - pf(L1)) = \frac{2}{3}$, $BysP(P2) = 1 - (1 - pf(L1)) \cdot (1 - pf(L2)) = \frac{8}{9}$, $BysP(P3) = 1 - (1 - pf(L2)) \cdot (1 - pf(L3)) = \frac{7}{9}$, and $BysP(P4) = 1 - (1 - pf(L4)) = \frac{1}{3}$. Thus, $BysP$ will choose to expand first $P2$. In contrast, both $AvgP$ and $MaxP$ may expand some other potential lead first ($P1$ for $AvgP$ and $P1$ or $P3$ for $MaxP$).

5.1.6 Friends measure heuristic (FM)

The TONIC problem bears some resemblance to the *link prediction* problem [38], where the goal is to predict whether two profiles are connected (see Section 3 for more details). 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*.[5] In fact, a notion very similar to KD was extensively used in link-prediction research, where it is known as the common-friend concept [38]. 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 [26]. Formally, the friends measure (fm) between profiles two profiles (u and v) is defined as follows:

$$fm(u, v) = \sum_{x \in N(u)} \sum_{y \in N(v)} \delta(x, y) \tag{5}$$

$$\delta(x, y) = \begin{cases} 1 & \text{if } x = y \text{ or } (x, y) \in E \text{ or } (y, x) \in E \\ 0 & \text{Otherwise} \end{cases} \tag{6}$$

We note that the Friends Measure is a special case of the Katz measure [31] in undirected graphs, and is somewhat similar to the well-known Jaccard's coefficient [27].

In RTF, 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 $fm(pl, target)$ requires knowing the neighborhood of

[5] A more comprehensive discussion on the relation between link prediction and TONIC is given in Section 3.

pl and *target* ($N(pl)$ and $N(\mathit{target})$, respectively). $N(pl)$ is not known before *pl* is acquired and $N(\mathit{target})$ is also unknown, otherwise we would know all the leads. Instead, we use the neighborhoods of *pl* and *target* in the CKG. As only leads are acquired in RTF, $N(pl)$ in the CKG is exactly $L(pl)$. Since we assume that leads are all neighbors of *target*, then $N(target)$ in the CKG is simply the set of known leads (L). Thus, the resulting Friends Measure of *pl* and $target$ is:

$$fm(pl, target) = \sum_{x \in L} \sum_{y \in L(pl)} \delta(x, y) \tag{7}$$

The heuristic that chooses to expand the potential lead *pl* the highest $fm(pl, \mathit{target})$ is called *FM*.

Figure 2c provides an example of the FM heuristic. P1 and P2 are two potential leads, and *target* is denoted by T. P1 has one mutual friend with the target (L1) and therefore $fm(P1, T) = 1$. In contrast, P2 also has one mutual friend with the target (L3), however the target's neighborhood and P2's neighborhood's contain an additional mutual friend (L2) and therefore $fm(P2, T) = 2$ Therefore, the FM heuristic will choose to expand node $P2$ before node $P1$.

5.1.7 Runtime complexity

Our focus in this work is to minimize the number of queries performed. Thus, the CPU runtime of our methods is of lesser importance. Nonetheless, all the proposed RTF heuristics can be easily implemented in low-order polynomial time. RND and FIFO simply require a suitable data structure (array or queue) to run in constant time. KD can be implemented by sorting the potential leads according to their degrees. The CC, FM and Promising heuristics requires runtime that is linear in the number of edges of the CKG. Thus, we do not expect computational runtime to be a problem in practice.

5.2 Experimental results

Next, we evaluate the performance of the different RTF heuristics. Most experimental results presented in this paper are on the Google+, but Section 9 presents highlights from our research on other other OSNs. Google+ is one of the largest social networks, having more than 540M registered users and 300M users that are active monthly (according to Wikipedia).

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. [24] 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 RTF 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.

The search for leads for each target was executed using RTF using the following heuristics: RND, FIFO, CC, KD, AvgP, MaxP, BysP, and FM. Three friends of every target were randomly chosen as the initial leads. The search continued until *all* potential leads were expanded.

We analyzed the obtained results and evaluated the different TONIC heuristics with respect to the anytime objective function (Definition 4). Comparing anytime algorithms is usually done by comparing their *performance profile* [55]. The performance profile of an anytime algorithm is a plot showing the solution quality as a function of the algorithm runtime. In our case, the solution quality is the number of leads found, and the algorithm runtime is the number of queries performed. This is shown in Figure 4: the x-axis of is the percentage of potential leads queried so far (by *Islead()*) out of the total number of potential leads reachable from the initial leads, and the y-axis represents the fraction of leads found out of all existing leads, averaged over all the 100 targets in our data set. We set the x-axis here as the percentage of potential leads queried so far and not the actual number of queries for the following reasons. First, to allow averaging over targets with significantly different number of performed queries. Second, since this experiment is for RTF, we did not count the number of *Acquire()* queries as those are only applied to leads (and never to non-leads). In the figures presented later for ETF, we show the exact number of queries (and not percentage) and also consider the *Acquire()* queries. Note that since RTF limits the search to only acquire leads, not all leads are reachable from the initial leads, and thus the y-axis does not reach 100% even after all potential leads are checked.

The result in Figure 4 show 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.

6 Extended TONIC framework (ETF)

As seen in the results above, RTF may miss 30% of the leads. These 30% of leads are not reachable from the set of initial leads by a path which contains leads only. In this section, we relax the restriction imposed by RTF, allowing non-leads to be acquired. We refer to this

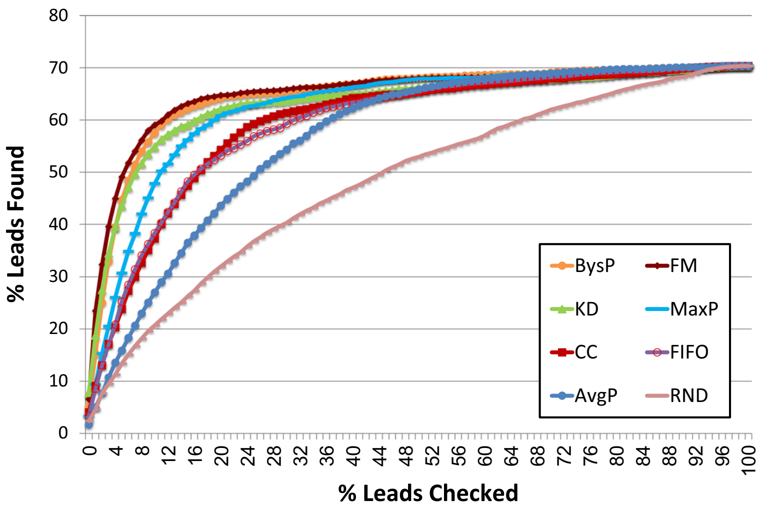


Figure 4 % of leads found vs. % of potential leads checked

TONIC framework as the Extended TONIC Framework (ETF). The scope of the search in ETF is larger than RTF, enabling reaching a larger set of profiles and potentially finding more 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, RTF is actually ETF(0), since only leads are acquired, while running Algorithm 1 unrestricted is in fact ETF(∞).

6.1 Reachable leads in ETF

The choice of the n parameter in ETF affects the number of leads that can be reached. Another factor that affects the number of reachable leads is the number of initial leads. Increasing both parameters (n and the number of initial leads) is expected to increase the number of reachable leads. This is demonstrated in Table 1, which shows the average percentage of reachable leads as a function of these parameters (averaged over all the targets in our data set). We use the term *tier* to refer to the set of profiles that are reachable with ETF(n) for a given number of initial leads. The table columns represent the number of tiers used, and the rows are the number of initial leads (rows). Note that since the initial leads are expected to be very hard to obtain (e.g., obtained via manual labor), we focused our analysis on a relatively small numbers of initial leads.

First, consider the impact of the number of initial leads. From a single initial lead the TONIC process can reach a significant fraction of all leads (more than 60% in all cases). Every additional initial lead increases the fraction of reachable leads. The second initial lead increases the fraction of reachable leads by a margin of 6.6% in tier 0 and approximately 4% in higher tiers. This margin decreases with each additional initial lead, and we observe only a marginal advantage for adding more than 3 initial leads. As an operational guideline we suggest obtaining initial leads with different kinds of acquaintances with the target. For example, one family member, one co-worker, and one blog fan.

With every additional tier the number of profiles in the corresponding tier grows, and the search will be able to reach more leads. The results in Table 1 show this trend very clearly. The percentage of reachable leads increases significantly when considering also profiles from tier 1. For example, with 3 initial leads 70.51% leads are reachable in tier 0, while an additional 17% of the leads become reachable by considering tier 1 profiles too. Thus, in $ETF(1)$ we can find more leads than in RTF. However there is not much difference between the number of leads found in higher tiers (tier 2 and 3). Therefore we focus in the rest of this paper on ETF(1).

Table 1 % of leads reachable in different tiers from different numbers of initial leads. Note that tier 0 is actually RTF

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

6.2 ETF heuristics

A key difference between RTF and ETF is that $OPEN$ in RTF contained only potential leads while in ETF $OPEN$ can also contain non-leads. This is because in RTF a potential lead found to be a non-lead is discarded, while in ETF(n) discovered non-leads are re-inserted into $OPEN$. These re-inserted profiles are later considered for expansion if they are not too far from a known lead (where “too far” is with respect to the n parameter). Note that the distance of non-leads to known leads can change as the search progresses and new leads are discovered.

Figure 5 shows an execution of ETF. Stages (1)-(4) are similar to the RTF (see 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.

Since the number of reachable profiles (and reachable non-leads) with ETF is much larger than with RTF, ETF can potentially perform worse than RTF. 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.

6.2.1 EFIFO heuristic

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 5 illustrates such an expansion order. Figure 6a shows the performance of EFIFO in our data set (described in Section 5.2)

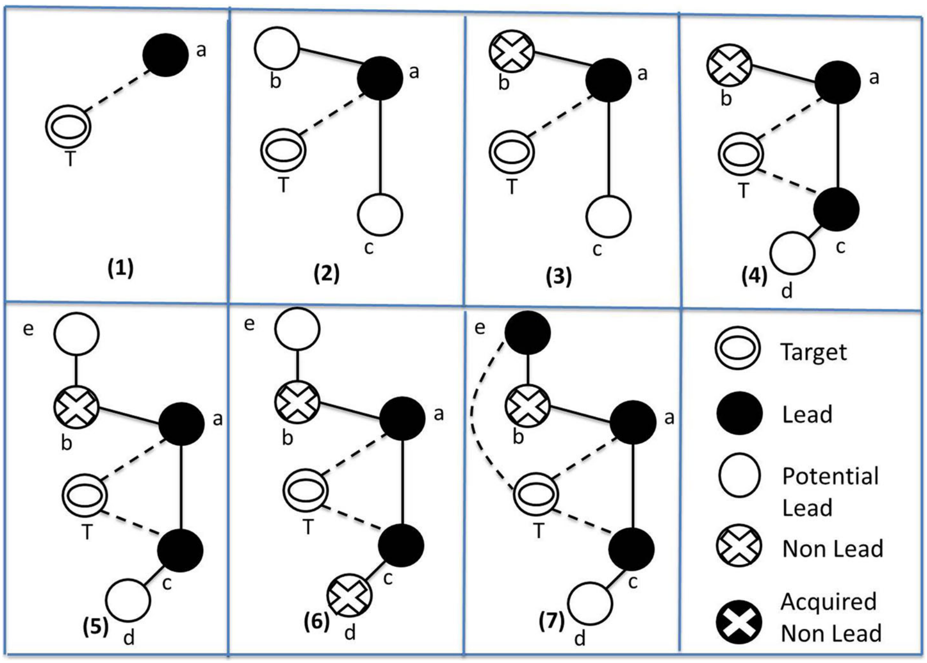
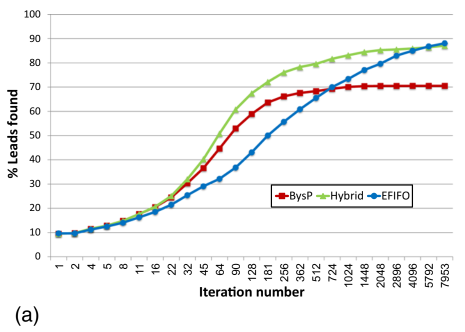


Figure 5 An example of using ETF to search for leads



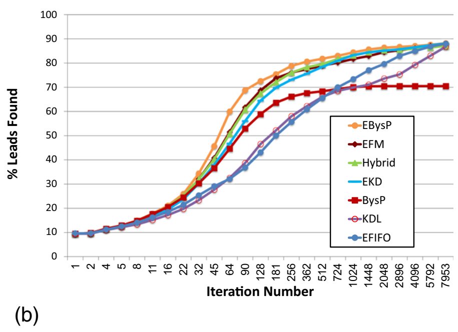


Figure 6 Performance of ETF heuristics. The x-axis is number of executed iterations of Algorithm 1. The y-axis is the percentage of leads found by *IsLead()* up to that point. **a** compares Hybrid vs. BysP(0) and EFIFO(1). **b** compares all the heuristics together

versus that of BysP, the best RTF 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.[6] As can be seen, EFIFO discovers leads slower than BysP but eventually reaches more leads.

6.2.2 Hybrid heuristic

To enjoy the complementary benefits of BysP, which finds leads fast by effectively focusing on clusters of leads and the extended reachability of EFIFO, we propose an adaptive *hybrid* heuristic that starts 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*$(\cdot)$ queries allowed since the last lead was found. If U unsuccessful *IsLead*$(\cdot)$ 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. Initially, set U to some constant. Whenever a lead is acquired (with BysP), U is set to be the number of *IsLead*$(\cdot)$ queries done so far. For example, if a lead was found in the third *IsLead*$(\cdot)$ operation then U is set to 3 and if no lead will be found in the next 3 *IsLead*$(\cdot)$ operations then we will switch from the BysP heuristic to FIFO.

Empirical evaluation (see Figure 6a) shows that $hybrid$ is able to outperform both BysP and EFIFO, suggesting that $hybrid$ switches heuristics when EFIFO starts to outperform BysP. The x-axis in Figure 6a corresponds to the number of iterations of the main loop in Algorithm 1 (lines 5–18). In each iteration one node is removed from OPEN and is queried according to the Algorithm 1.

6.2.3 Variants of the known degree heuristic

KD, described above for RTF, expands the potential lead with the highest degree in the CKG. Next we discuss how to adapt it to ETF. Let $EKD(p)$ 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

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

are acquired in RTF, there is at least one lead in every edge of the CKG. Consequently, in RTF $EKD(p) = KDL(p) = KD(p)$ for every potential lead p. In ETF, $EKD(p)$ 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 EKD and KDL, each expanding the node (either potential lead or non-lead) in $OPEN$ with the highest $EKD(\cdot)$ and $KDL(\cdot)$, respectively.

Figure 7a depicts a CKG that demonstrates the difference between EKD 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 $EKD(P1) = 4$, $EKD(NL1) = 1$, and $EKD(P2) = 3$, while KDL will expand P2 since $KDL(P2) = 3$, $KDL(P1) = 2$, and $KDL(NL1) = 1$.

The intuition behind EKD and KDL differ. EKD 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 EKD performs significantly better than KDL in ETF(1) (as shown in Figure 6b explained in the experimental section).

6.2.4 BysP and FM heuristics for ETF

BysP and FM were the best performing RTF heuristics (see Section 5.2), and they can be easily adapted for ETF. The key difference between RTF and ETF is that non-leads can be acquired, and thus while in RTF neighbors of profiles that are considered for expansion (either *Islead()* or *Acquire()*) 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 RTF 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 7b. $OPEN$ 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

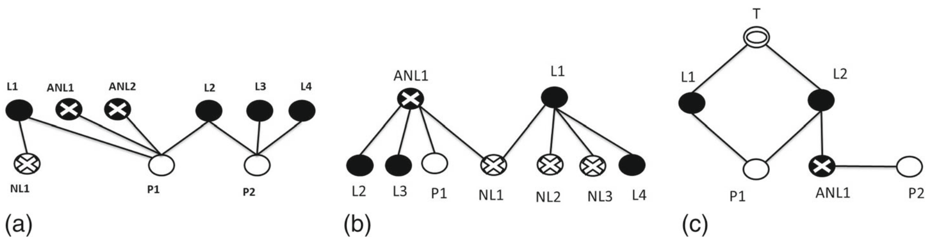


Figure 7 Examples used to illustrate the ETF heuristics. **a** is used for EKD and KDL, **b** is used for EBysP, and **c** is used for EFM

$\frac{1}{4}$, 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 7c 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.

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 RTF experiments earlier (Section 5.2). However now, the relevant neighborhood of each target is larger since it contains an additional tier.

Figure 6 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 RTF 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.

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 Maximizing the net gain

Up until now, we focused on the anytime objective (Definition 4 represented by the x-axis). The experiments showed given B operations, how many leads have we found?

In this section we analyze the proposed TONIC frameworks and heuristics from the perspective of the second objective function we proposed, which aims at maximizing the net gain (Definition 5). The net gain is roughly related to the steepness of the slopes in Figure 6b. 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 6b 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.

7.1 Net gain results

Figure 8 demonstrates this, showing the net gain (y-axis) as the search progresses (x-axis, number of *Acquire()* queries executed) for BysP and EBysP. In this experiment we set the cost of *Islead()* to be zero, the cost of *Acquire()* to one, and the reward of finding a lead is 40.

We chose to evaluate only the BysP and EBysP heuristics because BysP is the best heuristic for RTF(=ETF(0)) and EBysP is the best heuristic for ETF(1). First, observe that the

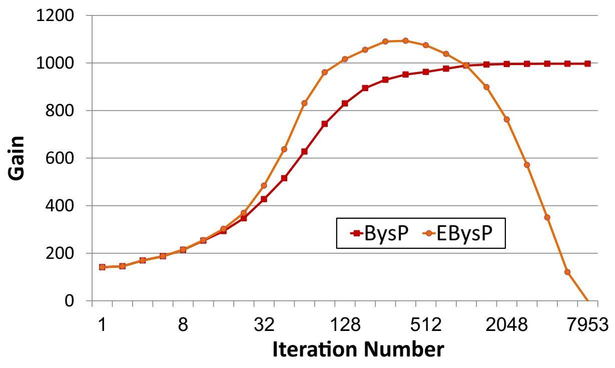


Figure 8 Average net-gain as a function of the number iterations for $R_{Acquire}$=40 and $C_{Acquire}$=1)

gain of BysP does not decrease throughout the search. This is because RTF 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 RTF cannot decrease unless either $C_{Acquire} > R_{Acquire}$ or C_{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 8 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 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. We propose a method for choosing when to stop crawling in Section 7.2.

Table 2 shows potential of having a perfect stopping mechanism for choosing when to stop, showing the maximal gain achievable for BysP and EBysP, for different values of $R_{Acquire}$ and assuming that $C_{Acquire}$=1 and $C_{IsLead} = 0$. For every value of $R_{Acquire}$ we marked in bold the results for the algorithm that provided the highest net gain. BysP reaches a maximal gain higher than EBysP when $R_{Acquire} \leq 3$, and this reverses when $R_{Acquire} \geq 4$. This is reasonable because larger $R_{Acquire}$ 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 9 shows the difference between their gains (vertical-axis $\uparrow$) as a function of $R_{Acquire}$ (horizontal-axis $\nearrow$) and the number of iterations (depth-axis $\nwarrow$). For example, the solid line

Table 2 Average net-gain for different $R_{Acquire}$ values

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

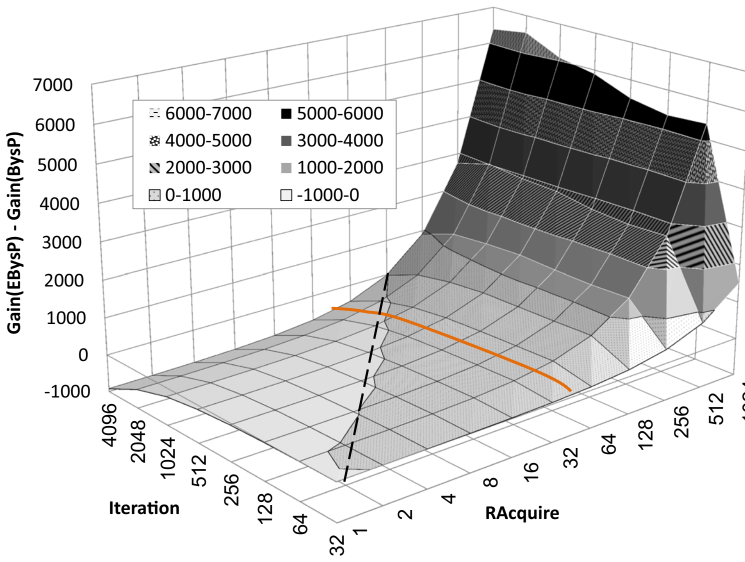


Figure 9 The difference in net gain between BysP(0) and BysP(1)

at $R_{Acquire} = 40$ represents the difference between gains of EBysP and BysP as depicted in Figure 8. The dashed line represents the relation between $R_{Acquire}$ and iterations where the gains of both heuristics are equal. EBysP (using ETF(1)) benefits from higher rewards and suffers for longer executions, while BysP (using RTF) is a valid choice if $R_{Acquire}$ is not much larger than $C_{Acquire}$ and we expect a long execution. Thus, the choice of between BysP and EBysP could be determined upfront if one knows the number of iterations the search will be run, $R_{Acquire}$, and $C_{Acquire}$.

7.2 Identifying when to stop

So far, the discussion focused on how to choose the best profile to query in every step. As illustrated in Figure 8, in order to maximize the net gain (Definition 5) it is extremely important to know when to stop the TONIC process, since every query incurs a cost and the rewards are limited.

A baseline stopping condition for a TONIC process is to stop after a fixed number of queries. However, the maximal net gain as well as the iteration at which it is achieved varies across different TONIC processes, depending on the total number of leads and the structure of the social network. Moreover, an optimal stopping condition must consider $C_{Acquire}$, C_{IsLead}, and $R_{Acquire}$. Next, we explore two stopping condition for deciding when to stop the TONIC process that considers this costs and reward values. These stopping condition are designed to take advantage of available knowledge about TONIC processes that were executed in the past.

7.2.1 Learning a zero rule stopping condition

Let $\mathcal{T}$ be a set of n TONIC processes that were executed in the past. For every TONIC process $T \in \mathcal{T}$ and every iteration i of that TONIC process, let $L(T, i)$, $NL(T, i)$, $PL(T, i)$,

$Acquire(T, i)$, and *IsLead(T,i)* be the number of leads, non-leads, potential leads, *A*cquire queries, and *I*sLead queries, respectively, found/performed during the first i iterations of T.

Given the cost and reward values the net-gain of T at every iteration i is:

$$NG(T, i, R_{Acquire}, C_{Acquire}, C_{IsLead}) = R_{Acquire} \cdot L(T, i) - \big(C_{Acquire} \cdot Acquire(T, i) + C_{IsLead} \cdot IsLead(T, i)\big)$$

When $R_{Acquire}$, $C_{Acquire}$, and C_{IsLead} are clear from context, we use the shorthand notation $NG(T, i)$ to denote the corresponding net gain for stopping in iteration i when running T.

The first stopping condition we propose is called *Z*eroR and is based on a *zero rule* classifier. A zero rule classifier for Binary classification problems is a classifier that ignores all the features of the classified instance and outputs the class of the majority of instances in the training set. Similar to a zero rule classifier, *Z*eroR stops every TONIC process after a fixed number of iterations, regardless of any knowledge gained during the current TONIC process. This fixed number of iterations is computed according to the set of available TONIC processes ($\mathcal{T}$), such that stopping the TONIC processes in $\mathcal{T}$ in that iteration would have given the maximal net gain, on average. Formally, *Z*eroR stops at iteration I_{best}, which is defined as follows;

$$I_{best} = \arg\max_{i} \sum_{T \in \mathcal{T}} NG(T, i)/|\mathcal{T}| \tag{8}$$

The *Z*eroR stopping condition has a clear limitation: it stops all future TONIC processes at the same iteration, following a "one size fits all" approach. Importantly, it ignores the information collected during the TONIC process about the target and its surrounding network. This information includes the CKG, the number of acquired leads and non-leads, and the number of potential leads. Indeed, we observed empirically that the structure of the CKG of different targets can be very different.

7.2.2 Learning a dynamic stopping condition

The second stopping condition we propose, denoted as *L*earnDynamic, models the dilemma of when to stop as a Binary classification problem and solves it via supervised learning [42]. Every instance being classified is a pair (T, i), where T is a TONIC process and i is an iteration. The two possible classes for an instance (T, i) are "stop" and "continue". An iteration i in a TONIC process $T \in \mathcal{T}$ is labeled as "stop" if the net gain of all following iterations is less than or equal to $NG(T, i)$ ($\forall_{j>i} NG(T, j) \leq NG(T, i)$) and is labeled as "continue" otherwise ($\exists_{j>i} NG(T, j) > NG(T, i)$).

To create a training set, we label all the iterations of all the TONIC processes in $\mathcal{T}$. The resulting training set has therefore $\sum_{T \in \mathcal{T}} |T|$ instances, where $|T|$ is the number of iterations in T.

Given this training set, we use Machine Learning algorithms to train a Binary classifier that accepts a TONIC process and an iteration i and outputs "stop" or "continue". The *L*earnDynamic stopping condition decides to stop according to this output.

In order to apply standard Machine Learning algorithms, we define a set of features to extract from each instance (T, i). We experimented with a wide range of features and have found the following to be most effective.

- **Leads acquisition rate.** This feature is the number of leads found per iteration in the past X iterations, where X is a parameter. Formally, for an instance (T, i), we define the leads acquisition rate feature for parameter X as $\frac{L(T,i)-L(T,i')}{i-i'}$ where $i' = \max(0, i - X)$. In our experiments we used two leads acquisition rate features, one with $X = 5$ and the other with $X = 10$.

- **Leads to non-leads ratio.** The ratio between the number leads and non-leads found so far – $L(T,i)/NL(T,i)$.
- **Leads to potential-leads ratio.** The ratio between the number leads and potential leads found so far – $L(T,i)/PL(T,i)$.

When the reward for getting a lead ($R_{Acquire}$) is not significantly larger than the query costs, it is worthwhile to stop the TONIC process early. In such cases, most instances in the trainings set will be labeled as "stop" creating an *imbalanced* training set. Similarly, when $R_{Acquire()}$ is very high compared to the query costs and number of profiles in the target's neighborhood, most instances will be labeled as "continue", again creating an imbalanced training set. Most supervised learning algorithms are designed for balanced datasets and may perform poorly when given imbalanced training set. Indeed, we observed this in a preliminary set of experiments (not reported).

We addressed this shortcoming using *under-sampling*, a known technique for addressing imbalanced training sets, where instances from the majority class are removed until both classes have the same number of instances. Instances from the majority class are usually selected uniformly at random. In this study, we chose the instances of the majority class that are closer to the decision boundary in order to improve the prediction accuracy. That is, if a TONIC process has 500 iteration, the first 100 labeled "continue" and the last 400 labeled "stop", then we only chose for our training set the first 200 iterations. This TONIC-specific under-sampling method proved to be effective in our experiments.

Both stopping conditions (ZeroR and *L*earnDynamic) are designed specifically for the Max Gain TONIC objective (Definition 5). As such, they are directly affected by the reward and query costs values ($R_{Acquire}$, $C_{Acquire}$, and C_{IsLead}) and adapt to it.

When the rewards or costs change, both models (ZeroR and *L*earnDynamic) need to be recomputed: ZeroR updates I_{best} and *L*earnDynamic re-trains its model. However, to compute I_{best} (for ZeroR) or re-train the model (for *Learn*Dynamic) there is no need to execute new TONIC processes, only to analyze the existing data. In details, to compute I_{best} for ZeroR according to the new reward and costs, one needs to go over the iterations of the TONIC processes in $\mathcal{T}$, recompute the net gain in each iteration according to the new reward and costs, and then compute I_{best} according to (8). To re-train the model for *Learn*Dynamic, one needs to go over the iterations of the TONIC processes in $\mathcal{T}$, re-label them as "continue" or "stop" according to the new net gain, and re-run the machine learning algorithm to train a new classifier.

7.2.3 Experimental results

We evaluated the proposed stopping conditions experimentally on the same dataset described earlier, using half of the dataset as the training set and the other half as the test set. We experimented with $R_{Acquire}$ values 3, 5, 10, 25, 50, and 100; and set $C_{Acquire} = 1$ and $C_{IsLead} = 0$ as in Section 7.1. After experimenting with several Machine Learning algorithms, we chose the XGBoost algorithm [15] as it produced the best overall results. We tuned its hyperparameters using grid search.

We compared ZeroR and *Learn*Dynamic to the following stopping conditions:

- **Fixed** X. A stopping condition according to which the TONIC process stops after exactly X iterations, where X is a parameter.

- **Oracle.** The offline-optimal stopping condition, which computes a-posteriori iteration in which the net gain is maximal and halts there. This stopping condition represents an upper bound of the net gain that can be achieved by any stopping condition.

We present the net gain obtained by each stopping condition with respect to the net gain of *Oracle* since no algorithm can obtain a higher net gain. Specifically, we used the *normalized loss* measure, defined as follows. Overloading the previous notation, let A be a stopping condition, T be a TONIC process, and $NG(A, T)$ be the net gain obtained when using A to stop T. The *normalized loss* of using A in T, denoted $NL(A, T)$, is the difference between the net gain obtained by A and the net gain obtained by *Oracle* divided by the net gain of *Oracle*. Formally, $NL(A, T) = (NG(Oracle, T) - NG(A, T))\,/NG(Oracle, T)$. Lower values of normalized loss are better, and a *normalized loss* of zero corresponds to obtaining optimal net gain. Table 3 shows the average *normalized loss* for *Learn*Dynamic, ZeroR, Fixed 20, Fixed 50, Fixed 100 and Fixed 250. Different columns correspond to different values of $R_{Acquire}$. For every value of $R_{Acquire}$, we highlighted in bold the best result over the evaluated stopping conditions. As can be seen, using *Learn*Dynamic yields the best results for most reward values, always providing a *normalized loss* smaller than 0.3.

The advantage of *L*earnDynamic over ZeroR diminishes for larger values of $R_{Acquire}$, and for $R_{Acquire} \geq 50$ the results of ZeroR are even better than *L*earnDynamic. To understand this trend, Table 4 shows the precision, recall, and F1 score of the classifier learned for the *L*earnDynamic stopping rule for different values of $R_{Acquire}$. Precision, recall, and F1 score are standard metrics for evaluating classifiers, and range from 0 to 1, where 1 is a perfect classification. We observe that the quality of our classifier degrades for higher values $R_{Acquire}$. This observation is consistent with the weaker performance of *Learn*Dynamic for high values of $R_{Acquire}$. This suggests that further improvements to the *Learn*Dynamic stopping condition may be obtained by improving the underlying learning algorithm, e.g., by obtaining more data, introducing new features, and using different learning schemes, and more advanced parameter tuning techniques. Nonetheless, the performance of *Learn*Dynamic is quite robust across all our experiments, often providing the best results or is at most withing 0.06 *normalized loss* of the best result.

8 TONIC as an information retrieval task

TONIC can be viewed as an Information Retrieval (IR) task, where the task is to retrieve as many leads as possible. This allows evaluating TONIC solvers with standard IR metrics. For example, after performing *x IsLead()* queries, the *Precision* is the fraction of leads found

Table 3 Normalized loss for the different stopping condition

$R_{Acquire}$	3	5	10	25	50	100
*L*earnDynamic	**0.30**	**0.14**	**0.17**	**0.14**	0.15	0.15
ZeroR	1.38	0.54	0.25	0.15	0.14	**0.09**
Fixed 25	0.83	0.49	0.43	0.43	0.44	0.46
Fixed 50	1.40	0.46	0.22	0.18	0.19	0.20
Fixed 100	2.86	0.97	0.32	**0.14**	**0.11**	0.11
Fixed 250	8.87	2.71	0.91	0.29	0.14	**0.09**

Table 4 Precision, recall, and F1 score for the classifier learned for the *L*earnDynamic stopping rule

$R_{Acquire}$	3	5	10	25	50	100
Precision	0.91	0.9	0.86	0.68	0.63	0.67
Recall	0.91	0.9	0.86	0.67	0.61	0.65
F1	0.91	0.9	0.86	0.66	0.59	0.64

so far out of the x queried profiles. *Recall* (also known as the true positive rate – TPR) is the fraction of leads found so far out of the total number of leads. Precision and Recall are often combined by the F1 measure, computed as $F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$.

Figure 10a plots average Precision, Recall, and the F1-measure as a function of the number of queries performed so far (x) for the Google+ dataset described above, using our best solver (EBysP). The Recall results corresponds to the results reported in the previous sections, showing a diminishing return effect: the recall increases quickly at the beginning of the search and then slows down until convergences. Precision is initially very high, but drops quickly as the search progresses and fewer leads are found. This is reasonable as there are significantly more non-leads than leads. Figure 10b zooms-in on the results of the first 100 iterations. Observe the precision results: until the 10th iteration the precision decreases, but then, it remains stable until the 30th iteration. In this period on average, two out of three queried profiles are leads, showing the strength of our search strategy (EBysP). Then, as the pool of available leads becomes sparser, fewer leads are being found, resulting afterwards in a quick drop in the precision and a more modest increase of the recall. Indeed, the 30th iteration is exactly the turning point where the F1 measure starts to decrease. In the limit (i.e,. after many iterations), most leads are found (so Recall is close to 1.0), the precision is almost zero since most profiles are not leads, and thus the F1 measure also drops down to zero.

The well-known *receiver operating characteristic* (ROC) curve [17], which measures the TPR as a function of the false positive rate (FPR), can also be created, by varying the values

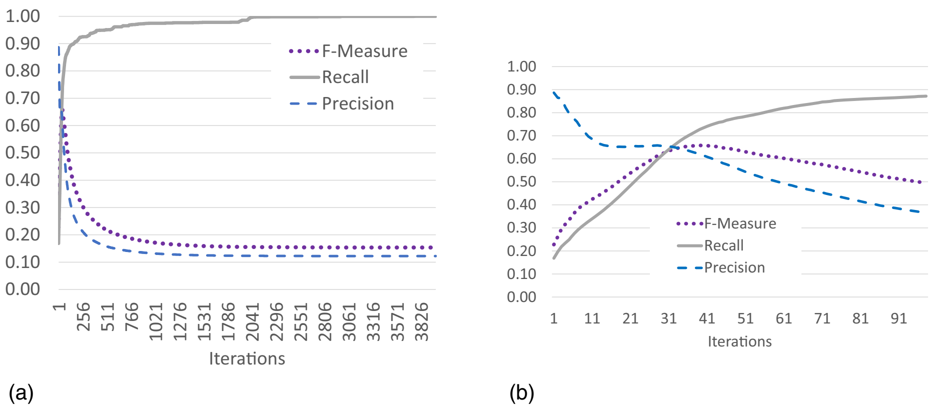


Figure 10 F1-measure, recall, and precision results for EBysP on the Google+ dataset. The right-hand side zooms in on the first 100 iterations

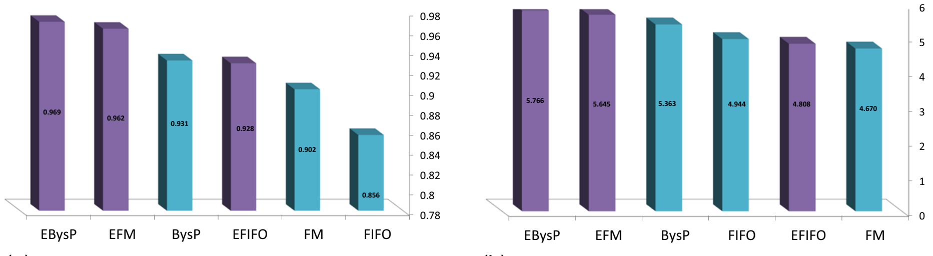
of x. The area under the ROC curve (AUC) is a popular IR measure, where a larger AUC indicates a better solver.

Figure 11a shows the average AUC of the algorithms: FIFO, EFIFO, FM , EFM, BysP and EBysP. The RTF heuristics (run under RTF) are colored in blue and the ETF heuristics (run under ETF(1)) are colored in purple. AUC results show trends similar to Figure 6 and Figure 4, 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.

Discounted cumulative gain (DCG) [30] is yet another metric for evaluating the quality of an ordered sequence of results. DCG accumulates the gain of finding the relevant results, i.e. the leads, such that the closer the lead is to the beginning of the sequence the higher its gain. Every *Islead()* query increases the network footprint of the intelligence collection process and with the footprint grows the chance the process will be discovered or the OSN service will block future acquisitions. Therefore, every *Islead()* query reduces the gain from further lead acquisitions and it is important to spot the leads as early in the search process as possible. Let $p_1, p_2, \ldots$ be the profiles sorted by the order in which *Islead()* query was applied to them. DCG is computed by $DCG = IsLead(p_1) + \sum_{i=2}^{n} \frac{IsLead(p_i)}{log_2 i}$, where $IsLead p_i = 1$ if the i^{th} profile is a lead and $IsLead p_i = 0$ otherwise.

Figure 11b shows the average DCG of the algorithms mentioned before. First, observe that in terms of DCG, using EFIFO yields worse results than using FIFO. This is in contrast to the AUC results (Figure 11a), where EFIFO yielded higher AUC compared to FIFO. This advantage of FIFO in terms of DCG is due to the fact that leads more densly populate the first tier. Therefore FIFO will find more leads in earlier iterations – yielding higher DCG – since it only searches the first tier and thus its search is less scattered. Thus, in general, for a given TONIC heuristic it may be better to use RTF than ETF if one wants to maximize DCG.

However, when the more intelligent TONIC heuristics are used, we observe again the benefit of using ETF. This is somewhat surprising, as one could expect that the first leads being found are from the first tier and thus RTF may perform better than ETF. These results suggest that the best performing TONIC heuristics – BysP and FM – are intelligent enough to consider expanding non-leads only when it is helpful, thus resulting in finding more leads than in RTF even in early iteration of the search. In summary, we observe that EBysP and EFM are robust across both AUC and DCG, showing the best performance in both measures.



(a) (b)

Figure 11 Evaluating TONIC heuristics using IR measures. The plots show the average AUC (**a**) and DCG (**b**) obtained on our benchmark

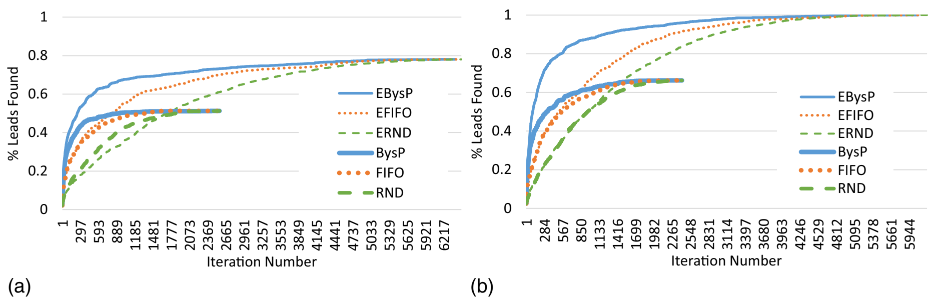


Figure 12 Evaluating TONIC heuristics on different OSNs. **a** shows results for the Live Journal OSN, and **b** shows results for the Pokec OSN

9 Experiments on different online social networks

To provide further support for our conclusions, we also performed a series of experiments on other OSNs. Namely, we experimented on LiveJournal [6, 35], which is an online community that allows member to maintain journals, blogs, and define friendship between profiles, and Pokec [49], which is the most popular OSN is Sloveniya. Both networks were made available as part of the Standform Network Analysis Project (SNAP) at snap.stanford.edu. Both OSNs are active and large (although not as large as Google+), where LiveJournal has more than 10 millions profiles and Pokec has more than 1.5 millions.

For each network we randomly chose 50 profiles as targets, considering only profiles with more than 30 friends and 4,000 edges in its 2-tier neighborhood. For each target we present results for both TONIC frameworks (RTF and ETF) and compare the baseline approaches random and FIFO with the best performing heuristic BysP.

Figure 12 shows the results for these two OSNs, in the same format as in Figure 7: the x-axis is the number of iterations and the y-axis is the percentage of leads found. The main trends observed for Google+ are seen here too:

- **RTF finds fewer profiles.** The RTF algorithms exhausts the set of leads they can find and converges to finding fewer leads than the ETF algorithm.
- **BysP and EBysP dominate the baseline methods.** For both OSNs, BysP is better than random and FIFO, and EBysP is better than ETF random and EFIFO.
- **EBysP dominates BysP.** Even before RTF exhausts all the leads it can reach, we observe that EBysP is able to find more leads faster than BysP. To demonstrate this, the average percentage of leads found after after 200 iterations was 45% and 64% for BysP and EBysP, respectively, in the Pokec OSN, and 39% and 47% for BysP and EBysP, respectively, in the Live Journal OSN.

It is also interesting to compare the behavior of the different frameworks – RTF and ETF – across the three OSNs we experimented with (Google+, Pokec, and Live Journal). Table 5

Table 5 % of leads reachable with RTF and ETF in different OSNs

Framework	Google+	Pokec	Live Journal
RTF	0.82	0.66	0.51
ETF	1.00	1.00	0.78

shows the average percentage of reachable leads found using RTF and ETF for our OSNs. We observe that Live Journal OSN is the most difficult in terms of TONIC, in the sense that RTF and even ETF ends up finding fewer percentage of leads compared Google+ and Pokec. Indeed, the observant reader will see that the ETF algorithms in Figure 12a do not converge to finding all leads. This suggests future research in which the limitation of ETF to the first tier will be reevaluated based on the OSN characteristics.

10 Conclusion and future work

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, RTF and ETF. RTF focuses the search by only acquiring leads. ETF generalizes RTF 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 RTF and ETF(1), we present several heuristics for guiding the search. These heuristics are evaluated experimentally on a real OSN. Evaluation results show that:

1) The Bayesian Promising (BysP and EBysP) heuristics and the heuristics based on the Friends Measure, significantly outperform other heuristics.
2) Using ETF results in substantially more leads than RTF when using the anytime objective.
3) Depending on the costs of the queries and on the reward of finding leads, either BysP (RTF) or EBysP (ETF(1)) is the preferable when using the MaxGain objective.
4) The number of reachable leads increases with both the number of initial leads and tier.

There are many exciting directions for future work. An obvious future work is to evaluate the proposed algorithms on a larger dataset with more OSNs, and studying how properties of a given OSN affect their performance. Another future research direction 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.

For example, one can learn a more sophisticated hybrid approach that combines information from several TONIC heuristics.

Another direction is to consider leads with different rewards, where finding a lead that provides more information about *target* is preferred.

In addition, one may consider higher cost for querying some profiles, e.g., to avoid querying profiles that are more sensitive and thus querying them is more risky. This raises a complex task of how to quantify information and how to quantify risk. 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.

11 Ethical aspects in TONIC

As mentioned earlier in this paper, a particular application of our investigation of effective TONIC solvers it to uncover the LOF of a target profile by exploring the LOFs of other profiles. Consequently, an effective TONIC solver may uncover a target's LOF even if the target has blocked direct access to its LOF through its profile. Although this information (other profiles LOF) is publicly available, the ability to automatically and effectively access this information raises some ethical concerns. First, one may argue that if the target has blocked access to its LOF, then by searching for its LOF we are accessing to the target's private information, even if the search itself is not done on the target but on other profiles. Second, one can imagine settings in which criminals use the proposed algorithm to collect information about citizens for various criminal activities.

Indeed both concerns are valid and need to be discussed. 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). This is often done manually, for example, before hiring a person, or when police forces monitoring affiliates of known criminals in social networks. In fact, there are existing tools such as Palantir (www.palantir.com) that automatically collects publicly available information on a given profile. In that sense, our work is a more effective version of an existing approach (which usually applies a FIFO strategy, which we show to be inferior).

Moreover, we emphasize that an effective TONIC solver can be used for many good purposes. It can serve as an helpful tool for law enforcement agencies. For example, imagine a search for information about a pedophile in an OSN. In fact, the social network paradigm has been successfully used to investigate organized crime in the Netherlands [33]. Alternatively, an effective 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. Indeed, like many other advances in science the algorithms we propose can be used for good and bad purposes, depending on its user.

References

1. Adamic, L.A., Lukose, R.M., Puniyani, A.R., Huberman, B.A.: Search in power-law networks. Phys. Rev. E **64**, 046135 (2001)
2. Aggarwal, C.C., Al-Garawi, F., Yu, P.S.: Intelligent crawling on the world wide web with arbitrary predicates. In: Proceedings of the 10th international conference on World Wide Web. ACM, pp. 96–105 (2001)
3. Almpanidis, G., Kotropoulos, C., Pitas, I.: Combining text and link analysis for focused crawling—an application for vertical search engines. Inf. Syst. **32**(6), 886–908 (2007)
4. Altshuler, Y., Aharony, N., Fire, M., Elovici, Y., Pentland, A.: Incremental learning with accuracy prediction of social and individual properties from mobile-phone data, CoRR, vol. arXiv:1111.4645. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1111.html#abs-1111-4645 (2011)
5. Altshuler, Y., Elovici, Y., Cremers, A.B., Aharony, N., Pentland, A.: Security and Privacy in Social Networks. Springer, Berlin (2012)
6. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group formation in large social networks: Membership, growth, and evolution. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 44–54 (2006)
7. Barabási, A.-L., Réka, A.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)

8. Bidoki, A.M.Z., Yazdani, N., Ghodsnia, P.: FICA: A fast intelligent crawling algorithm. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence. IEEE Computer Society, pp. 635–641 (2007)
9. Bnaya, Z., Puzis, R., Stern, R., Felner, A.: Social network search as a volatile multi-armed bandit problem. ASE Human **2**(2), pp–84 (2013)
10. Bujlow, T., Carela-Español, V., Sole-Pareta, J., Barlet-Ros, P.: A survey on web tracking: mechanisms, implications, and defenses. Proc. IEEE **105**(8), 1476–1510 (2017)
11. Cai, R., Yang, J.-M., Lai, W., Wang, Y., Zhang, L.: irobot: An intelligent crawler for web forums. In: Proceedings of the 17th international conference on World Wide Web. ACM, pp. 447–456 (2008)
12. Chakrabarti, S., Van den Berg, M., Dom, B.: Focused crawling: a new approach to topic-specific web resource discovery. Comput. Netw. **31**(11), 1623–1640 (1999)
13. Chang, C., Kayed, M., Girgis, M., Shaalan, K., et al.: A survey of web information extraction systems. IEEE Trans. Knowl. Data Eng. **18**(10), 1411 (2006)
14. Chen, Z., Ma, J., Lei, J., Yuan, B., Lian, L.: An improved shark-search algorithm based on multi-information. In: 2007. FSKD 2007. Fourth International Conference on Fuzzy Systems and Knowledge Discovery. IEEE, vol. 4, pp. 659–658 (2007)
15. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), pp. 785–794 (2016)
16. Cho, J., Garcia-Molina, H., Page, L.: Efficient crawling through url ordering. Comput. Netw. ISDN Syst. **30**, 161–172 (1998)
17. Croft, W., Metzler, D., Strohman, T.: Search engines: Information retrieval in practice. Addison-Wesley, Reading (2010)
18. Davis, D., Lichtenwalter, R., Chawla, N.V.: Multi-relational link prediction in heterogeneous information networks. In: 2011 International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE, pp. 281–288 (2011)
19. De Bra, P., Post, R.: Searching for Arbitrary Information in the Www: the Fish-Search for Mosaic. In: WWW (1994)
20. Diligenti, M., Coetzee, F., Lawrence, S., Giles, C.L., Gori, M., et al.: Focused crawling using context graphs. In: VLDB, pp. 527–534 (2000)
21. Dong, Y., Tang, J., Wu, S., Tian, J., Chawla, N.V., Rao, J., Cao, H.: Link prediction and recommendation across heterogeneous social networks. In: 2012 IEEE 12th International Conference on Data Mining. IEEE, pp. 181–190 (2012)
22. Ermakova, T., Fabian, B., Bender, B., Klimek, K.: Web Tracking – a Literature Review on the State of Research. In: HICSS 51 (2018)
23. Felner, A., Stern, R., Ben-Yair, A., Kraus, S., Netanyahu, N.: PhA*: Finding the shortest path with A* in unknown physical environments. J. Artif. Intell. Res. **21**, 631–679 (2004)
24. Fire, M., Tenenboim, L., Lesser, O., Puzis, R., Rokach, L., Elovici, Y.: Link prediction in social networks using computationally efficient topological features. In: IEEE international conference on social computing (SocialCom), pp. 73–80 (2011)
25. Fire, M., Katz, G., Elovici, Y., Shapira, B., Rokach, L.: Predicting student exam's scores by analyzing social network data. In: AMT, pp. 584–595 (2012)
26. Fire, M., Tenenboim-Chekina, L., Puzis, R., Lesser, O., Rokach, L., Elovici, Y.: Computationally efficient link prediction in a variety of social networks. ACM Trans Intell Syst Technol (TIST) **5**(1), 10 (2013)
27. Fire, M., Tenenboim-Chekina, L., Puzis, R., Lesser, O., Rokach, L., Elovici, Y.: Computationally efficient link prediction in a variety of social networks, ACM Trans. Intell. Syst. Technol. **5**(1), 1–25 (2014)
28. Gjoka, M., Kurant, M., Butts, C.T., Markopoulou, A.: Walking in facebook: A case study of unbiased sampling of osns. In: INFOCOM, pp. 1–9 (2010)
29. Hersovici, M., Jacovi, M., Maarek, Y.S., Pelleg, D., Shtalhaim, M., Ur, S.: The shark-search algorithm. an application: tailored web site mapping. Comput. Netw. ISDN Syst. **30**(1), 317–326 (1998)
30. Jarvelin, K., Kekalainen, J.: Cumulated gain-based evaluation of ir techniques. ACM Trans. Inf Syst **20**(4), 422–446 (2002)
31. Katz, L.: A new status index derived from sociometric analysis. Psychometrika **18**(1), 39–43 (1953)
32. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM **46**(5), 604–632 (1999)
33. Klerks, P.: 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 (2001)
34. Kurant, M., Gjoka, M., Butts, C.T., Markopoulou, A.: Walking on a graph with a magnifying glass: Stratified sampling via weighted random walks. In: ACM Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), pp. 281–292 (2011)

35. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Math. **6**(1), 29–123 (2009)
36. Li, X., Smith, J.D., Dinh, T.N., Thai, M.T.: Privacy issues in light of reconnaissance attacks with incomplete information. In: IEEE/WIC/ACM International Conference on Web Intelligence (WI), pp. 311–318 (2016)
37. Li, X., Smith, J.D., Thai, M.T.: Adaptive reconnaissance attacks with near-optimal parallel batching. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). IEEE. pp. 699–709 (2017)
38. Liben-Nowell, D., Kleinberg, J.: The link-prediction problem for social networks. J Amer Soc Inf Sci Technol **58**(7), 1019–1031 (2007)
39. McPherson, M., Smith-Lovin, L., Cook, J.M.: Birds of a feather: Homophily in social networks. Annu. Rev. Sociol. **27**(1), 415–444 (2001)
40. Menczer, F., Pant, G., Srinivasan, P., Ruiz, M.E.: Evaluating topic-driven web crawlers. In: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, pp. 241–249 (2001)
41. Mislove, A., Viswanath, B., Gummadi, K.P., Druschel, P.: You are who you know: inferring user profiles in online social networks. In: Proceedings of the third ACM international conference on Web search and data mining. ACM, pp. 251–260 (2010)
42. Mitchell, T.M.: Machine learning. McGraw-Hill, McGraw-Hill (1997)
43. Pawlas, P., Domański, A., Domańska, J.: Universal web pages content parser. In: Computer Networks. Springer, pp. 130–138 (2012)
44. Russell, S.J., Norvig, P.: Artificial intelligence - A modern approach pearson education (2010)
45. Samama-Kachko, L., Puzis, R., Stern, R., Felner, A.: Extended Framework for Target Oriented Network Intelligence Collection. In: Symposium on Combinatorial Search (SoCS) (2014)
46. Stern, R., Kalech, M., Felner, A.: Searching for a K-Clique in Unknown Graphs. In: SOCS (2010)
47. Stern, R.: Finding patterns in an unknown graph. AI Commun. **25**(3), 229–256 (2012)
48. Stern, R.T., Samama, L., Puzis, R., Beja, T., Bnaya, Z., Felner, A.: TONIC Target Oriented Network Intelligence Collection for the Social Web. In: AAAI (2013)
49. Takac, L., Zabovsky, M.: Data analysis in public social networks. In: International Scientific Conference and International Workshop Present Day Trends of Innovations, pp. 1–6 (2012)
50. Tang, J., Lou, T., Kleinberg, J.: Inferring social ties across heterogenous networks. In: Proceedings of the fifth ACM international conference on Web search and data mining. ACM, pp. 743–752 (2012)
51. Tang, J., Yao, L., Zhang, D., Zhang, J.: A combination approach to web user profiling. ACM Trans. Knowl. Discov. Data **5**(1), 2:1–2:44 (2010)
52. Vempaty, N.R., Kumar, V., Korf, R.E.: Depth-first vs best-first search. In: National Conference on Artificial Intelligence (AAAI), pp. 434–440 (1991)
53. Wang, W., Chen, X., Zou, Y., Wang, H., Dai, Z.: A focused crawler based on naive bayes classifier. In: 2010 Third International Symposium on Intelligent Information Technology and Security Informatics (IITSI). IEEE, pp. 517–521 (2010)
54. Watts, D.J., Strogatz, S.: Collective dynamics of 'small-world' networks. Nature **393**, 6684 (1998)
55. Zilberstein, S.: Using anytime algorithms in intelligent systems. AI Mag. **17**(3), 73–83 (1996)