Listing Maximal k-Plexes in Large Real-World Graphs

Zhengren Wang, Yi Zhou, Mingyu Xiao, Bakhadyr Khoussainov

Zhengren Wang

Algorithms and Logic Group in UESTC

April 1, 2022

1. Background
2. Algorithms
3. Implementation Techniques
4. Experiments

Finding Cohesive Groups

Finding ***cohesive groups*** (or ***communities***) has received attention from various areas.

- In the WWW, identify clients sharing similar interests and serve them with a common proxy to reduce network traffic.
- In social networks, discover closely related individuals.
- In biological networks, predict the structure and function of protein.

Clique Model

Naturally, cohesive groups can be modeled with ***Cliques***.
A clique is a subgraph where vertices are pairwise connected, i.e., a complete graph.

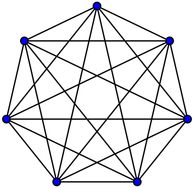


Figure 1: K_7

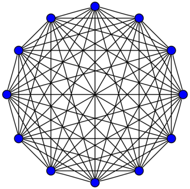


Figure 2: K_{12}

k-Plex Model

Due to various reasons like *data noise*, communities rarely appear as cliques.
k-Plexes allow every vertex missing at most $k-1$ links to other vertices.

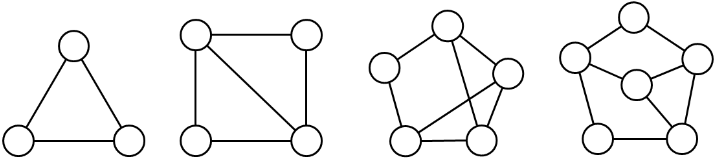


Figure 3: 1, 2, 3, 4-plex

Properties

Lemma (Hereditary Property)

- *Any induced subgraph of a k-plex is still a k-plex.*
- *A k-plex is maximal if it is not a subgraph of any larger k-plex.*

Lemma (Distance Property)

- *Any k-plex with at least $2k - 1$ vertices has its diameter at most 2.*
- *A k-plex with at most $2k - 2$ vertices may be unconnected.*

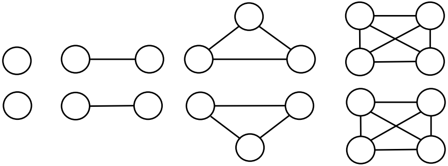


Figure 4: unconnected $2, 3, 4, 5$-plex

Translated Problems

We model cohesive groups with ***maximal k-plexes***.

Problem (Listing maximal k-plexes)

Given a graph G, a positive integer k, list all maximal k-plexes from G.

Problem (Listing large maximal k-plexes)

Given a graph G, two positive integers k and l where $l \geq 2k - 1$, list all maximal k-plexes with at least l vertices.

The Bron-Kerbosch Algorithm

Our algorithm stems from the Bron-Kerbosch algorithm, say *BKPlex*.
BKPlex accepts three sets P, C and X as parameters,

- P: ***plex vertices*** of the growing k-plex,
- C: ***candidate vertices*** for further branching,
- X: ***excluded vertices*** to avoid non-maximal solutions.

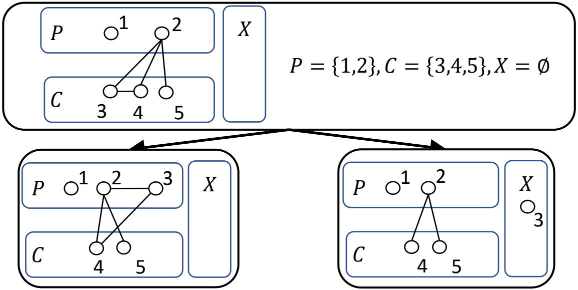
then lists maximal k-plexes $G[P']$ with three properties:

- $P \subseteq P'$
- $P' \subseteq P \cup C$
- $\forall v \in X$, the subgraph $G[\{v\} \cup P']$ is not a k-plex.

Hereditary Prop. Any induced subgraph of a k-plex is still a k-plex.

The Bron-Kerbosch Algorithm

BKPlex branches by doing bipartition recursively. [1]



Vtx	1	2	3	4	5
Br1	P	P	**P**	**C**	**C**
Br2	P	P	**X**	**C**	**C**

Figure 5: An example of BKPlex.

[1] Simplified for clarity. In fact, a variant of bipartition.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

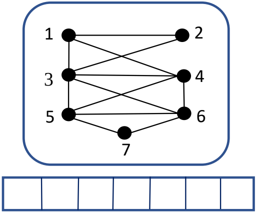


Figure 6: An example of degeneracy ordering.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

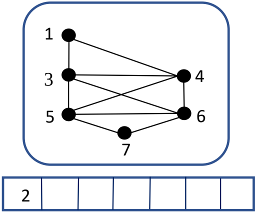


Figure 6: An example of degeneracy ordering.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

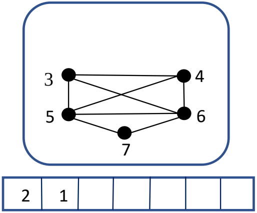


Figure 6: An example of degeneracy ordering.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

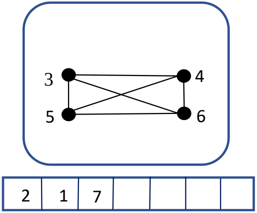


Figure 6: An example of degeneracy ordering.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

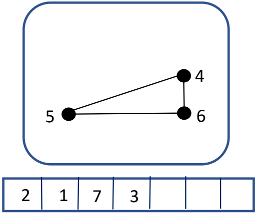


Figure 6: An example of degeneracy ordering.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

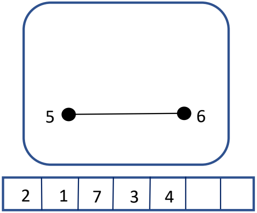


Figure 6: An example of degeneracy ordering.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

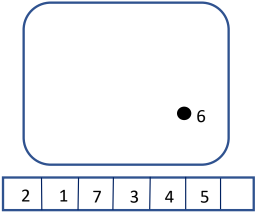


Figure 6: An example of degeneracy ordering.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

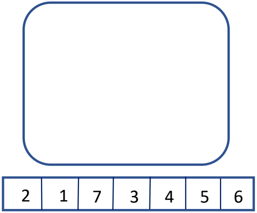


Figure 6: An example of degeneracy ordering.

Degeneracy Ordering

- $\eta = v_1, \ldots, v_n$ is called *degeneracy ordering* (*core ordering*) if each vertex v_i has the minimum degree in the induced subgraph $G[\{v_i, \ldots, v_n\}]$.
- Given a degeneracy ordering $\eta = v_1, \ldots, v_n$, the degree of v_i in $G[\{v_i, \ldots, v_n\}]$ is called the *core number* of v_i.
- For any degeneracy ordering of the same graph, the largest core number among all vertices is a constant D called *degeneracy*.
- Due to the sparsity of many real-world graphs, $D \ll \Delta \ll n$ where Δ is the maximum degree.

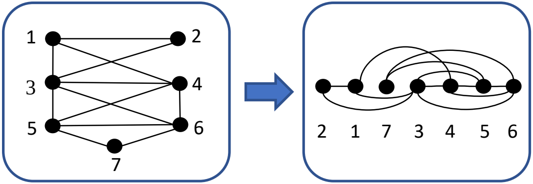


Figure 6: An example of degeneracy ordering.

Recent Algorithms [3]

Pivot Heuristic Zhou et al. (2020) proposed *BKPivot* with a *pivot* heuristic in the branch scheme of BKPlex, and improved its running time from $O^*(2^n)$ to $O^*(\gamma_k^n)$, where $\gamma_k < 2$. [2]

Graph Decomposition Conte et al. (2018) proposed *D2K*, a decomposition-based algorithm for listing k-plexes with the diameter at most 2.
For each vertex, D2K builds a local subgraph and then adopts BKPlex to list maximal k-plexes locally which runs in $O^*(2^{D\Delta})$.

Distance Prop. Any k-plex with at least $2k-1$ vertices has the diameter at most 2.

[2] The notation O^* omits the polynomial factors.
[3] Simplified for clarity.

Recent Algorithms [3]

Pivot Heuristic Zhou et al. (2020) proposed *BKPivot* with a *pivot* heuristic in the branch scheme of BKPlex, and improved its running time from $O^*(2^n)$ to $O^*(\gamma_k^n)$, where $\gamma_k < 2$. [2]

Graph Decomposition Conte et al. (2018) proposed *D2K*, a decomposition-based algorithm for listing k-plexes with the diameter at most 2.
For each vertex, D2K builds a local subgraph and then adopts BKPlex to list maximal k-plexes locally which runs in $O^*(2^{D\Delta})$.

Distance Prop. Any k-plex with at least $2k-1$ vertices has the diameter at most 2.

We combine them and propose *ListPlex* running in $O^*(\gamma_k^D)$.

[2] The notation O^* omits the polynomial factors.
[3] Simplified for clarity.

ListPlex
Listing All Maximal k-Plexes

Based on Distance Property, ListPlex divides its task into two parts.

- *Part I*: k-plexes with size at most $2k-2$. (Solved directly by BKPlex)
- ***Part II***: k-plexes with size at least $2k-1$.

Distance Prop. Any k-plex with at least $2k-1$ vertices has the diameter at most 2.

ListPlex
Part II

Procedure

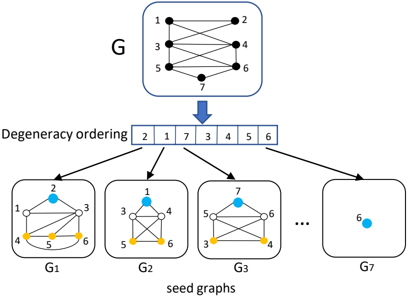
1) Sort V by a degeneracy ordering $\eta = v_1 \dots v_n$.

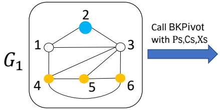
- $N^k(v)$ denotes k-hop neighbors of v.
- $N^k_{\succ}(v_i)$ denotes $N^k(v_i) \cap \{v_{i+1}, ..., v_n\}$, say ***forward*** k-hop neighbors of v_i

2) From v_1 to v_n, in the i-th iteration, list maximal k-plexes with i as the minimum index of vertices.

- Build ***seed graph*** $G_i = G[\{v_i\} \cup N_{\succ}(v_i) \cup N^2_{\succ}(v_i)]$.
- Call BKPivot with given combinations of $N^2_{\succ}(v_i)$, say ***seed set*** S.
- Validate maximality in G for k-plexes generated in G_i.

ListPlex
Part II



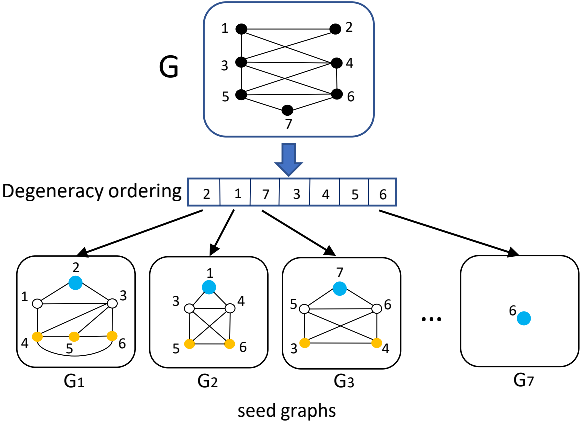


S	Ps	Cs	Xs
∅	{2}	{1,3}	{4,5,6}
{4}	{2,4}	{1,3}	{5,6}
{5}	{2,5}	{1,3}	{4,6}
{6}	{2,6}	{1,3}	{4,5}
{4,5}	{2,4,5}	{1,3}	{6}
{4,6}	{2,4,6}	{1,3}	{5}
{5,6}	{2,5,6}	{1,3}	{4}

Figure 7: An example of ListPlex's Part II.
(L) sort V in degeneracy ordering η and induce seed graphs G_i for each $v_i \in \eta$.
(R) enumerate $S \subseteq N^2_{\succ}(v_i)$ with bound $|S| \leq k-1$ ($k=3$) and call BKPivot with P_s, C_s, X_s.

ListPlex

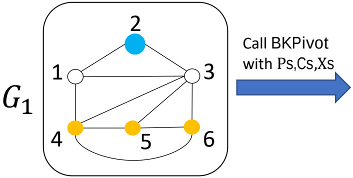
Graph Decompostion



- Cohesive groups appear locally in large real-world graphs,
- Parallelism,
- Smaller scale and better locality,
- $D \ll \Delta$.

ListPlex

Seed Set



S	Ps	Cs	Xs
∅	{2}	{1,3}	{4,5,6}
{4}	{2,4}	{1,3}	{5,6}
{5}	{2,5}	{1,3}	{4,6}
{6}	{2,6}	{1,3}	{4,5}
{4,5}	{2,4,5}	{1,3}	{6}
{4,6}	{2,4,6}	{1,3}	{5}
{5,6}	{2,5,6}	{1,3}	{4}

- At most $k-1$ vertices come from $N^2_{\succ}(v_i)$,
- Reducing candidates fast,
- $|N^2_{\succ}(v_i)|$ is potentially $D\Delta$,
- Pruning rules.

With lower bound l, a related lower bound l' can be derived.

Lemma

Assume $|P| \geq l$, for any vertex pair $u, v \in P$, $|N(u) \cap N(v) \cap P| \geq l'$

Removing unfruitful candidates from G_i reduces the scale of G_i.

- Consider vertex pair (v_i, u), $u \in V_i$.

Dropping unfruitful seed sets S saves the forthcoming exponential search.

- Consider vertex pair (u, v) from some S.

1. Background
2. Algorithms
3. **Implementation Techniques**
4. Experiments

Reducing cache misses

Validating maximality in G for maximal k-plex of G_i suffers a high amount of cache misses.

Alleviation:

- Build a bipartite graph B_i for each G_i, which serves as a cache of currently useful data of large G,
- Pruning bipartite graph just like seed graph.

Parallelization

ListPlex owns appealing parallel features.

How To:

- Parallelize searches of maximal k-plexes on each G_i, say T_i,
- Split T_i when some cores are idle for better load balance,
- Construct degeneracy ordering and perform generated T_i in parallel.

1. Background
2. Algorithms
3. Implementation Techniques
4. Experiments

Dataset

All graphs are taken from SNAP and LAW [4].

Table 1: Considered networks and their properties

Network	n	m	Δ	D
jazz	198	2742	100	29
ca-grqc	5241	14484	81	43
gnutella08	6301	41554	97	10
wiki-vote	7116	100763	1065	53
lastfm	7624	55612	216	20
as-caida	26475	53381	2628	22
soc-epinions	75888	405739	3044	67
soc-slashdot	82144	500480	2548	54
email-euall	265214	365569	7636	37
amazon0505	410236	2439436	2760	10
in-2004	1353703	13126172	21869	488
soc-pokec	1632803	22301964	14854	47
as-skitter	1696415	11095298	35455	111
soc-livejournal	4847571	68993773	14815	360
arabic-2005	22744080	639999458	575628	3247
uk-2005	39459925	936364282	1372171	584
it-2004	41291594	1150725436	1243927	3209
webbase-2001	118142155	1019903190	816127	1506

[4] http://law.di.unimi.it/

Listing All Maximal k-Plexes

Table 2: Listing all maximal k-plexes in small graphs

Network	k	#k-plexes	The running time (s)				Speedup
			BKPlex	BKPivot	ListPlex	ListPlex(16)	
jazz	2	35214	648.864	0.29	**0.086**	0.408	0.211
jazz	3	3602575	772.826	17.55	**6.477**	0.832	7.785
jazz	4	193056583	3226.746	829.40	**417.646**	26.187	15.949
ca-grqc	2	13718439	OOT	1858.02	**649.985**	40.880	15.899
gnutella08	2	19866959	1500.208	3627.57	**1117.858**	70.207	15.922
wiki-vote	2	66193264	10356.553	10671.92	**1526.884**	95.656	15.962
lastfm	2	29086855	2643.394	6676.89	**1989.701**	124.525	15.978

Listing Large Maximal k-Plexes

Table 3: The running time of listing large maximal k-plexes from small and medium graphs by CommuPlex[5], D2K[6] and ListPlex.

Graph	k	l	#k-plexes	The running time (s)		
$(\lvert V\rvert, \lvert E\rvert)$				CommuPlex	D2K	ListPlex
jazz (198, 2742)	4	12	2745953	25.218	33.054	**4.498**
lastfm (7624, 55612)	4	12	1827337	20.724	23.991	**4.586**
as-caida (26475, 53381)	3	12	281251	5.684	13.421	**0.867**
	4	12	15939891	300.388	785.506	**47.98**
amazon0505 (410236, 2439436)	2	12	376	1.825	0.641	**0.137**
	3	12	6347	11.359	0.77	**0.286**
	4	12	105649	47.049	5.338	**1.171**
email-euall (265214, 365569)	2	12	412779	8.793	11.199	**1.946**
	3	12	32639016	619.384	1043.266	**91.62**
		20	2637	10.754	53.691	**0.429**
	4	20	1707177	825.126	3800.889	**24.089**
soc-slashdot (82144, 500480)	2	12	27208777	376.071	213.141	**59.42**
		20	11411028	227.016	137.159	**32.988**
		30	453	10.77	16.481	**0.688**
	3	12	2807943240	OOT	26029.006	**7813.045**
		20	1303148522	28361.707	15308.777	**4538.022**
		30	1679468	699.876	2066.598	**51.364**
	4	30	502699966	OOT	OOT	**6680.261**
as-skitter (1696415, 11095298)	2	50	47969775	OOT	OOT	**520.884**
	2	100	0	1.793	2.951	**0.716**
	3	50	21070497438	OOT	OOT	OOT
	3	100	0	2.37	3.285	**0.718**
in-2004 (1353703, 13126172)	2	50	25855779	7663.843	576.06	**150.212**
	2	100	9978037	5899.638	256.225	**72.063**
	3	50	29045783792	OOT	OOT	OOT
	3	100	4257410159	OOT	OOT	**28384.76**

Graph	k	l	#k-plexes	The running time (s)		
$(\lvert V\rvert, \lvert E\rvert)$				CommuPlex	D2K	ListPlex
wiki-vote (7116, 100763)	2	12	2919931	75.871	115.757	**17.653**
		20	52	4.52	11.289	**0.591**
		30	0	1.033	**0.027**	0.091
	3	12	458153397	OOT	OOT	**2185.598**
		20	156727	595.636	1852.186	**9.384**
		30	0	1.072	**0.029**	0.1
	4	20	46729532	OOT	OOT	**1174.2**
		30	0	9.17	3.627	**0.112**
soc-pokec (1632803, 22301964)	2	12	7679906	1537.506	172.987	**47.475**
		20	94184	1064.371	20.03	**15.161**
		30	3	662.64	**8.637**	9.557
	3	12	520888893	OOT	OOT	**1607.285**
		20	5911456	1470.536	856.393	**46.262**
		30	5	717.425	**9.993**	10.127
	4	20	318035938	34048.155	OOT	**1825.216**
		30	4515	1140.117	111.987	**11.211**
soc-epinions (75888, 405739)	2	12	49823056	843.9	735.589	**193.307**
		20	3322167	137.427	180.061	**19.382**
		30	0	8.995	12.109	**0.492**
	3	20	548634119	27037.614	35525.693	**3072.267**
		30	16066	546.69	2591.439	**6.123**
	4	30	13172906	OOT	OOT	**661.103**
com-livejournal (4847571, 68993773)	2	340	650322	2284.435	OOT	**109.382**
	2	345	0	57.548	13589.487	**6.914**
	3	340	555718694	OOT	OOT	**22863.467**
	3	345	3963139	24861.871	OOT	**826.183**

[5] Zhou et al. (2020)
[6] Conte et al. (2018)

Listing Large Maximal k-Plexes

Table 4: The parallel running time of large networks by ListPlex and D2K with 16 threads.

Graph ($\lvert V\rvert$, $\lvert E\rvert$)	k	l	#k-plexes	The running time (s) D2K(16)	ListPlex(16)
arabic-2005 (22744080, 639999458)	2	800	224870903	2195.272	**714.159**
	2	1000	236897	151.328	**40.202**
	3	800	>25062182205	OOT	OOT
	3	1000	34155502	587.967	**128.737**
uk-2005 (39459925, 936364282)	2	250	106243475	OOT	**355.855**
	2	500	256406	318.118	**35.001**
	3	250	>18336111409	OOT	OOT
	3	500	28199814	9506.661	**121.726**
it-2004 (41291594, 1150725436)	2	2000	675111	340.904	**41.983**
	2	3000	675111	307.735	**38.468**
	3	2000	197679229	4254.456	**724.979**
	3	3000	197679229	4235.389	**715.002**
webbase-2001 (118142155, 1019903190)	2	800	1599005	374.134	**54.19**
	2	1000	1164383	346.393	**53.651**
	3	800	1785341050	36116.817	**5521.386**
	3	1000	1484341137	35005.343	**6960.816**

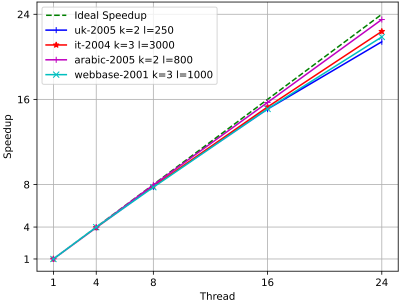


Figure 8: The speedup of ListPlex for the large graphs with different parameters.

Excluding unfruitful seed sets

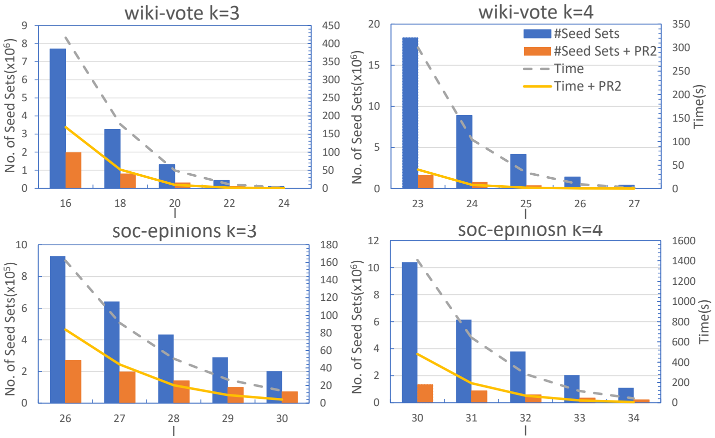


Figure 9: The number of seed sets and running time with and without Prune Rule 2.

Reducing cache misses

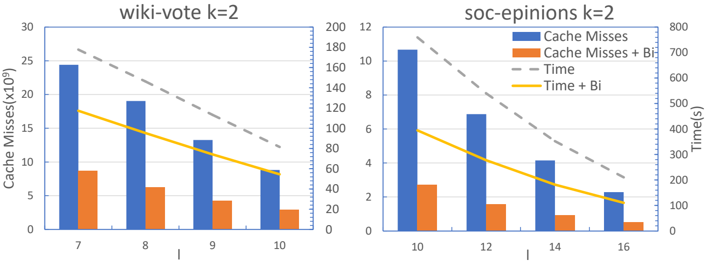


Figure 10: The total number of data cache misses and the running time with and without using bipartite graph B_i.

Q&A

Thanks!