Cairo University

Faculty of Engineering

Computer Engineering Department

Fourth year

MACHINE INTELLIGENCE

NuQClq: An Effective Local Search Algorithm for Maximum Quasi-Clique Problem

Team #16 Members

Name	**Section**	**B.N.**
Bishoy Atef	1	22
Robert Mounir	1	31
Mostafa Elgendy	2	26
Weam Bassem	2	36

Paper Overview:

The maximum quasi-clique problem (MQCP) is an important extension of the maximum clique problem with wide applications. Recent heuristic MQCP algorithms can hardly solve large and hard graphs effectively. The paper develops an efficient local search algorithm named NuQClq for the MQCP. There are two types of algorithms for the MQCP (which are exact algorithms and heuristic algorithms). Exact algorithms guarantee the optimality of their solutions, but they may fail to solve large-size instances of the problem.

Problem Definition:

Given a graph, a clique is a subset of vertices in which each pair of vertices in this subset are adjacent (i.e., connected with an edge).

The definition of MQCP:

Given a graph G = (V, E) and a fixed constant γ ∈ (0,1], a **γ-quasi-clique** is a subset S of V such that:

- S satisfies the property of clique
- Density(G[S]) >= γ, where G[S] is the subgraph induced by S, and Density(G) is the ratio between the number of edges in G and the number of all possible edges in G.

The solution to MQCP is to find a **γ-quasi-clique** S with the most vertices. The MQCP is proven to be np-hard for every value of γ.

The relevant topic of Machine Intelligence:

The paper developed a local search-based algorithm for solving the stated problem. The algorithm maintains a current candidate solution denoted as S which is a subset of V. It uses the following scoring functions to decide which vertex in the graph to add to (or remove from) the candidate solution:

- Primary scoring function:
  The adjacency $d_S(v)$ of a vertex $v \in V$ is the number of vertices in S that are connected to v.
  $d_S(v) = |\{u \mid u \in S, (u, v) \in E\}|$
  The algorithm prefers the vertex with higher value of d_S when choosing a vertex $v \notin S$ to be added to S, and prefers the vertex with lower value of d_S when choosing a vertex v to be removed from S. However, the values of d_S are often the same during the search. By experimenting, 14.64% of candidate vertices have the same best score values in the algorithm.

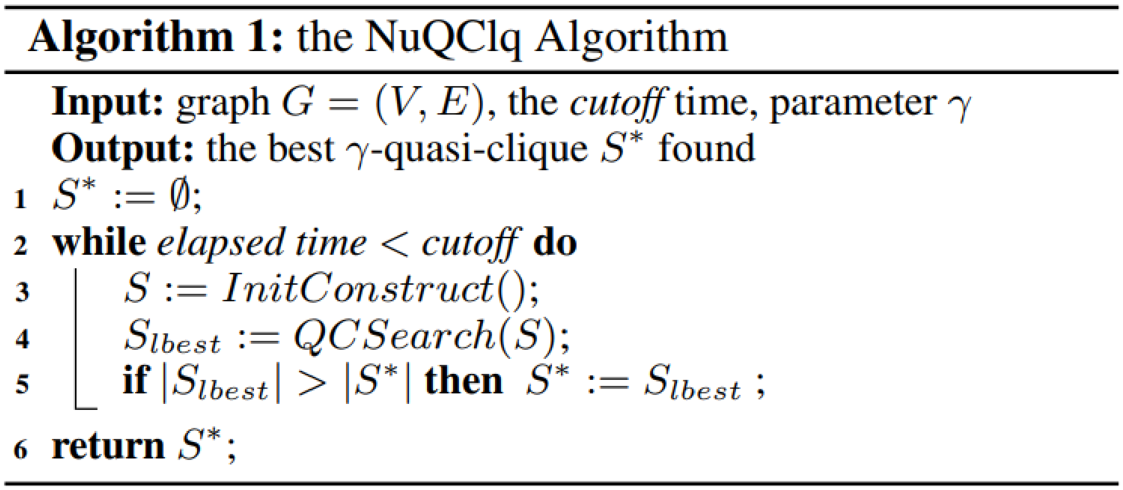
- Secondary scoring function:
  This function is designed to further select a vertex among the vertices that have the same d_S values. The edges are distinguished into three types according to a measurement on the number of endpoints included in S.

The function is denoted as λ(e), and it is evaluated on every edge in the graph:

- λ(e) = 0: this means that none of the vertices of e is included in S
- λ(e) = 1: this means that one of the vertices of e is included in S, e is called **critical edge** in this case.
- λ(e) = 2: this means that both vertices of e are included in S, e is called **full edge** in this case.

Intuitively, we should pick vertices which share more edges. A mechanism that encourages converting critical edges to full edges would help to achieve this. The algorithm prefers the vertices that are incident to **more critical edges** when choosing a vertex v ∉ S to be added to S, and prefers vertices that are incident to **fewer full edges** when choosing a vertex to be removed from S.

The proposed algorithm is shown in the following figure:



```
Algorithm 1: the NuQClq Algorithm
Input: graph $G = (V, E)$, the cutoff time, parameter $\gamma$
Output: the best $\gamma$-quasi-clique $S^*$ found
$S^* := \emptyset$;
while elapsed time $<$ cutoff do
    $S := InitConstruct()$;
    $S_{lbest} := QCSearch(S)$;
    if $|S_{lbest}| > |S^*|$ then $S^* := S_{lbest}$ ;
return $S^*$;
```

Initially, the best found γ-quasi-clique is initialized to an empty set and it is denoted by S^*.

The method InitConstruct works as follows: It selects a vertex with the lowest frequency value (the frequency of the vertex is increased whenever the vertex is added to (or removed from) the solution). Then it performs some iterations until the set cannot be expanded more and returns the candidate set.

The method QCSearch works as follows: It takes the candidate set as an input. Then, it iteratively modifies the set by using the previously stated scoring functions and returns the best local set (S_{lbest}).

Then, the algorithm compares the sizes of S^* and $S_{lbest}$and updates S^* if the local search produces a larger set.

The basic directions of the related work in literature:

Configuration checking (CC) is a diversification strategy that aims to reduce the cycling problem in local search algorithms. In the context of our problem, it can be discussed as the following: when selecting a vertex to be added to the candidate solution, check its configuration which is usually identified by the states in the neighborhood of the vertex (i.e., in S or not in S). If this configuration hasn't changed since the last removal of the vertex from S, then the vertex shouldn't be added back to S. One of the methods of CC used in literature is the dynamic-threshold configuration checking (DCC). It allows adding a vertex v if a certain number of vertices in the neighborhood of v have been added to S and is controlled by a dynamic threshold which is incremented whenever v is added to S.

The paper’s scientific contribution:

The paper proposed a new CC strategy named BoundedCC. It is implemented in a similar way to DCC. Notice that the DCC doesn’t put a limit on the threshold of the vertices which makes the frequently operated vertices have large thresholds and therefore they are prevented from being added to the solution for a long period of time. By experimenting, It was noticed that the maximum threshold value is 6.3 times more than the average threshold value which means that there exists vertices that will be forbidden for a long time. The BoundedCC strategy showed improvements over the DCC by making the maximum threshold value 1.5 times more than the average threshold value. This is done by setting an upper bound on the threshold so that no vertex is forbidden for a long time and whenever the threshold exceeds the upper bound, it is set to one.

Paper Evaluation:

The MQCP is applied widely in many applications such as social networks. The shown results in the paper indicate run-time improvement over three state of the art algorithms for that problem by using classic benchmarks. My opinion is that the paper includes all the necessary details and illustrations to be able to implement their proposed algorithm.