A Content Recommendation Policy for Gaining Subscribers

Konstantinos Theocharidis
University of the Peloponnese & Information Management Systems Institute, Athena Research Center, Greece

Manolis Terrovitis
Information Management Systems Institute
Athena Research Center, Greece

Spiros Skiadopoulos
University of the Peloponnese, Greece

Panagiotis Karras
Aarhus University, Denmark

ABSTRACT

How can we recommend content for a *brand* agent to use over a series of rounds so as to gain new subscribers to its social network page? The *Influence Maximization* (IM) problem seeks a set of k users, and its content-aware variants seek a set of k post features, that achieve, in both cases, an objective of expected influence in a social network. However, apart from raw influence, it is also relevant to study gain in subscribers, as long-term success rests on the subscribers of a brand page; classic IM may select k users from the subscriber set, and content-aware IM starts the post's propagation from that subscriber set. In this paper, we propose a novel *content recommendation policy* to a brand agent for *Gaining Subscribers by Messaging* (GSM) over many rounds. In each round, the brand agent messages a fixed number of social network users and invites them to visit the brand page aiming to gain their subscription, while its most recently published content consists of features that intensely attract the preferences of the invited users. To solve GSM, we find, in each round, which content features to publish and which users to notify aiming to maximize the cumulative subscription gain over all rounds. We deploy three GSM solvers, named RANDOM, SCAN, and SUBSTITUTE, and we experimentally evaluate their performance based on VKontakte (VK) posts by considering different user sets and feature sets. Our experimental results show that SUBSTITUTE provides the best solution, as it is significantly more efficient than SCAN with a minor loss of efficacy and clearly more efficacious than RANDOM with competitive efficiency.

CCS CONCEPTS

• **Information systems** → **Social networks**; **Recommender systems**; **Content match advertising**; **Content ranking**; **Texting**.

KEYWORDS

content recommendation, subscription gain, messaging, ranking

ACM Reference Format:
Konstantinos Theocharidis, Manolis Terrovitis, Spiros Skiadopoulos, and Panagiotis Karras. 2022. A Content Recommendation Policy for Gaining Subscribers. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3477495.3531885

SIGIR '22, July 11–15, 2022, Madrid, Spain

ACM ISBN 978-1-4503-8732-3/22/07…$15.00
https://doi.org/10.1145/3477495.3531885

1 INTRODUCTION

The problem of *Influence Maximization* (IM) [7] is relevant and useful to stakeholders (henceforward, *brands*) that pursue viral marketing campaigns in social networks. The classic IM [4] seeks k users that maximize the influence of a *fixed* post in a network; the inverse variant of IM [3] seeks k content *features*[1] to form a viral post that starts its diffusion from a *fixed* set of initial adopters.

Nowadays, most brands maintain social network pages for advertising purposes, since social network users choose to *follow* pages they are interested in; these followers are the *subscribers* of a brand. Yet, in the IM literature, subscribers are usually *taken for granted* [3, 5] or *ignored* [4, 7]. In the former case, it is lucrative for the content-aware IM techniques to apply in practice only when several subscribers exist; yet, *new* brands having zero or limited subscribers cannot benefit from such techniques. So, we provide a concrete way for such brands to gain subscribers and take advantage of works in [3, 5]. In the latter case, the classic IM problem applies independently of subscribers but their loyalty capabilities are not explored. For instance, it is more feasible and economic for a brand to motivate k influential loyal subscribers for promoting its posts than searching for k agnostic adopters that may not be supporters of the brand and contribute loosely to its promotion. So, even *established* brands with several subscribers, can benefit from our gaining subscribers method so as to find even more influential loyal users for classic IM purposes [4, 7]. Further, in the real world, the network topology is usually not known on its whole [2, 6, 8], whereas subscribers are always known, even if that knowledge requires explicit on-demand retrieval. Therefore, the need arises to focus on subscribers and study how brands can gain subscribers.

In this paper, we propose a novel multi-round *content recommendation policy* that a brand agent/advertiser can use to *Gain Subscribers by Messaging* (GSM). As the GSM problem takes place over many rounds, we deploy three algorithms that solve GSM in a non-adaptive way (beforehand) for all rounds. Our solutions recommend to the advertiser, in each round, which k content features to publish and which m non-subscriber users to notify of those k features so as to maximize chances to gain the subscription of those m users. The notification is done by messaging (e.g., a short message acting like an invitation to visit the brand page), and each user is notified once; that user is never notified again for any reason. The best GSM solver is the one that achieves the maximum *subscription gain* over all rounds. We define subscription gain (henceforward, *SG*) as a weighted sum depicting the aggregate preference of m

[1] We consider that each *feature* corresponds to a specific social network page.

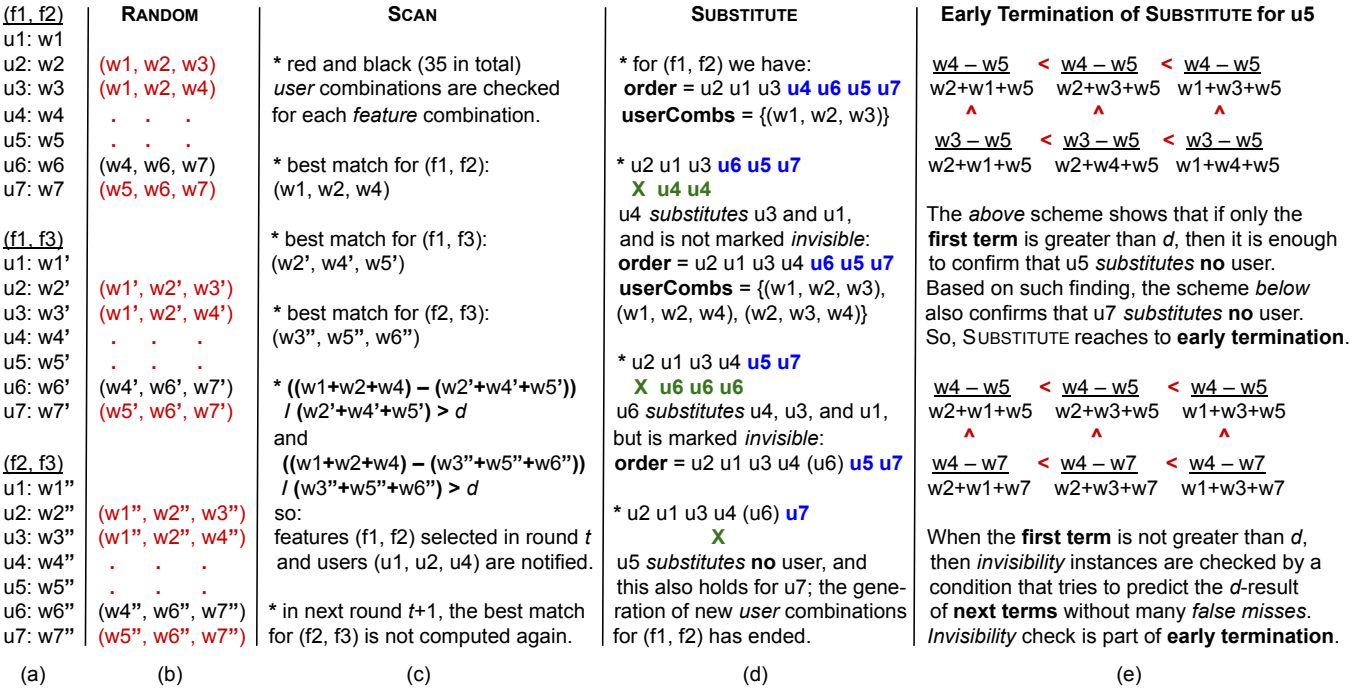


Figure 1: An example that shows the basic execution components of GSM solvers for $k = 2$ features and $m = 3$ users.

users for k features. For any two (k, m) solutions that achieve similar SG, we consider the cumulative (no duplicates allowed) *reach* of their respective m users to select the (k, m) solution with the maximum reach; the reach (henceforward, R) of a user is equal to her out-degree. R acts as a second filter (when needed) that helps to the selection of most influential new subscribers.

GSM naturally applies to social networks, such as VK[2], which constitutes the Russian version of Facebook in terms of usability and scale. In social networks, the pages to which a user subscribes, form the features (preferences) of user and in this work we consider real VK posts to fine-tune with different weights such features for our experiments. Moreover, VK strictly allows 20 messages per 12 hours to any user who has a VK account and wishes to send a message to any other non-friend VK user. Thus, each message is valuable for the advertiser and constitutes a single chance to attract the attention of notified user. By solving GSM, the advertiser prioritizes the publishing (as each round has priority over the next round) of the right k-size content for the right m users to maximize the gain of subscribers. If, alternatively, the advertiser were to apply a random messaging policy, then she would gain subscribers at a lower pace, and also face the danger of losing access to her page for some period due to spam reports sent by notified users to the VK company; a user invited to visit a page of no interest may report spamming. To make motivation clear, we present the next example:

Example: Consider Figure 1. If advertiser has no algorithm to solve GSM, then she randomly selects m users (e.g., v_4, v_6, v_7) and k features (e.g., f_1, f_3); this is a trivial approach and we do not use it in this work. Yet, if she uses RANDOM algorithm, then for the selected m users (e.g., same as previous) she finds the k features that give the maximum SG (e.g., f_1, f_2 with $SG = w_4 + w_6 + w_7$), where w_4 equals to the weighed sum of v_4 in regards to f_1 and f_2. Lastly, if she uses SCAN or SUBSTITUTE algorithm, then she finds a (k, m) solution with much higher SG due to considering all possible (k, m) combinations to optimally solve GSM instead of depending on random selections. A higher SG expresses a higher probability[3] that some of m invited users will become subscribers to brand's page. Algorithms in Figure 1 will be gradually discussed in paper.

We summarize our contributions as follows: (1) we propose the GSM problem that applies to any social network; (2) deploy three GSM solvers, named RANDOM, SCAN, and SUBSTITUTE; and (3) provide a rich experimental evaluation that verifies the superiority of SUBSTITUTE over others; to the best of our knowledge, the problem of gaining subscribers using content has not been studied previously.

2 GSM SOLVERS

We define the GSM problem as follows: Given a social network $G = (V, E)$ with $|V|$ users and $|E|$ edges, a feature universe L, a weighted feature set F_v of size $|L|$ capturing the preferences of each user v, a budget k, a limited number of m notification messages, a similarity threshold d, and a number of rounds n, find in each round t what k content features to publish and which m users to notify so as to maximize the cumulative subscription gain SG over n rounds:

$$SG = \max \sum_{t=1}^{n} SG_t(k, m)$$

2.1 The Solver RANDOM

RANDOM constitutes a baseline that solves GSM. In each round t, it uniformly at random selects m users from V not chosen in previous rounds and focuses on selecting features. In more detail, RANDOM

[2]See https://vk.com/

[3]The sum of weights over features for each user equals to 1. So, the maximum SG value equals to m and each SG value divided by m belongs to range $[0, 1]$.

searches for the k-size feature set that yields the maximum gain in round t with regard to the selected m users, and increases the cumulative SG and reach R. As a last step, it keeps track of all notified users to avoid messaging them again. In Figure 1b, RANDOM selects the users $v4, v6, v7$ and so it compares only the 3 black user combinations to find which k features are the best match (yield the highest SG) for selected users. Note that RANDOM overlooks all the red user combinations and that depicts its crucial deficiency.

2.2 The Solver SCAN

Algorithm SCAN presents the solver SCAN. In a nutshell, SCAN computes and stores in each round t the best match (the m-size user combination that yields the maximum SG) for each k-size feature combination by examining all possible m-size user combinations, so as to find the *roundBest* for t, and skips in next rounds any feature combinations that do not need to be processed again.

Algorithm SCAN

```
   Input    : G, L, F_v
   Output   : SG, R // cumulative subscription gain and reach over n rounds in GSM
   Param.   : k, m, d, n
   // Each entry of fC maps k features to the m users who yield the maximum SG
1  for each feature combination c_f from k feature combinations of L do fC[c_f] = ∅;
   // Each entry of uC maps m users to their reach R as computed in G
2  for each user combination c_u from m user combinations of V do uC[c_u] = −1;
3  SG = 0; R = 0; deletedV = ∅;
4  for t = 1, ..., n do
5      roundBest.users = ∅; roundBest.features = ∅; roundBest.SG = 0;
       roundBest.R = 0; // initialize the best solution for round t
6      for each feature combination c_f ∈ fC do
           // 1. Check to may skip computation of fC[c_f] and Update roundBest
7          if all users in deletedV are not contained in fC[c_f] then
8              c_f.SG = computeSG(fC[c_f], c_f, F_v);
9              if roundBest.SG ≥ c_f.SG then
10                 diff = (roundBest.SG − c_f.SG) ÷ c_f.SG;
11                 if diff > d then continue;
12             else
13                 diff = (c_f.SG − roundBest.SG) ÷ roundBest.SG;
14                 if diff > d then
15                     if uC[fC[c_f]] = −1 then
16                         uC[fC[c_f]] = computeReach(fC[c_f], G);
17                     roundBest.users = fC[c_f];
                       roundBest.features = c_f; roundBest.SG = c_f.SG;
                       roundBest.R = uC[fC[c_f]]; continue;
18             Repeat lines 15-16 to compute uC[fC[c_f]];
19             if uC[fC[c_f]] > roundBest.R or (uC[fC[c_f]] = roundBest.R
               and c_f.SG > roundBest.SG) then Update roundBest as in line 17;
               // continue is executed either the condition in if is true or false
           // 2. Compute fC[c_f] with the examination of each c_u ∈ uC (loop here!)
20         Let crnt_c_u be the current best user combination stored in fC[c_f];
21         Let next_c_u be the next c_u ∈ uC for examination to may update fC[c_f];
22         Compare crnt_c_u with next_c_u by following a similar process to lines 8-19
           so as to may update fC[c_f] and crnt_c_u with next_c_u;
           // 3. Update roundBest
23         Repeat lines 8-19 to update roundBest;
24     SG = SG + roundBest.SG; R = R + roundBest.R;
25     deletedV = ∅; for each user v ∈ roundBest.users do deletedV.insert(v);
26     Delete each c_u ∈ uC that contains at least one user v ∈ deletedV;
27 return SG, R;
```

We indicate the algorithm's workflow by marking three execution steps with bold numbers above Line 7, Line 20, and Line 23. In the first step (Lines 7–19), if all users notified in the previous round are *not* contained in the best match for current feature set c_f, then the best match for c_f does not change and it is used again to possibly update *roundBest* based on similarity threshold d. If the first step is not executed (condition in Line 7 is false), then we move to the second step (Lines 20–22). This step is the most costly part of SCAN as it examines all sets of m users sequentially to find the best match for c_f. After this processing, the third step (Line 23) possibly updates *roundBest* by using the found best match for c_f.

In Figure 1c, SCAN compares 35 m-size user combinations for each one of (f_1, f_2), (f_1, f_3), and (f_2, f_3) so as to find their best matches. Then, SCAN compares those three best matches and finds that the best match of (f_1, f_2) yields the higher SG over others, and so the features f_1, f_2 and users v_1, v_2, v_4 are selected in current round. Note that in next round, the best match of (f_2, f_3) remains unchanged since it does not overlap with notified users of previous round.

2.3 The Solver SUBSTITUTE

Algorithm SUBSTITUTE presents the solver SUBSTITUTE.

Algorithm SUBSTITUTE

```
   Input    : G, L, F_v
   Output   : SG, R // cumulative subscription gain and reach over n rounds in GSM
   Param.   : k, m, d, n
   // Each entry of fC maps k features to the m users who yield the maximum SG
1  for each feature combination c_f from k feature combinations of L do fC[c_f] = ∅;
2  for each feature combination c_f ∈ fC do
3      v.SG = computeSG(v, c_f, F_v); allV[c_f].insert((v, v.SG)); // v.SG desc sort
4      Form selV[c_f] by taking the first m * n pairs of allV[c_f]; // plus d-extensions
5  SG = 0; R = 0; deletedV = ∅;
6  for t = 1, ..., n do
7      roundBest.users = ∅; roundBest.features = ∅; roundBest.SG = 0;
       roundBest.R = 0; // initialize the best solution for round t
8      for each feature combination c_f ∈ fC do
           // 1. Check to may skip computation of fC[c_f] and Update roundBest
9          Repeat lines 7-19 of SCAN with only difference that now c_f.R =
           computeReach(fC[c_f], G) is used and so c_f.R replaces uC[fC[c_f]];
           // 2. Create only the necessary m-size user combinations (uC) for c_f
10         Form the first c_u by taking the first m pairs of selV[c_f] and set
           uC[c_u] = −1; // each c_u is a set of pairs (v, v.SG) here
11         offset = m; c_f.invisible = ∅; // it contains each invisible v ∈ selV[c_f]
12         for e2 = selV[c_f].get(m + 1), ..., selV[c_f].last() do
13             subs = ∅; invis = 0; travs = 0;
14             for e1 = selV[c_f].get(offset), ..., selV[c_f].first() do
15                 if e1.v ∈ c_f.invisible then { invis + +; travs + +; continue; }
16                 e1_comb = ∅; e2_comb = ∅; denom.SG = e2.v.SG;
17                 for ei = selV[c_f].first(), ..., selV[c_f].get(m) do
18                     if ei = e1 then continue;
19                     e1_comb.insert(ei.v); e2_comb.insert(ei.v);
20                     denom.SG = denom.SG + ei.v.SG;
21                     After m − 1 loop executions break;
22                 diff = (e1.v.SG − e2.v.SG) ÷ denom.SG;
23                 if diff > d then break; else subs.insert(e1); // e2 can sbst e1
24                 e1_comb.insert(e1.v); e1.R = computeReach(e1_comb, G);
25                 e2_comb.insert(e2.v); e2.R = computeReach(e2_comb, G);
26                 e1_out = G.outEdges(e1.v); e2_out = G.outEdges(e2.v);
27                 if (e1.R ≥ e2.R and |e1_out| ≥ |e2_out|) then invis + +;
28                 travs + +; // traversal of selV[c_f] proceeds with its previous e1
29             if |subs| = 0 then break; // termination; no more uC are created
30             offset + +; if travs = invis then c_f.invisible.insert(e2.v) and
               continue; // e2 marked as invisible for all traversed e1; no uC for e2
31             For each e1 ∈ subs and for each c_u ∈ uC where e1 ∈ c_u, replace e1
               with e2 to form c_u' and set uC[c_u'] = −1; // each c_u' is a new uC
           // 3. Compute fC[c_f] with the examination of each c_u ∈ uC (loop here!)
32         Repeat lines 20-22 of SCAN; // c_f.R in place of uC[fC[c_f]] as in line 9
           // 4. Update roundBest
33         Repeat line 23 of SCAN; // c_f.R in place of uC[fC[c_f]] as in line 9
34     SG = SG + roundBest.SG; R = R + roundBest.R;
35     deletedV = ∅; for each user v ∈ roundBest.users do deletedV.insert(v);
36     Delete from each selV[c_f] all pairs that contain a user v ∈ deletedV;
37 return SG, R;
```

This solver addresses the main bottleneck of SCAN, which is the examination of all user combinations in its second step. In

particular, SUBSTITUTE adds an intermediate step (Lines 10–31) that creates only the necessary m-size user combinations to find the best match for current feature set c_f. To do that, it utilizes the structure $selV$ (initialized in Lines 2–4) that stores for each c_f a set of pairs corresponding to users associated with their achieved SG in c_f, organized in descending order by their SG values.

Specifically, the first m entries of $selV[c_f]$ form the first user combination (Line 10). Then, we compare each entry $e2$ (Line 12) having a position greater than m in $selV[c_f]$ with each previous entry $e1$ of $selV[c_f]$ (Line 14) to check whether $e2$ can *substitute* $e1$. If a substitution happens (Line 23), there may exist a better solution for c_f having $e2$ in place of $e1$. Otherwise, if $e2$ cannot substitute the first compared $e1$, then $e2$ cannot substitute any $e1$, and also this holds for any next $e2$, resulting in *early termination* (Line 29). This termination is beneficial for two reasons: first, there is no need to compare more $(e2, e1)$ pairs to find a better solution for c_f, and second, it ends the generation of new user combinations for c_f thereby incurring less overhead in third step (Line 32). To further enhance that termination, we add a condition in Line 27 that counts instances of *invisibility*. A case of invisibility occurs when the current combination that includes $e1$ has no less reach than the same combination with $e2$ in place of $e1$, and also the atomic reach of $e1$ is no less than the atomic reach of $e2$; then, in most cases $e1$ is a better option than $e2$. If such evidence is observed for all the $e1$ to which we compare $e2$, then we mark $e2$ as invisible henceforward (Line 30). Intuitively, the invisibility of $e2$ means that it is very likely that no user combination derived by replacing any $e1$ with $e2$, would constitute a better solution for c_f, so we opt to ignore them.

Figure 1d presents the execution logic of SUBSTITUTE for $c_f = (f_1, f_2)$. The *order* represents $selV$ and *userCombs* includes the m-size user combinations for c_f that can be seen as a gradually constructed *knowledge base* to find the best match for c_f. With blue color we depict the not yet examined entries of $selV$, while with green color we capture its currently examined entries for substitution; the symbol X denotes no substitution after checking, and when no green mark exists, it means no checking at all of respective $(e2, e1)$ pair. In more detail, the first entry of *userCombs* comprises users v_1, v_2, v_3. Then, $e2 = v_4$ is compared with all black $e1$ that also they totally did not mark it as invisible, but v_4 does not substitute $e1 = v_2$, so *userCombs* extends with combinations derived after the substitution of v_4 with v_3 and v_1 to its current entries. After that, $e2 = v_6$ can achieve three substitutions, but throughout checking, it marked as invisible and so it does extend at all *userCombs*. Finally, $e2 = v_5$ compares only with $e1 = v_4$ and since it cannot substitute it, it leads to early termination as it is sure that v_5 cannot substitute any $e1$ and this also holds for any entry after v_5 (here, v_7). So, although SCAN examines 35 user combinations to find the best match for (f_1, f_2), SUBSTITUTE finds that best match by only examining 3 user combinations; the same logic applies for (f_1, f_3) and (f_2, f_3).

Figure 1e justifies why the aforementioned early termination is possible when v_5 cannot substitute v_4. Since the *first term* is greater than d, namely $\frac{w_4 - w_5}{w_2 + w_1 + w_5} > d$, depicting the comparison result of user combinations (v_2, v_1, v_4) and (v_2, v_1, v_5), and showing that v_5 cannot substitute v_4, we prove early termination based on red *less* symbols. In the first (top) scheme, we show that each *next term*, capturing a different d-comparison among v_5 and v_4 (first row) and among v_5 and v_3 (second row), is even greater than d if *first term* is greater than d; this also holds inductively for respective terms relative to v_1 and v_2. In the second (bottom) scheme, we apply a similar logic and show that each *next term* of second row that captures a different d-comparison among v_7 and v_4 is also greater than d if *first term* is greater than d, and so v_7 cannot substitute v_4 as also no other user based on the inductive logic of first scheme.

3 EXPERIMENTAL EVALUATION

We wrote code in C++ and ran experiments on an AMD Ryzen 5 4600U CPU @2.1 GHz machine with 16GB RAM running Linux Ubuntu 20.04.3 LTS 64-bit. For graph operations we used Lemon [1].

3.1 Setup

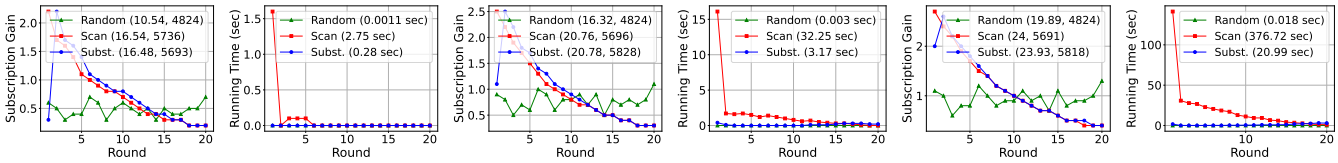
Datasets. As VK comprises 27 categories, we select the 1 and 2 most popular pages from each category, to form sets of 27 and 54 features. We uniformly at random select 10 different groups of 80 users for $|L| = 27$ and $|L| = 54$ (same groups for both L), and 10 different groups of 100 users for $|L| = 27$ and $|L| = 54$ (same groups for both L); there is no relation among groups of 80 and 100 users. So, all the experimental results in our figures are averaged by 10.
Tuning. For each selected user, we realistically tune her $F_v(f)$ value for each feature $f \in L$ by taking the average of her *like* responses in all posts of the years 2010-2017 of VK; as f we considered only the *brand* that published the relative post where user liked it. The feature weight sum of each $F_v(f)$ equals to 1, and for any f where no *like* found, we initialized $F_v(f)$ with a dummy value.
Parameters. The number of content features (k) is examined until $k = 3$, as higher k values were time-consuming for SCAN. The other parameters are fixed; the number of m invited users is 3, the similarity threshold d is 0.001, and the number of n rounds is 20.

3.2 Results

Figure 2 presents the subscription gain (SG) and running time (measured in seconds) results per round of RANDOM, SCAN, and SUBSTITUTE for $k = 1, 2$, and 3, and for $|L| = 27, |V| = 80$; Figure 3 does the same for $|L| = 27, |V| = 100$, Figure 4 for $|L| = 54, |V| = 80$, and Figure 5 for $|L| = 54, |V| = 100$. Besides the per round results, in each SG figure there is in legend for each solver a pair (X, Y), where X is the cumulative SG and Y is the cumulative reach R[4] over all rounds. Similarly, in each time figure, the legend mentions for each solver the cumulative running time over all rounds.

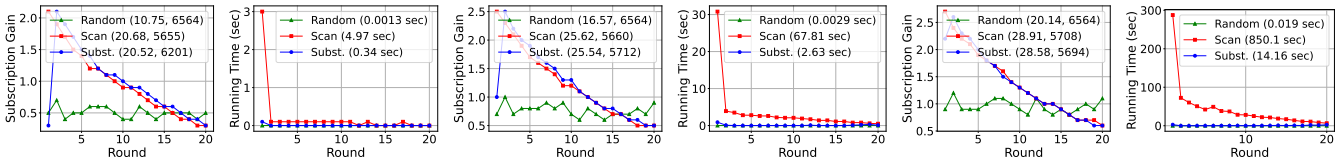
We observe that the general trend is the same in all figures; SUBSTITUTE is almost equally effective to SCAN and competitively efficient to RANDOM. It is evident that SUBSTITUTE achieves a slightly less cumulative SG in regards to SCAN due to the poor SG result it achieves in first round; that behavior depends on aggressive false misses derived from true evaluation of condition in Line 27, but as rounds evolve such false misses reduce due to skipping of each c_f having an unaffected best solution (Line 9). Except for first round, the SG of both solvers is very close and diminishes as rounds grow because promising subscription candidates are messaged from the beginning. Yet, the SG of RANDOM is the lowest and can be seen as fixed across rounds, as it does not apply messaging with priority as previous solvers. Regarding running time, SUBSTITUTE is slower than

[4]We present R results for completeness; their analysis is not critical for GSM.



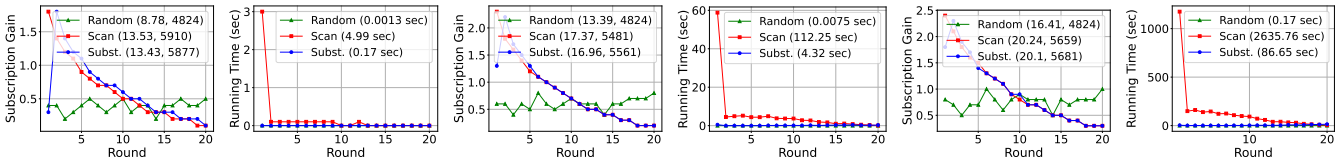
(a) $k = 1$, SG (b) $k = 1$, Time (c) $k = 2$, SG (d) $k = 2$, Time (e) $k = 3$, SG (f) $k = 3$, Time

Figure 2: SG per round and Time per round results of RANDOM, SCAN, and SUBSTITUTE for $|L| = 27$, $|V| = 80$, and $k = 1, 2,$ and 3.



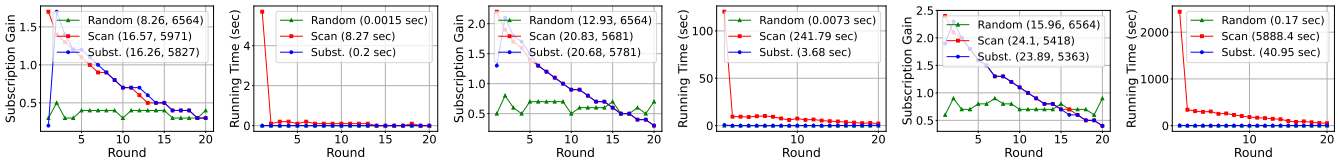
(a) $k = 1$, SG (b) $k = 1$, Time (c) $k = 2$, SG (d) $k = 2$, Time (e) $k = 3$, SG (f) $k = 3$, Time

Figure 3: SG per round and Time per round results of RANDOM, SCAN, and SUBSTITUTE for $|L| = 27$, $|V| = 100$, and $k = 1, 2,$ and 3.



(a) $k = 1$, SG (b) $k = 1$, Time (c) $k = 2$, SG (d) $k = 2$, Time (e) $k = 3$, SG (f) $k = 3$, Time

Figure 4: SG per round and Time per round results of RANDOM, SCAN, and SUBSTITUTE for $|L| = 54$, $|V| = 80$, and $k = 1, 2,$ and 3.



(a) $k = 1$, SG (b) $k = 1$, Time (c) $k = 2$, SG (d) $k = 2$, Time (e) $k = 3$, SG (f) $k = 3$, Time

Figure 5: SG per round and Time per round results of RANDOM, SCAN, and SUBSTITUTE for $|L| = 54$, $|V| = 100$, and $k = 1, 2,$ and 3.

RANDOM but constant, while SCAN performs better as rounds evolve but has a clear scalability issue when any of V, L, or k increases.

Moreover, we remark that in all figures as k grows all solvers achieve higher efficacy (SG) and lower efficiency (time). This is logical as a higher k yields a higher feature weight sum of m users to k features, but also it incurs more feature combinations for processing. Another interesting result is to see what happens when V increases over a fixed L, and what happens when L increases over a fixed V. In the former case, by comparing the Figs. 2 and 3, as also the Figs. 4 and 5, we note that the SG of RANDOM is constant, while the SG of other two solvers clearly improves; a larger selection pool of users is always more beneficial for prioritized solvers. Yet, with more candidate user combinations present for processing, only SUBSTITUTE improves its performance on running time (e.g., see Figs. 2f, 3f, and Figs. 4f, 5f), whereas SCAN heavily deteriorates its performance; more users offer more chances for early termination cases to occur in Line 29 of SUBSTITUTE since it is more likely that some users clearly separate over others. In the latter case, by comparing Figs. 2 and 4, as well as Figs. 3 and 5, we observe that both the efficacy and efficiency of all solvers worsen. The SG deteriorates as a heavier feature weight segmentation in F_v (due to larger L) decreases the feature weights of each user and so incurs lower feature weight sums of m users to k features. Further, a more intense segmentation strengthens the similarity of feature weights in F_v, and so increases the (k, m) candidates not filtered by threshold d. Lastly, the inferior running time is expected due to processing a larger pool of candidate feature combinations.

4 CONCLUSION

We proposed that brands can use a *content recommendation policy* to gain subscribers to their social network pages via messaging. We deployed three algorithms, RANDOM, SCAN, and SUBSTITUTE to this task using a realistic tuning of VK posts. Our thorough experimental study on different user and feature sets verified that SUBSTITUTE outperforms other solvers. To our knowledge, this is the first work to study how brands can gain subscribers using content. In the future, we intend to solve GSM on larger user and feature sets.

ACKNOWLEDGMENTS

This work is supported by the Horizon 2020 Framework Programme, grant agreement No. 957345: "MORE"; by the Operational Program Peloponnesus 2014-2020, SodaSense project; and by the Danish Council for Independent Research, Research Project 9041-00382B.

REFERENCES

[1] Balázs Dezső, Alpár Jüttner, and Péter Kovács. 2011. LEMON – an Open Source C++ Graph Template Library. *ENTCS* 264, 5 (2011), 23–45.

[2] Thibaut Horel and Yaron Singer. 2015. Scalable Methods for Adaptively Seeding a Social Network. In *WWW*. 441–451.

[3] Sergei Ivanov, Konstantinos Theocharidis, Manolis Terrovitis, and Panagiotis Karras. 2017. Content Recommendation for Viral Social Influence. In *SIGIR*. 565–574.

[4] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the Spread of Influence through a Social Network. In *KDD*. 137–146.

[5] Ansh Khurana, Alvis Logins, and Panagiotis Karras. 2020. Selecting Influential Features by a Learnable Content-Aware Linear Threshold Model. In *CIKM*. 635–644.

[6] Paul Lagrée, Olivier Cappé, Bogdan Cautis, and Silviu Maniu. 2019. Algorithms for Online Influencer Marketing. *TKDD* 13, 1 (2019), 1–30.

[7] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence Maximization on Social Graphs: A Survey. *TKDE* 30, 10 (2018), 1852–1872.

[8] Lior Seeman and Yaron Singer. 2013. Adaptive Seeding in Social Networks. In *FOCS*. 459–468.