Clique Cover Problem

Implementation of Algorithms
Result Analysis

Group No: 10

Student ID:
1505002
1505044
1505057
1505097
1505101

Vertex Clique Cover

Clique Cover Decision Problem:

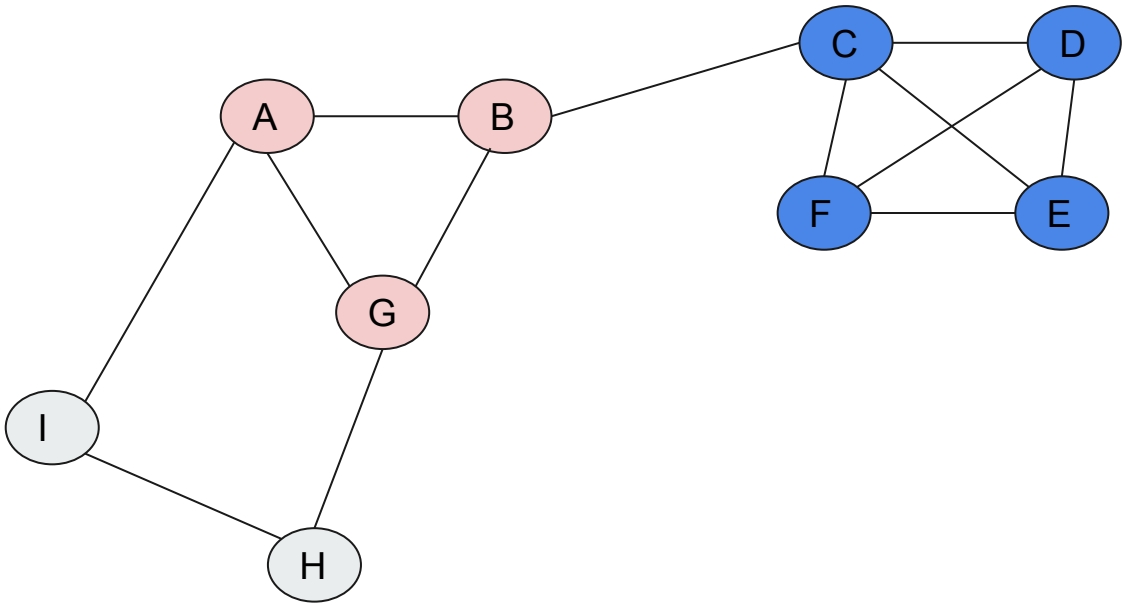
Given a graph G(V, E) and a number K, we need to answer yes or no if we can partition V(G) in K cliques

Minimum Clique Cover Problem:

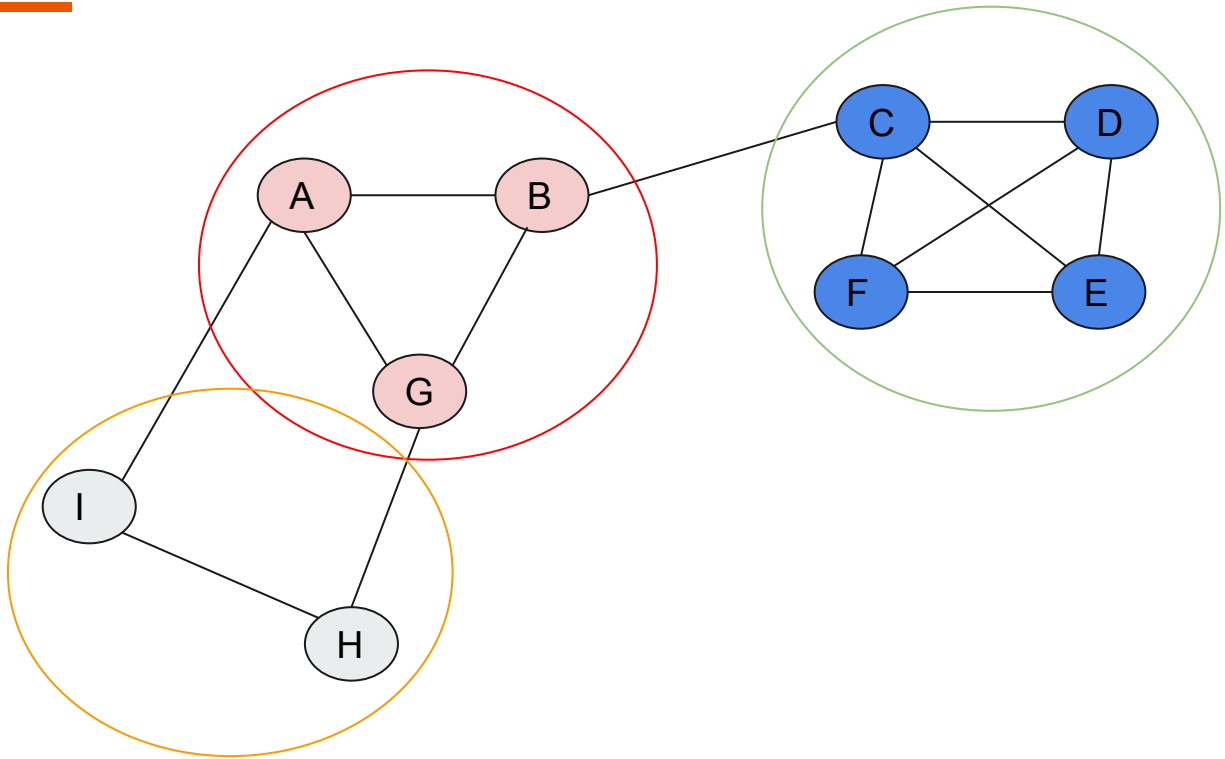
Given a graph G(V, E), we need to output the smallest number for which clique cover exists. This number is called the clique cover number.

In this presentation we will discuss some algorithms regarding these problems. As already discussed, we can derive a solution for clique cover decision problem from clique cover number and vice versa.

Let’s look at an example...



Let’s look at a scenario...



Algorithms We Have Covered

1. Brute Force and Exact Algorithm
    a. Sum over Subset DP
2. Approximation Algorithms
3. Heuristic Metaheuristic Algorithms
    a. Tabu Search for Vertex Clique Cover
    b. Greedy Clique Cover
    c. Iterative Greedy Clique Cover

Dataset

- 10 Randomly generated graphs
- DIMACS (Center for Discrete Mathematics and Theoretical Computer Science)
- Implementation challenge for NP-hard problems (1993)
- 21 graphs with different edge density,nodes and edges

Dataset

Graph Name	Nodes	Edges
Gnp10_0.2.clq	10	11
Gnp11_0.2.clq	11	11
Gnp13_0.2.clq	13	11
Gnp14_0.2.clq	14	22
Gnp16_0.2.clq	16	27
Gnp17_0.2.clq	17	26
Gnp19_0.2.clq	19	30
Gnp21_0.2.clq	21	38
Gnp22_0.2.clq	22	51
Gnp24_0.2.clq	24	52

Graph Name	Nodes	Edges	Edge Density	Graph Type
C125.9.clq	125	6963	89.8 %	Dense
gen200_p0.9_44.b	200	17910	90 %	Dense
gen200_p0.9_55.b	200	17910	90 %	Dense
brock200_2.b	200	9876	50 %	Normal
brock200_4.b	200	13089	65 %	Normal
C250.9.clq	250	27984	89.9 %	Dense
p_hat300_1.clq	300	10933	24.4 %	Sparse
p_hat300-2.clq	300	21928	48.9 %	Normal
gen400_p0.9_55.b	400	71820	90 %	Dense
gen400_p0.9_75.b	400	71820	90 %	Dense
gen400_p0.9_65.b	400	71820	90 %	Dense
brock400_2.b	400	59786	75 %	Dense
brock400_4.b	400	59765	75 %	Dense
C500.9.clq	500	112332	90 %	Dense
p_hat700-1.clq	700	60999	24.9 %	Sparse
p_hat700-2.clq	700	121728	49.8 %	Normal
brock800_2.b	800	208166	65 %	Normal
brock800_4.b	800	207643	65 %	Normal
C1000.9.clq	1000	450079	90.1 %	Dense
hamming10-4.clq	1024	434176	82.9 %	Dense
p_hat1500-1.clq	1500	284923	25.3 %	Sparse

Brute Force Algorithm

Step 1 : Find the power set of **V** where **V** is the set of vertices of graph **G**.
Step 2 : Take an empty list **S**.
Step 3 : Take every element(which is actually a subset of V) of **P(V)** and check whether it's a complete subgraph or clique then add this to list **S** otherwise skip the element.
Step 4 : Find all possible subsets of **S** of size **k**.
Step 5 : Take every subsets and check whether it covers all vertices of **V** or not.

Brute Force Algorithm

Step 1 : Find the power set of **V** where **V** is the set of vertices of graph **G**.
Step 2 : Take an empty list **S**.
Step 3 : Take every element(which is actually a subset of V) of **P(V)** and check whether it's a complete subgraph or clique then add this to list **S** otherwise skip the element.
Step 4 : Find all possible subsets of **S** of size **k**.
Step 5 : Take every subsets and check whether it covers all vertices of **V** or not.

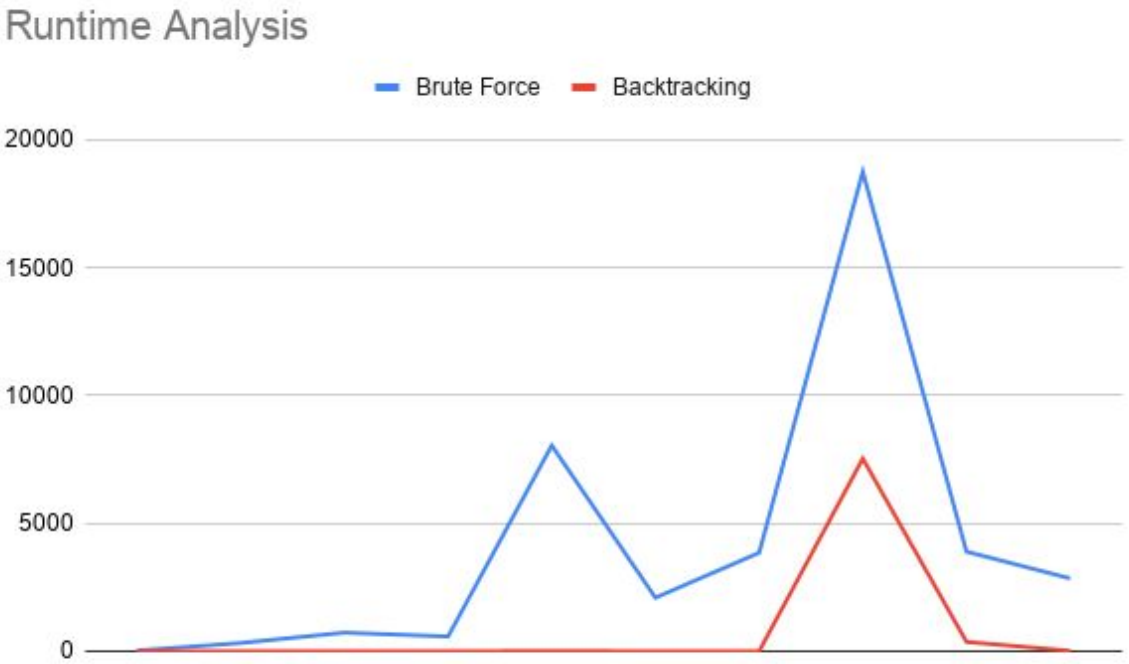
The time complexity of brute force algorithm is $O(n2^{kn})$

Brute Force Algorithm (DP Solution)

- Improvement of previous algorithm
- The number of configurations tested is way smaller
- Able to stop its exploration of the solution if it is sure that there won't be any viable & better solution with the given configuration
- Slightly faster !!!!

Runtime comparison (in second)

Graph Name	Nodes	Edges	Clique Cover	Brute Force	Backtracking
Gnp10_0.2.clq	10	11	5	26.873	0.002
Gnp11_0.2.clq	11	11	6	324.117	0.007
Gnp13_0.2.clq	13	11	9	728.866	0.002
Gnp14_0.2.clq	14	22	6	578.319	0.131
Gnp16_0.2.clq	16	27	8	8047.176	11.05
Gnp17_0.2.clq	17	26	9	2093.475	1.977
Gnp19_0.2.clq	19	30	10	3851.746	2.356
Gnp21_0.2.clq	21	38	10	18745.945	7533.604
Gnp22_0.2.clq	22	51	9	3895.389	366.368
Gnp24_0.2.clq	24	52	11	2847.249	20.143


Runtime Analysis
Brute Force
Backtracking
20000
15000
10000
5000
0

Exact DP Algorithm

1. Find all Cliques.
2. For each subset S of N vertices count the number of clique which are subset of this subset using SOS dp.
3. Calculate the number of K size subset of all clique set with repetition which misses at least one vertice.
4. If the above number of less than $(total\_clique)^k$, the we can cover the graph with k clique.

Greedy Clique Covering (GCC)

Input: graph **G** = **[V, E]**

permutation **P** = **[P_1, P_2, ..., P_n]** of vertices in **V**

Output: clique covering **S** of **G**

Algorithm:

1. for $c = 1..n$
2. sizes(c) = 0
3. for $i = 1..n$
4. $j = \mathbf{P}_i$
5. c = find_equal($\Gamma(v_j, c)$, sizes(c))
6. $\mathbf{V}_c = \mathbf{V}_c \cup \{v_j\}$
7. return $S = \{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_k\}$

Iterative Greedy Clique Covering (IG)

Input: graph **G** = [**V**, **E**]

Output: clique covering **S** of **G**

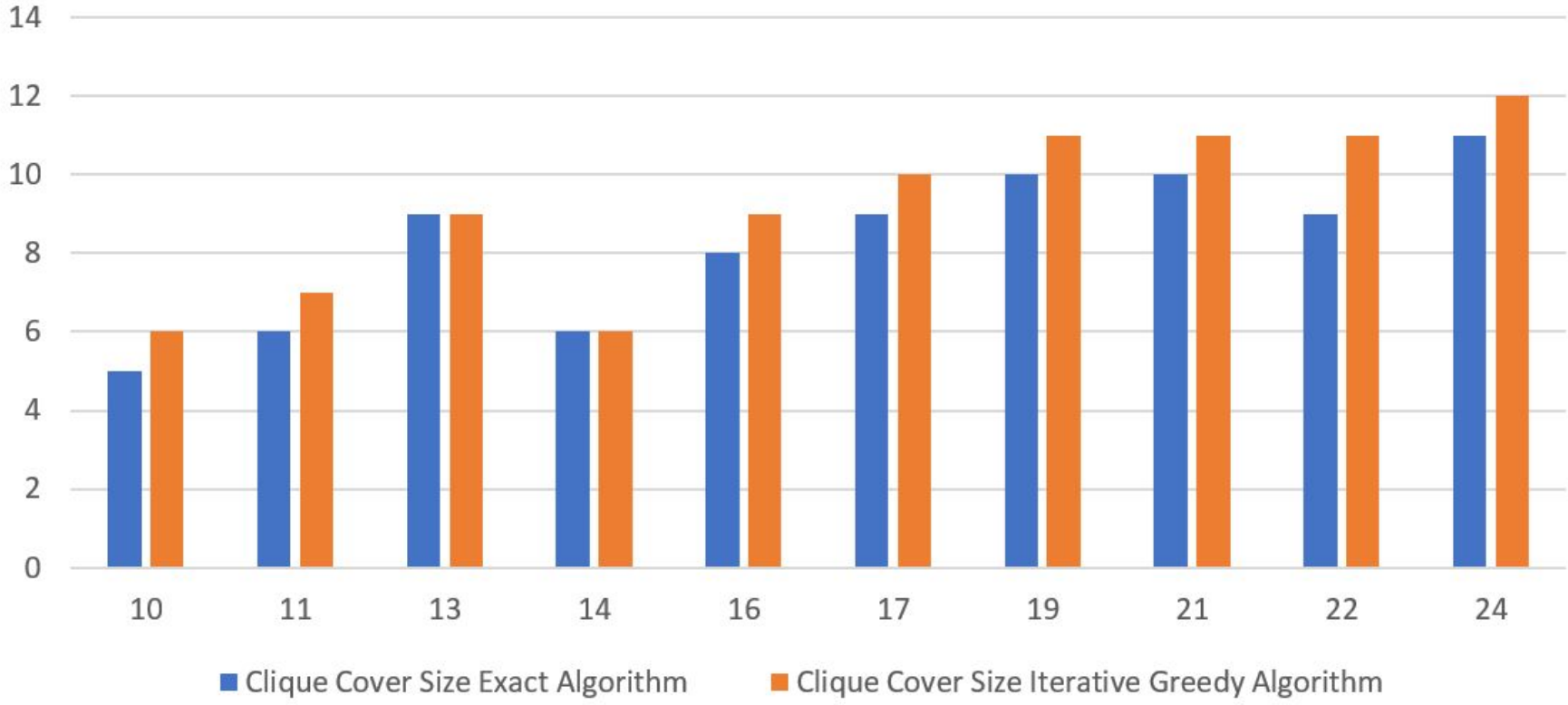
Algorithm:

1. **P** = random permutation(1, 2, ..., *n*)
2. while stopping criterion is not met
3. {$\mathbf{V}_1$, $\mathbf{V}_2$, ..., $\mathbf{V}_k$} = **GCC**(**G**, **P**)
4. with p_{rev} probability
5. **P** = [$\mathbf{V}_k$, $\mathbf{V}_{k-1}$, ..., $\mathbf{V}_1$]
6. else
7. **P** = random permutation($\mathbf{V}_1$, $\mathbf{V}_2$, ..., $\mathbf{V}_k$)
8. if ϑ(**G**) is known and *k* = ϑ(**G**)
9. return **S** = {**V**1, **V**2, ..., **V**k}
10. return **S** = {$\mathbf{V}_1$, $\mathbf{V}_2$, ..., $\mathbf{V}_k$}

Performance Comparison for Small Graph

Graph Name	Vertices	Edges	Clique Cover Size	
			Exact Algorithm	Iterative Greedy Algorithm
Gnp10_0_2	10	11	5	6
Gnp11_0_2	11	11	6	7
Gnp13_0_2	13	11	9	9
Gnp14_0_2	14	22	6	6
Gnp16_0_2	16	27	8	9
Gnp17_0_2	17	26	9	10
Gnp19_0_2	19	30	10	11
Gnp21_0_2	21	38	10	11
Gnp22_0_2	22	51	9	11
Gnp24_0_2	24	52	11	12

Exact vs Iterative Greedy Clique Cover Size Comparion

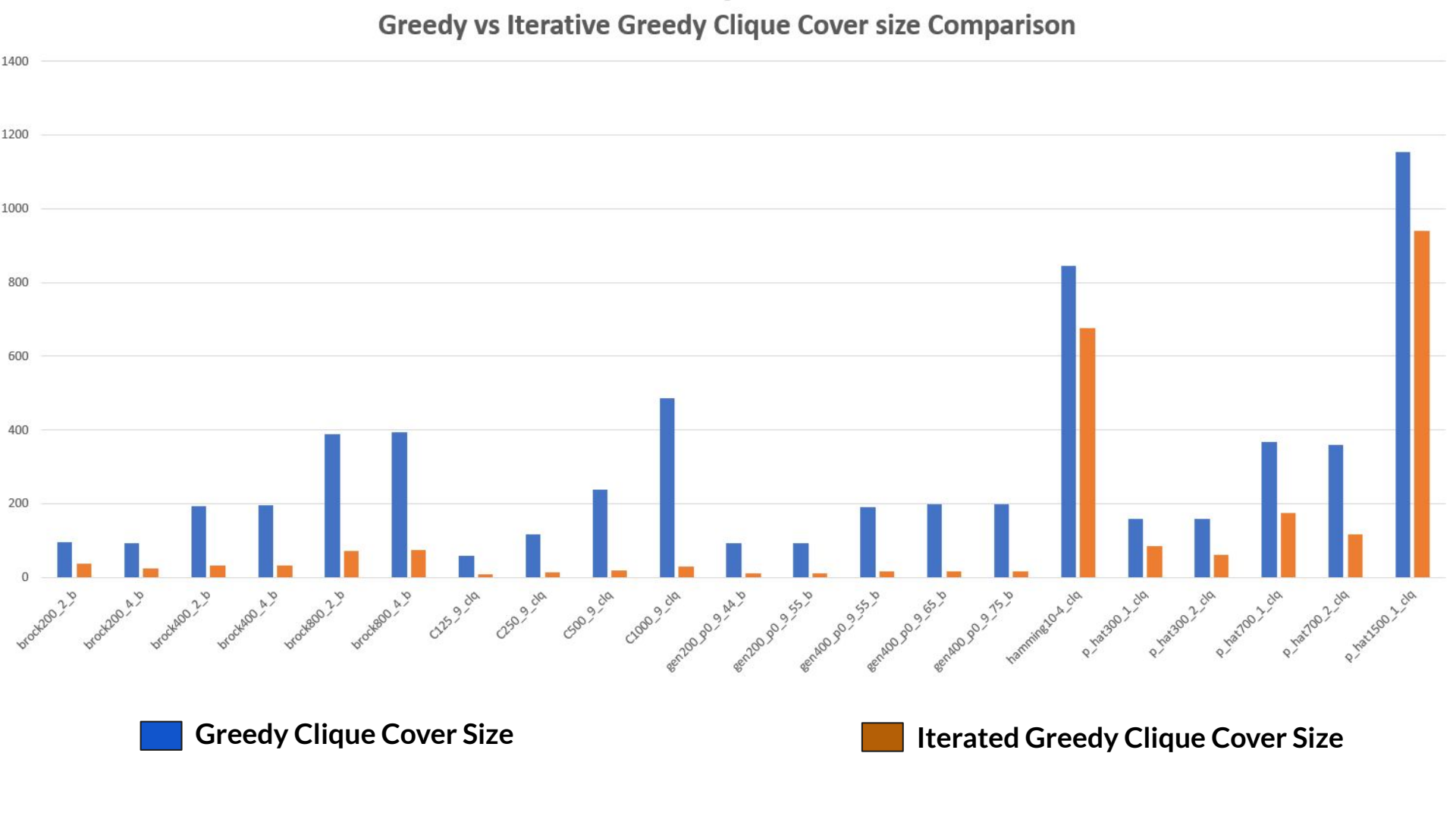


Performance Comparison on DIMACS Dataset

Graph Name	Vertices	Edges	Greedy Algorithm	Iterative Greedy Algorithm
brock200_2_b	200	9876	97	37
brock200_4_b	200	13089	92	25
brock400_2_b	400	59786	193	33
brock400_4_b	400	59765	196	33
brock800_2_b	800	208166	389	73
brock800_4_b	800	207643	394	74
C125_9_clq	125	6963	60	8
C250_9_clq	250	27985	118	13
C500_9_clq	500	112332	238	20
C1000_9_clq	1000	450079	487	31

Performance Comparison on DIMACS Dataset(continue)

Graph Name	Vertices	Edges	Greedy Algorithm	Iterative Greedy Algorithm
gen200_p0_9_44_b	200	17910	93	11
gen200_p0_9_55_b	200	17910	93	11
gen400_p0_9_55_b	400	71820	191	17
gen400_p0_9_65_b	400	71820	198	17
gen400_p0_9_75_b	400	71820	198	17
hamming10-4_clq	1024	51925	845	676
p_hat300_1_clq	300	10933	160	85
p_hat300_2_clq	300	21928	159	61
p_hat700_1_clq	700	60999	367	176
p_hat700_2_clq	700	121728	361	116
p_hat1500_1_clq	1500	67406	1153	941


Greedy vs Iterative Greedy Clique Cover size Comparison
1400
1200
1000
800
600
400
200
0
brock200_2_b
brock200_4_b
brock400_2_b
brock400_4_b
brock800_2_b
brock800_4_b
C125_9_clq
C250_9_clq
C500_9_clq
C1000_9_clq
gen200_p0_9_44_b
gen200_p0_9_55_b
gen400_p0_9_55_b
gen400_p0_9_65_b
gen400_p0_9_75_b
hamming10-4_clq
p_hat300_1_clq
p_hat300_2_clq
p_hat700_1_clq
p_hat700_2_clq
p_hat1500_1_clq
Greedy Clique Cover Size
Iterated Greedy Clique Cover Size

Thank You