Online Supplement A: Subgraph Query between t_1 and t_2

Algorithm 1 shows the pseudo-code of the sub-graph query between t_1 and t_2, which is the first portion for all the subsequent queries discussed in this section. In the following queries, the output of the Algorithm 1 will be given as an input. The time complexity of the Algorithm 1 is $O(n+m)$ where n is the number of nodes, and m is the number of edges in original graph.

Algorithm 1 Query Sub-Graph between t_1 and t_2

```
1: Input: G(V, E): graph with n nodes and m edges
2: Input: t1, t2: timestamp
3: Output: G_t1_t2 : Subgraph between t1 and t2
4: define function get_key(val):
5: Initialize key_indx_node, key_indx_edge Lists as empty list()
6: for key, value in networkX.get_attributes(G, 'otype').items() do
7:     if val = value then
8:         key_indx_node.Append(key)
9:         key_indx_edge.Append(key)
10:    end if
11: end for
12: return key_indx_node, key_indx_edge
13: posts, comments, users = get_key(['post', 'comment', 'user'])
14: votes, authoreds, replis = get_key(['vote', 'authored', 'reply'])
15: Initialize subG_vote, subG_reply, subG_authored, user_in, post_in, cm_in Lists as empty list()
16: for vote in votes do
17:     if G.edges()[vote]['time'] ≥ t1 and G.edges()[vote]['time'] ≤ t2 then
18:         subG_vote.Append(vote)
19:         user_in.Append(vote[0])
20:         if G.nodes()[vote[1]]['otype'] = 'comment' then
21:             cm_in.Append(vote[1])
22:         end if
23:     end if
24: end for
25: for reply in replies do
26:     if G.edges()[reply]['time'] ≥ t1 and G.edges()[reply]['time'] ≤ t2 then
27:         subG_reply.Append(reply)
28:         if reply[1] not in cm_in then
29:             cm_in.Append(reply[1])
30:         end if
31:     end if
32: end for
33: for authored in authoreds do
34:     if authored[1] in posts or cm_in) then
35:         subG_authored.Append(authored)
36:     end if
37: end for
38: G_node = G.subgraph(post + cm_in + user_in)
39: Graph = G_node.edge_subgraph(subG_vote + subG_authored + subG_reply) G_t1_t2 = networkX.Graph(Graph)
40: return = G_t1_t2
```

Online Supplement B: Querying on Category of Posts

Algorithm 2 is the query following the subgraph between t_1 and t_2 query to select posts with a specific category within that time frame. The time complexity of this algorithm is $O(p+q)$, where p is the number of nodes and q is the number of edges in subgraph between t_1 and t_2.

Algorithm 2 Query SubGraph between t_1 and t_2 with an specified category

1: **Input:** category: List of category (string)
2: **Input:** $G_t_1_t_2$: Subgraph between t_1 and t_2
3: **Output:** $G_t_1_t_2_category$: Subgraph between t_1 and t_2 with defined posts' categories
4: posts_indices = list(key **for** key, value **in** networkX.get_node_attributes($G_t_1_t_2$,'otype').items() **if** value = 'post')
5: $subG_t_1_t_2_post$ = list(idx **for** idx **in** posts_indices **if** $G_t_1_t_2$.nodes()[idx]['category'] **in** category)
6: Initialize related_node, related_edge, comment_related Lists as empty list()
7: **for** index **in** $subG_t_1_t_2_post$ **do**
8: /*finding related nodes and edges to posts' categories */
9: related_node.Append($subG_t_1_t_2_post$.in_edges(index)[0]))
10: related_edge.Append($subG_t_1_t_2_post$.in_edges(index))
11: comment_related.Append($subG_t_1_t_2_post$.out_edges(index)[1]))
12: /* finding related nodes and edges to the comments of posts' categories */
13: **for** cm_idx **in** comment_related **do**
14: related_node.Append($G_t_1_t_2$.in_edges(cm_inx)[0] **if not in** related_node)
15: related_edge.Append($G_t_1_t_2$.out_edges(cm_inx) **if not in** related_edge)
16: **end for**
17: **end for**
18: G_nodes_category = $G_t_1_t_2$.subgraph(related_node + comment_related + $subG_t_1_t_2_post$)
19: $G_t_1_t_2_category$ = networkX.Graph(G_nodes_category.edge_subgraph(related_edge))
20: **return** $G_t_1_t_2_category$

Online Supplement C: Popularity Query

Algorithm 3 is the query on the subgraph between t_1 and t_2 to sort posts based on their vote and comment count. Its time complexity is $O(plog(p)+pq)$, where p is the number of nodes and q is the number of edges in subgraph between t_1 and t_2. If $O(pq) \gg O(plog(p))$, the time complexity is $O(pq)$, or vice versa.

Algorithm 3 Query to Sort posts based on their vote and comment count

```
1: Input: G_t1_t2 : Subgraph between t1 and t2
2: Output: comments_sorted, votes_sorted: Lists of posts in descending order based on their comment and vote count
3: Initialize vote_count List as empty list()          ▷ stores posts' indices and number of votes they gained
4: Initialize comment_count List as empty list()       ▷ stores posts' indices and number of comments they gained
5: for node_index in G_t1_t2.nodes() do
6:     vote = 0                                        ▷ for each post, vote resets to zero
7:     comment = 0                                     ▷ for each post, comment resets to zero
8:     if G_t1_t2.nodes()[node_index]['otype'] = 'post' then
9:         for [in_edge_index, out_edge_index] in G_t1_t2.out_edges(node_index) do
10:            if G_t1_t2.edges()[node_index,out_edge_index]['otype'] = 'reply' then
11:                comment + = 1
12:            end if
13:        end for
14:        comment_count.Append([node_index, comment])
15:        for [out_edge_index, in_edge_index] in G_t1_t2.in_edges(node_index) do
16:            if G_t1_t2.edges()[out_edge_index, node_index]['otype'] = 'vote' then
17:                vote + = 1
18:            end if
19:        end for
20:        vote_count.Append([node_index, vote])
21:    end if
22: end for
23: comments_sorted = sorted(comment_count, key=lambda x: x[1], reverse = True)
24: votes_sorted = sorted(vote_count, key=lambda x: x[1], reverse = True)
25: return comments_sorted, votes_sorted
```

Online Supplement D: Actively Engaged Query

Algorithm 4 is the query on the subgraph between t_1 and t_2 to sort users based on their activities within that time interval. Its time complexity is $O(UlogU)$, where U is the number of users in the input list.

Algorithm 4 Query to Sort users based on Activities

1: **Input:** users : List of user node type indices in original graph
2: **Input:** $G_t_1_t_2$: Subgraph between t_1 and t_2
3: **Output:** user_activity_sorted: List of users in descending order who are actively engaged
4: Initialize user_activity List as empty list() ▷ stores users and count of their activities
5: **for** user **in** users **do**
6:     **if** user **in** $G_t_1_t_2$.nodes() **then**
7:         user_activity.Append([user, len($G_t_1_t_2$.out_edges(user))])
8:     **end if**
9: **end for**
10: user_activity_sorted = sorted(user_activity, key=lambda x: x[1], reverse = **True**)
11: **return** user_activity_sorted

Online Supplement E: Query Performance

The defined queries have a low execution time compared to the size of the graph (some examples illustrated run times on a single machine; here we present query performance results on a compute cluster). Figure 10 illustrates the execution time of each query for a different number of an initial set of posts on 1 node with 128 AMD Epyc 7702 CPU @ 2.0GHz processors and 1 TB memory. We focused on the number of *posts* in each run and implemented four queries for those posts. We considered the same t_1 and t_2 for all the queries. For the second query, we retrieved posts with the ”Steemit” category. As demonstrated in Figure 10, the query execution time even for 10K posts is below 6 seconds.

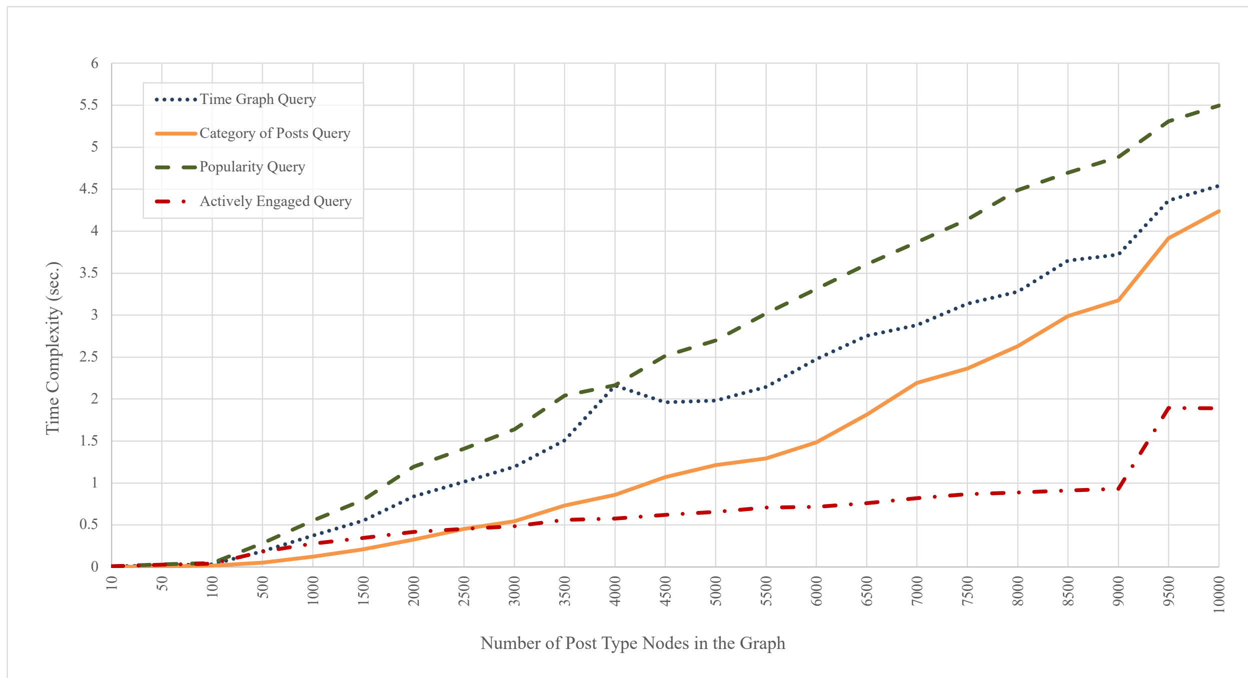

Figure 10 Query time complexity

Online Supplement F: Meta-paths LP results on Health Steemit Graph

Table 13 Health Steemit Graph: Various train-test split rates arranged by the time of post creation. The section on "Temporal Homogeneous Graph" lacks meta-paths, while the "Valid Meta-paths" and "Refined Valid Meta-paths" sections are based on proposed temporal meta-paths. "Refined Valid Meta-paths" encompass posts with at least one previous connection

Health Category	70-30 train-test split rate											
	Temporal Homogeneous Graph				Valid Meta-paths				Refined Valid Meta-paths			
	# Labels (test set)	Baseline	# TP Labels	Accuracy	# Labels (test set)	Baseline	# TP Labels	Accuracy	# Labels (test set)	Baseline	# TP Labels	Accuracy
Cancer	398	3.88%	20	5.03%	42	7.32%	37	88.10%	17	2.96%	12	70.59%
Diet	1244	12.13%	14	1.13%	37	6.45%	5	13.51%	13	2.26%	5	38.46%
Drugs	388	3.78%	22	5.67%	25	4.36%	3	12.00%	7	1.22%	3	42.86%
Health	783	7.63%	96	12.26%	50	8.71%	0	0.00%	11	1.92%	0	0.00%
Healthcare	766	7.47%	42	5.48%	99	17.25%	10	10.10%	70	12.20%	10	14.29%
Healthy	2079	20.27%	159	7.65%	60	10.45%	5	8.33%	19	3.31%	4	21.05%
Medical	319	3.11%	74	23.20%	63	10.98%	5	7.94%	21	3.66%	5	23.81%
Medicine	577	5.63%	307	53.21%	32	5.57%	2	6.25%	4	0.70%	1	25.00%
Meditation	2412	23.52%	995	41.25%	76	13.24%	9	11.84%	37	6.45%	8	21.62%
Yoga	1291	12.59%	354	27.42%	90	15.68%	45	50.00%	65	11.32%	40	61.54%
Total	**10257**	**-**	**2083**	**20.31%**	**574**	**-**	**121**	**21.08%**	**264**	**-**	**88**	**33.33%**

Health Category	80-20 train-test split rate											
	Temporal Homogeneous Graph				Valid Meta-paths				Refined Valid Meta-paths			
	# Labels (test set)	Baseline	# TP Labels	Accuracy	# Labels (test set)	Baseline	# TP Labels	Accuracy	# Labels (test set)	Baseline	# TP Labels	Accuracy
Cancer	366	4.52%	4	1.09%	33	8.62%	26	78.79%	16	4.18%	9	56.25%
Diet	879	10.85%	65	7.39%	22	5.74%	2	9.09%	6	1.57%	2	33.33%
Drugs	341	4.21%	10	2.93%	17	4.44%	2	11.76%	5	1.31%	2	40.00%
Health	547	6.75%	99	18.10%	30	7.83%	0	0.00%	7	1.83%	0	0.00%
Healthcare	479	5.91%	26	5.43%	75	19.58%	19	25.33%	57	14.88%	19	33.33%
Healthy	1662	20.52%	133	8.00%	32	8.36%	3	9.38%	9	2.35%	3	33.33%
Medical	145	1.79%	46	31.72%	34	8.88%	5	14.71%	12	3.13%	5	41.67%
Medicine	454	5.60%	176	38.77%	18	4.70%	2	11.11%	4	1.04%	1	25.00%
Meditation	2241	27.67%	618	27.58%	48	12.53%	13	27.08%	33	8.62%	11	33.33%
Yoga	986	12.17%	315	31.95%	74	19.32%	31	41.89%	57	14.88%	29	50.88%
Total	**8100**	**-**	**1492**	**18.42%**	**383**	**-**	**103**	**26.89%**	**206**	**-**	**81**	**39.32%**

Health Category	90-10 train-test split rate											
	Temporal Homogeneous Graph				Valid Meta-paths				Refined Valid Meta-paths			
	# Labels (test set)	Baseline	# TP Labels	Accuracy	# Labels (test set)	Baseline	# TP Labels	Accuracy	# Labels (test set)	Baseline	# TP Labels	Accuracy
Cancer	294	6.20%	5	1.70%	17	8.90%	15	88.24%	10	5.24%	8	80.00%
Diet	816	17.20%	13	1.59%	11	5.76%	2	18.18%	4	2.09%	2	50.00%
Drugs	255	5.38%	4	1.57%	9	4.71%	0	0.00%	1	0.52%	0	0.00%
Health	371	7.82%	41	11.05%	16	8.38%	0	0.00%	5	2.62%	0	0.00%
Healthcare	279	5.88%	111	39.78%	39	20.42%	19	48.72%	32	16.75%	19	59.38%
Healthy	204	4.30%	57	27.94%	17	8.90%	1	5.88%	4	2.09%	1	25.00%
Medical	94	1.98%	20	21.28%	20	10.47%	3	15.00%	7	3.66%	3	42.86%
Medicine	240	5.06%	23	9.58%	11	5.76%	1	9.09%	2	1.05%	1	50.00%
Meditation	1860	39.21%	363	19.52%	21	10.99%	1	4.76%	9	4.71%	1	11.11%
Yoga	331	6.98%	233	70.39%	30	15.71%	17	56.67%	22	11.52%	15	68.18%
Total	**4744**	**-**	**870**	**18.34%**	**191**	**-**	**59**	**30.89%**	**96**	**-**	**50**	**52.08%**