Calculating a maximum independent set of a given Graph

Gil Lopes Teixeira

*Resumo* – Este relatório apresenta dois métodos de calcular o Máximo Set Independete de um dado Grafo não orientado.

O primeiro algoritmo performa uma procura exaustiva entre todas as combinações de vertices não adjacentes. O Segundo é um algoritmo greedy que escolhe como primeiro vértice da solução o que tiver menos vertices adjacentes e continua da mesma forma para a lista de vérticess não adjacentes ao primeiro.

*Abstract* – This report presents two methods on calculating the Maximum Independent Set of a non oriented Graph.

The first is an exhaustive algorithm that searches between all the possible Sets of non adjacent vertexs. The second one is a greedy algorithm that first selects the vertex with least adjacent vertexs and then procedes with the same technique with the set of vertexs non-adjacent to the first one.

# I. Introduction

The maximum independent set of a graph is the largest set of non adjacent vertexs from the graph. In the following chapters we will elaborate on the algorithms built!

A lot of times there are multiple sets that are maximum independent sets. The exhaustive algorithm must search through every combination but the freedy finds a valid optimal solution with less complex steps.

# II. exhaustive Algorithm

Calculating the maximum independent set of a Graph G with n Vertexs:

:

Best case:

All vertexes are connected to the ith vertex!

Worst Case:

**findAdjacentVertexs(k):**

This depends on the number of edges! For a given graph with n vertexs the maximum amount of edges is:

Lets respresent this amount as !

The minimum amount of edges would be 0!

**Conclusion**

0 edges case:

Undirected Graph G with V vertexs but 0 edges:

Best case!

Most edges case:

Undirected Graph G with n vertexs but edges:

Worst Case!

Complexity:

# III. Greedy Algorithm

Calculating the maximum independent set of a Graph G with n vertexs:

This algorithm always chooses, in the next step, the valid\* vertex with minimum adjacent vertexs.

\*A valid vertex is a vertex that is not adjacente to the vertexs already choosen.

Let’s represente the length of the vertexs chosen list as v\_len

getMinEdgesVertex():

FindNonAdjacentVertexs(graph g, vertexs\_list v\_list):

Returns a subset of g containing vertexs that are not adjacente to any of the vertexs in v\_list.

Com v1 = len(v\_list)

findAdjacentVertexs(graph g, vertex v):

Worst Case:

Graph G with n vertexs and edges:

Best case:

Graph G with n vertexs and edges. 0!

**Conclusion**

Calculating the maximum independent set:

Graph G with n vertexs and edges:

findAdjacentVertexs with 0 edges represents 0.

findNonAdjacentVertexs(x)

, x being the size of the set of vertexs already selected!

So:

For a graph G with edges:

getMinEdges(i) = !

findAdjacentVertexs(i) = n\*(n-1)´

findNonAdjacentVertexs(i) = v\_len \* findAdjacentVertexs() + 2 \* n

Result:

=

Complexity: