Paper Review: Efficient Maximum k-Plex Computation over Large Sparse Graphs

Introduction

Graph models are widely applied in data analysis fields such as social media, communication networks, collaboration networks, web graphs, and the internet because they naturally capture relationships between entities. The graph data in these applications are typically globally sparse but locally dense. Identifying locally dense subgraphs is important in many applications, such as identifying large-scale community structures in social networks to understand interaction patterns within the network.

A classic concept of dense subgraphs is the clique, which requires that every pair of vertices has an edge between them. However, the concept of a clique is too strict, and large tightly connected communities in real networks rarely manifest as cliques. Therefore, various relaxed clique models have been proposed in the literature, such as the k-plex. A k-plex allows each vertex to miss up to k neighbors. The problem of computing the maximum k-plex and enumerating the maximum k-plex has recently gained increasing attention. However, existing algorithms still have limitations in efficiency, especially when dealing with large sparse graphs.

Major Contributions

1. **New Framework**: A new framework is designed for computing the maximum k-plex on large sparse graphs by iteratively extracting small dense subgraphs from the graph and solving the problem for each extracted subgraph using a branch-and-bound search.
2. **CTCP Algorithm**: An efficient graph reduction algorithm, CTCP, is proposed to reduce the size of the input graph by pruning vertices and edges. CTCP computes a reduced graph with lower time complexity than existing techniques.
3. **BBMatrix Algorithm**: A new branch-and-bound algorithm, BBMatrix, is developed for the dense subgraphs extracted from the input graph. It uses an adjacency matrix representation of its input graph and utilizes first-order and second-order information for pruning and upper bound estimation.

Framework Details

- **Framework Description**: The framework is divided into two parts:
    1. **Part One**: Compute an initial larger k-plex and reduce the graph using CTCP. CTCP reduces the graph to its (lb + 1 - k)-core and (lb + 1 - 2k)-truss by removing vertices with degrees less than lb + 1 - k and edges participating in fewer than lb + 1 - 2k triangles.
    2. **Part Two**: Iteratively extract small dense subgraphs from the graph and compute the maximum k-plex containing a specified vertex using BBMatrix.

kPlexS Algorithm Details

kPlexS is a new algorithm designed for the maximum k-plex computation problem. This algorithm efficiently computes the maximum k-plex on large sparse graphs by combining the core techniques of CTCP and BBMatrix.

1. CTCP Algorithm

CTCP (Core-Truss Co-Pruning) is a graph reduction algorithm aimed at reducing the size of the input graph through the following two steps:

- **Core Pruning**: Remove all vertices with a degree less than lb + 1 - k, reducing the graph to its (lb + 1 - k)-core.
- **Truss Pruning**: Remove all edges participating in fewer than lb + 1 - 2k triangles, reducing the graph to its (lb + 1 - 2k)-truss.

The time complexity of CTCP is O(δ(G) × |E|), where δ(G) is the degeneracy of the graph. CTCP is used not only in the preprocessing step but also iteratively called after each vertex removal to further reduce the graph's size.

2. BBMatrix Algorithm

BBMatrix is a branch-and-bound algorithm specifically designed for the dense subgraphs extracted from the input graph. It uses an adjacency matrix to represent its input graph and utilizes first-order and second-order information for pruning and upper bound estimation. The core techniques of BBMatrix include:

- **Adjacency Matrix Representation**: BBMatrix uses an adjacency matrix to represent its input graph, making it more efficient to incrementally maintain first-order and second-order information.
- **Branching and Pruning Rules**: BBMatrix optimizes the branch-and-bound search process using a set of branching and pruning rules. It leverages first-order and second-order information for upper bound estimation and pruning, enhancing the algorithm's efficiency.
- **Incremental Calculation Techniques**: During the recursive process, BBMatrix employs incremental techniques to efficiently apply upper bound estimations and pruning.

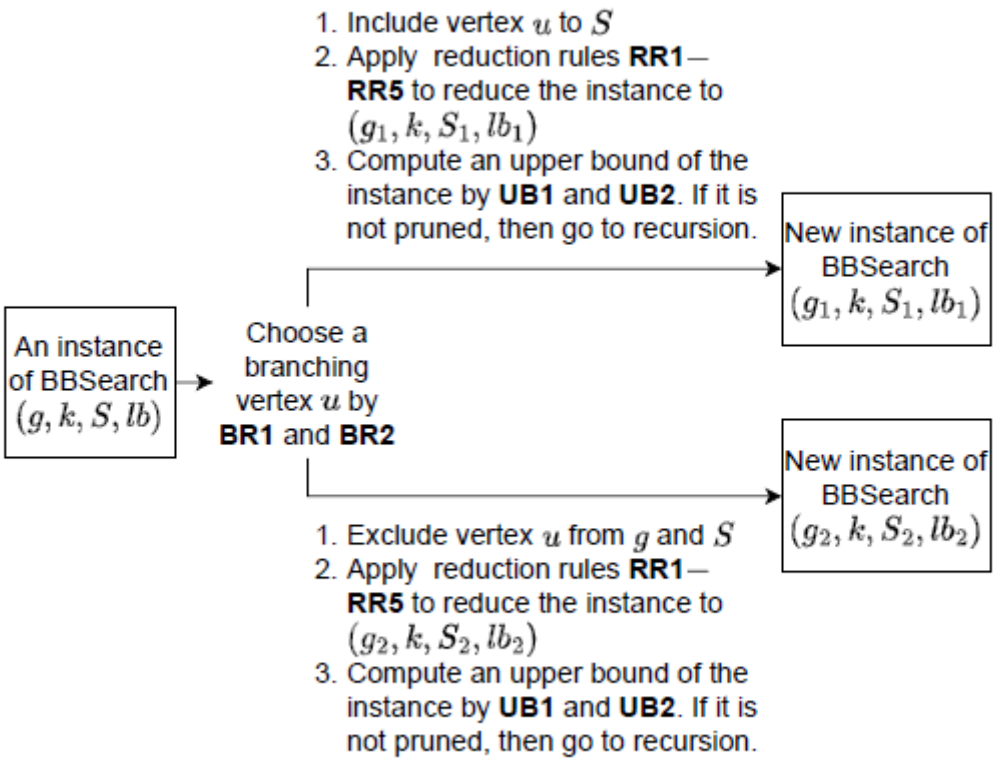

Figure 2: An overview of the flow of the BBSearch algorithm

The above image illustrates the flow of the BBSearch algorithm, primarily used to compute the maximum k-plex in a graph. The process is as follows:

1. **Start**: Begin with a BBSearch instance (g, k, S, lb), where g is the graph, k is the allowed maximum number of missing neighbors, S is the current partial solution, and lb is the lower bound.
2. **Select Vertex**: Choose a branching vertex u through the branching rules (BR1 and BR2).
3. **Include Vertex u**:
    - Add vertex u to the partial solution S.
    - Apply reduction rules (RR1-RR5) to simplify the instance.
    - Compute the upper bound of the instance (UB1 and UB2). If not pruned, recursively process the new instance.
4. **Exclude Vertex u**:
    - Exclude vertex u from the graph g and the partial solution S.
    - Apply reduction rules (RR1-RR5) to simplify the instance.
    - Compute the upper bound of the instance (UB1 and UB2). If not pruned, recursively process the new instance.

This demonstrates how to recursively select vertices and use reduction rules to compute the maximum k-plex.

Experimental Results

- **Experimental Setup**: Extensive experiments were conducted on two benchmark graph sets. The results show that the kPlexS algorithm consistently solves more graph instances within the given time limits compared to state-of-the-art algorithms (BnB, Maplex, and KpLeX).

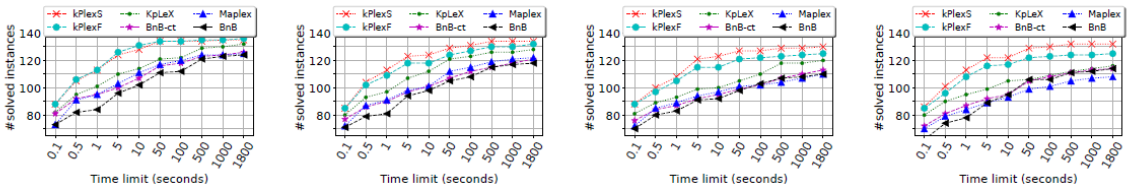

Figure 3: Number of solved instances for real-world graphs (vary time limit, best viewed in color)

The image above compares the number of instances solved by six algorithms (kPlexS, kPlexF, KpLeX, BnB-ct, Maplex, BnB) on real graph sets. The results show that kPlexS and kPlexF consistently solve the most instances within different time limits, with kPlexS solving more instances within 50 seconds than other algorithms do within 1800 seconds. The superiority of kPlexS comes from its application of second-order techniques in dense subgraph extraction and branch-and-bound search.

Table 8: Preprocessing for $k = 5$ (time in seconds, P is the heuristically computed k-plex, (V_K, E_K) is the reduced graph)

ID	kPlexS				KpLeX				Maplex				BnB			
	Time	$\|P\|$	$\|V_K\|$	$\|E_K\|$	Time	$\|P\|$	$\|V_K\|$	$\|E_K\|$	Time	$\|P\|$	$\|V_K\|$	$\|E_K\|$	Time	$\|P\|$	$\|V_K\|$	$\|E_K\|$
G_1	0.23	43	209	5824	0.15	43	209	5824	0.38	44	151	4183	1.24	47	60	1634
G_2	1.20	1236	1239	766923	2.42	1236	1239	766923	1.89	1236	1239	766923	349.73	1236	1239	766923
G_3	0.01	75	120	6339	0.01	75	128	6976	0.04	75	128	6976	0.10	75	121	6434
G_4	0.66	50	291	9368	0.30	50	292	9421	0.97	50	292	9421	4.88	50	292	9771
G_5	1.02	67	114	5405	0.43	68	111	5167	1.32	67	118	5686	8.03	68	111	5256
G_6	0.12	24	51153	1205204	0.54	24	51153	1205309	1.32	24	51153	1205309	3.31	24	51154	1231573
G_7	0.02	40	102	3521	0.02	39	111	3944	0.05	40	102	3521	0.10	39	113	4131
G_8	0.02	6	61351	95242	0.01	5	91813	125704	0.03	6	61351	95242	0.04	5	91813	125704
G_9	0.09	23	527	21384	0.21	21	705	29568	0.56	24	413	16478	1.77	21	715	34612
G_{10}	0.64	36	60849	1982268	1.01	27	94795	3255707	24.82	36	60849	1982268	19.37	27	94795	3258857
G_{11}	0.13	28	452	13564	0.17	27	532	16438	0.38	29	370	10738	1.80	28	460	15327
G_{12}	0.36	24	212326	5507216	4.84	24	212326	5507216	5.66	24	212326	5507216	12.43	24	212461	5524384
G_{13}	0.71	21	404785	9371658	-	-	-	-	1.51	21	404785	9371658	-	-	-	-
G_{14}	0.36	22	660	22114	0.46	22	663	22187	1.37	22	663	22187	4.80	22	700	27964
G_{15}	0.48	23	673	22154	0.61	22	984	33637	1.76	23	702	23009	9.89	22	1015	41671
G_{16}	0.82	25	660	29604	0.79	25	661	29633	1.75	25	661	29633	10.15	25	677	35674
G_{17}	17.23	27	814	17119	9.14	27	825	17403	27.01	27	825	17403	55.07	30	298	6207
G_{18}	0.50	14	4818	82808	0.88	14	4820	82848	2.00	15	3586	67144	3.32	14	4867	88046
G_{19}	0.36	46	244	10546	-	-	-	-	0.09	48	226	9407	-	-	-	-
G_{20}	23.96	21	33082	845687	30.80	21	33103	846392	39.29	23	17124	452253	240.25	21	34767	1318592
G_{21}	21.72	33	775	21559	11.86	33	777	21635	30.90	33	777	21635	87.22	33	819	27622
G_{22}	15.23	13	0	0	12.90	12	578	4556	25.94	13	0	0	21.68	13	0	0

The table above presents the preprocessing results of four algorithms (kPlexS, KpLeX, Maplex, BnB) on 22 real graphs for k = 5. The data include preprocessing time, the k-plex size computed heuristically (P), and the number of vertices ($|V_k|$) and edges (E_k) in the reduced graph. The results show that kPlexS has the fastest preprocessing time on most graphs and significantly reduces the number of vertices and edges in the simplified graph. KpLeX's preprocessing time is comparable to kPlexS, but its performance is slightly lower on some graphs. Maplex and BnB have relatively longer preprocessing times, especially BnB on some graphs.

Conclusion

This paper proposes a new algorithm, kPlexS, which significantly improves the efficiency of computing the maximum k-plex on large sparse graphs by introducing new frameworks and techniques. Combining the core technologies of CTCP and BBMatrix, kPlexS demonstrates significant performance advantages in experiments and holds substantial practical value in various application domains. Future research can further optimize and extend this algorithm for dynamic graphs, distributed computing, and heterogeneous graphs. Additionally, kPlexS can be widely applied in social network analysis, bioinformatics, communication network optimization, e-commerce and recommendation systems, knowledge graph construction, financial risk management, and cybersecurity. For example, in social network analysis, it can help identify communities and assess influence; in bioinformatics, it can identify functional modules and gene regulatory mechanisms; and in communication networks, it can optimize network topology and traffic management. These applications showcase kPlexS's potential and broad application prospects in handling large-scale graph data.