Clique Maximal Enough

CliqueME is an approximation algorithm for the Maximal Clique in a graph with polynomial computational cost. Its operation requires the execution of a limited number of controls, in the surroundings of the optimal solution.

Pseudo Code

Considering:

- *n* number of nodes in graph

```
for i=1 to n do  // Loop 1
{
        put the node i in commons_vector
        while ( node i has adjacent nodes )   // Loop 2
        {
                put the node adjacent to i in commons_vector
                while ( node adjacent to i, has itself adjacent nodes) // Loop 3
                {
                        if the adjacent to itself, is also adjacent to i, put it in commons_vector
                }
                /* At the end of this loop, commons_vector is filled with commons node between
                   node i and its adjacent */
                for y=3 to n do // Loop 4
                {
                        /* I have to consider the third element of commons_vector, because the two
                           previous already form a clique */
                        for z=y+1 to n do // Loop 5
                        {
                                if node in commons_vector[y] NOT has as adjacent the node in
                                commons_vector[z]
                                {
                                        remove the node in commons_vector[z]
                                        move all subsequent one position to the left
                                        decrease z
                                }
                        }
        }
        /* At the end of this loop, in commons_vector will remain a clique, because purged from all
           nodes not common to those taken into consideration from time to time via the y counter */
        if (lenght of commons_vector > lenght CliqueMax)
                CliqueMax = commons_vector
        initialize commons_vector
        jump to the next node adjacent to i
        }
}
```

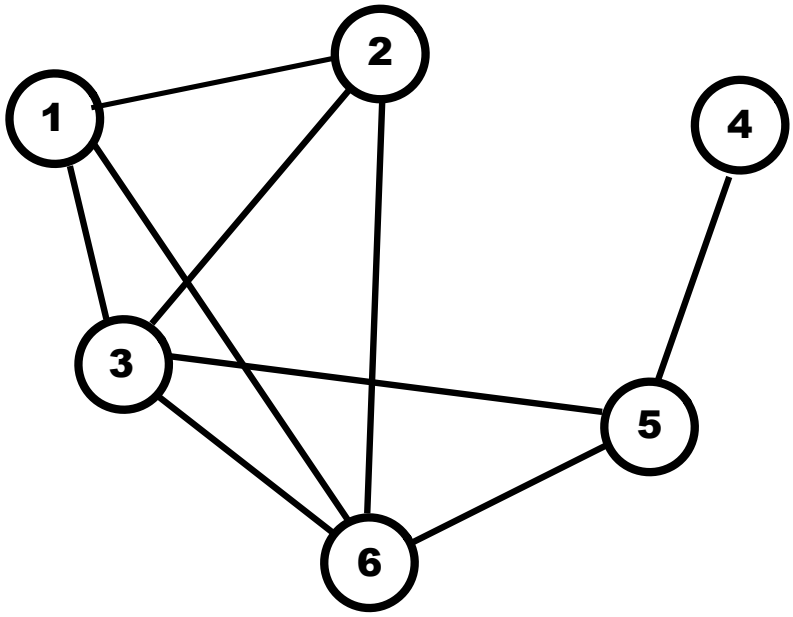
Computational Cost

As seen in pseudo code, we'll have 5 innested loops:

```
Loop 1 (n iterations in the worst case)
{
        Loop 2 (n iterations in the worst case)
        {
                Loop 3 (n iterations in the worst case)
                {
                        1 operation
                }
                Loop 4 (n iterations in the worst case)
                {
                        Loop 5 (n iterations in the worst case)
                        {
                                n operations
                        }
                }
        }
}
```

In the worst case, Loop 5, which is more innested, will have a cost of n^2
In his turn, Loop 4 will execute n times, n^2 operatons, so it will have a cost of n^3
Loop 3 execute one operation n times, so it's cost is linear (n).
Loop 2 execute n times (n operations + n^3), approximating its cost is of n^4
Loop 1 execute n times, n^4 operations, so its computational cost is of n^5

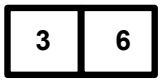
The algorithm, in the worst case, has a computational cost of θn^5



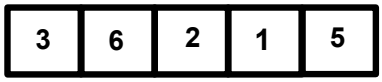
Example of working

i) Consider the node 3 and insert in a vector

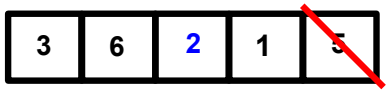
ii) Insert a his adjacent (in this case 6)



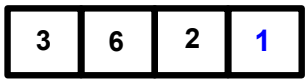
iii) Insert all the commons node between 3 and 6 (2 , 1 , 5)



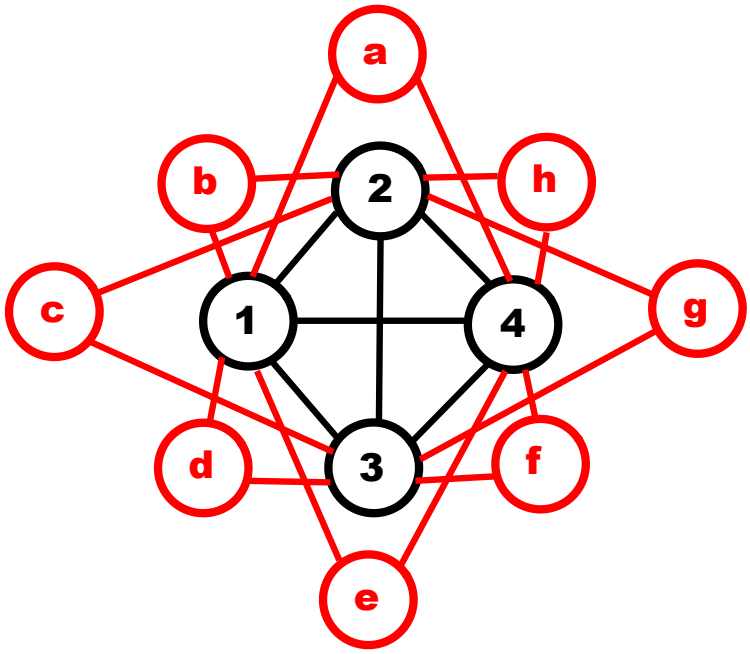
iv) Consider the third element (2) of vector and delete all the following not in common with it



v) Repeat, considering the fourth element. (1)



vi) But the fourth element has not subsequents to control and those on his right are certainly common to it.
The algorithm then terminates and the result is that within the vector will remain a clique.
In this case maximum. If until now we have not found a clique bigger than this, we save it as maximum up to now found.



Example of not working

i) Consider the node 3 and insert it in a vector

3

ii) Insert its adjacent (in this case 2)

3	2

iii) Insert all the commons node between 3 and 6 (c , 1 , 4)

3	2	c	1	4

iv) Consider the third element (c) of vector and delete all the following not in common with it

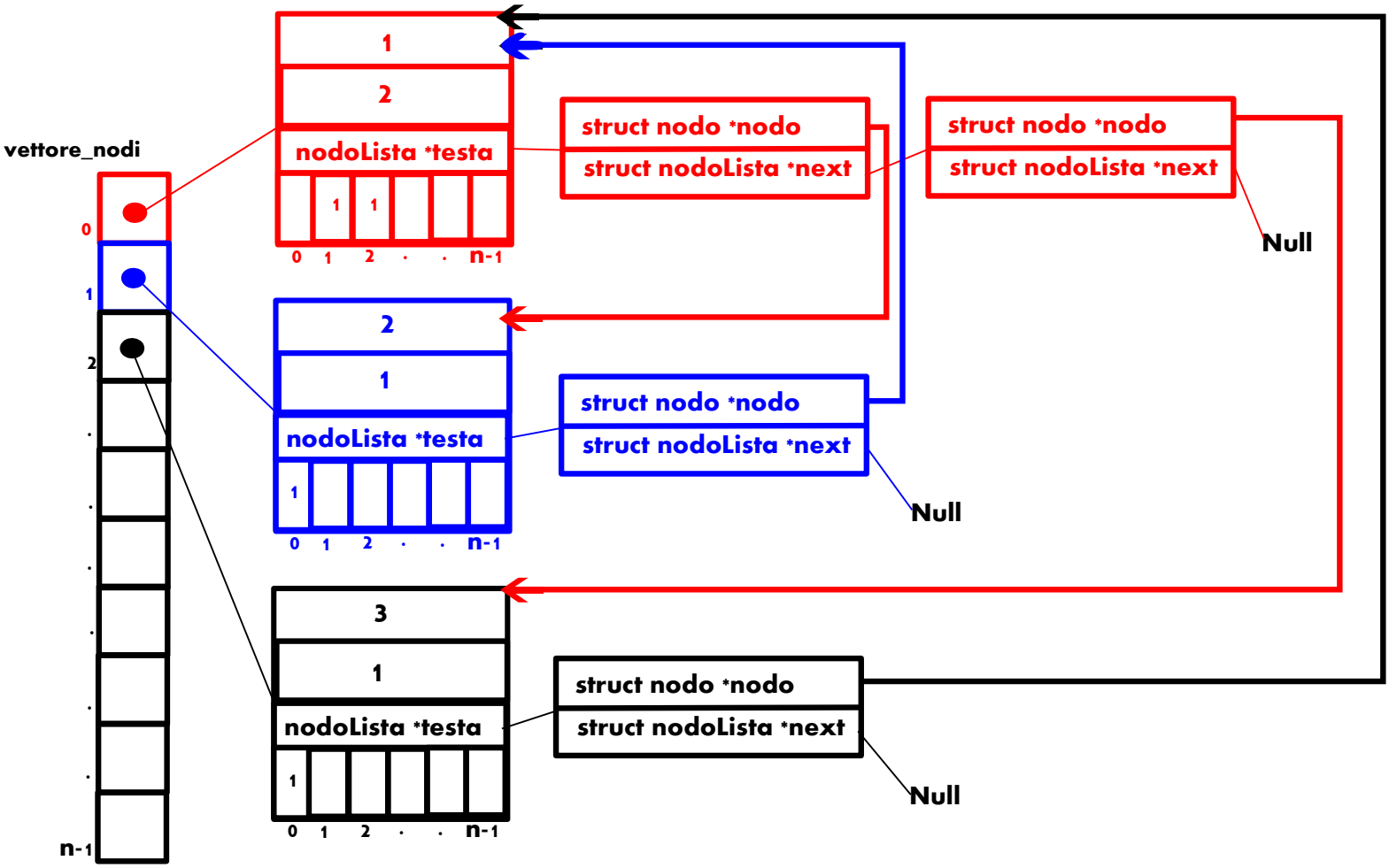
3	2	c	~~1~~	~~4~~

v) In this case, the added element c eliminates the nodes belonging to the maximum clique. The resulting vector will consist of just three elements. This happens even considering the other nodes in pairs of two. Note that permuting the order of the nodes, the example can terminate successfully.
In a real application, however, the permutations of (at worst) *n^2* edges, are *n^2!* (factorial), which is greater than *n^n* and therefore useless for the purpose of the algorithm.

Data structure used

```
struct nodo {
 int ID_Nodo;
 int N_Archi_Locali;
 struct nodoLista *testa;
 int *vettore_adiacenza;
};
```

```
struct nodoLista {
 struct nodo *nodo;
 struct nodoLista *next;
};
```



We can see that the structure at our disposal has both an adjacency matrix, and an adjacency list for each node. This involves a constant cost both to pass from one adjacent to another, andto control if two nodes are adjacent. The space occupied in the memory is about twice (in the worst case) of a adjacency matrix (n * n), which on modern machines is more than acceptable. For example, on a test with 4000 nodes and 7425226 arches, the space occupied in memory is about 300mb.

Final Considerations

The algorithm has a computational cost in the order of n^5 , as we have seen.
This allows you to control only a very limited number of *lucky* cases , exploiting the properties of the clique, that is to have all the nodes in common. The algorithm then has excellent probability to find the Maximum Clique in sparse graphs, but faces serious difficulties in particularly full graphs that have very small cliques, compared to the number of arcs in total.
For further analysis, other tests were performed on graphs found on the web.

Source: http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

All Benchmarks were performed on a single core at 2.6GHz, excluding the final, run on a single core at 2.0GHz.

Name	Nodes	Arcs	CliqueMax	Time	CliqueME
fbr30-15-1	450	83198	30	78s	26
fbr30-15-2	450	83151	30	78s	27
fbr30-15-3	450	83216	30	78s	27
fbr30-15-4	450	83194	30	81s	27
fbr30-15-5	450	83231	30	79s	26
fbr50-23-1	1150	580603	50	4531s	43
fbr50-23-2	1150	579824	50	4533s	43
fbr50-23-3	1150	579607	50	4440s	42
fbr50-23-4	1150	580417	50	4530s	42
fbr50-23-5	1150	580640	50	4476s	42
fbr59-26-1	1534	1049256	59	14948s	50
fbr59-26-2	1534	1049648	59	15091s	49
fbr59-26-3	1534	1049729	59	14897s	49
fbr59-26-4	1534	1048800	59	15283s	49
fbr59-26-5	1534	1049829	59	14563s	49
fbr100-40-1	4000	7425226	100	~7days	79

Andrea Mazzotti
A.Y. 2009-2010