Maximum Independent sets in a Graph

1. The report should be sent in as a PDF.
2. Clearly mark the front page of your report with your names and email addresses.
3. You are free to choose programming language, but you must make your own implementation of the main algorithms.
4. Due date: 17.00 the 21st of January.

1 Independent sets in a graph

An independent set in a graph \( G \) is a subset \( I \) of the vertices such that no vertex in \( I \) is a neighbour of any other vertex in \( I \). We say that \( I \) is a maximum independent set in \( G \) if no other independent set is larger than \( I \). The maximum independent set problem is to find the size of a maximum independent set in \( G \) and it is known to be a NP-hard computational problem. However many interesting problem are equivalent to finding a maximum independent set in a suitable graph, e.g. the problem of finding optimal error-correcting codes is a problem of this type.

The aim of this project is to first implement a greedy-type algorithm for finding a large, but mostly not optimal, independent set in a graph and then to compare the results with those of several versions of a branch and bound algorithm for the same problem.

1.1 Test graphs

In order to compare the programs we must have some graphs to use as input. We will construct a random family of test graphs.

1. Dense Random Graphs: A random graph on \( n \) vertices is defined by saying that for each pair of vertices \( u \) and \( v \) they are connected by an edge with probability 0.5. That is, for each pair of vertices \( u \) and \( v \) we can generate a random number in \((0,1)\) and if the random number is greater than 0.5 they are connected by an edge.

2. Medium Random Graphs: This random graph, on \( n \) vertices, is defined as for the dense case but now the probability that two vertices is connected is \( \frac{1}{\sqrt{n}} \).

3. Sparse Random Graphs: This random graph, on \( n \) vertices, is defined as for the dense case but now the probability that two vertices is connected is \( \frac{1}{n} \).

1.2 Problem

Implement a program which generates, dense random graphs, medium random graphs and sparse random graphs which you can use as input in the other problems.

2 A Greedy Algorithm

2.1 Method

The simple greedy algorithm for finding a large independent set works as follows.

1. Initialise \( I \) as \( I = \{\} \).
2. Find a vertex $u$ with the smallest possible number of neighbours in $G$. Set $I = I \cup \{u\}$ and delete $u$ and all neighbours of $u$ from $G$.

3. Repeat step 2 until there are no vertices left in $G$.

4. $I$ is our large independent set in $G$.

### 2.2 Problem

1. Implement the greedy algorithm and run it on the random graphs.

2. Make a plot showing the size of the independent set found as a function of $n$ for each type of graphs. For random graphs you should use at least 100 graphs for each value of $n$ that you use and take the average size of the independent sets found.

3. For how large values of $n$ can you run your program and stop within one minute for each type of graphs?

### 3 Branch and Bound

Our next task is to implement several variations of a branch and bound algorithm for finding the largest independent set in a graph and compare their performance. Each algorithm will be an improvement of the one before it.

#### 3.1 Algorithm 1: Stupid DFS branch and bound

A subproblem here will be given by a triple $(G, I, U)$, where $G$ is a graph and $I$ and $U$ are sets of vertices.

Given a graph $G$ for which we want to find a maximum independent set we now proceed as follows.

1. Fix an ordering of the vertices in $G$ and initialise a variable $bs = 0$. This variable will hold the size of the largest independents set found in $G$.

2. Start with list $S$ of subproblems containing only the pair $(G, \emptyset, \emptyset)$

3. Remove the most recent subproblem from $S$ and use that subproblem in the next step.

4. Let $u$ be the first vertex in $G$ which does not belong to $U$. If there is no such vertex the go to the next step otherwise do as follows:
   
   (a) If $I \cup \{u\}$ is an independent set in $G$ then add the two subproblems $(G, I, U \cup \{u\})$ and $G, I \cup \{u\}, U \cup \{u\}$ to $S$. If $|I| + 1 > bs$ then set $bs = |I| + 1$
   
   (b) If $I \cup \{u\}$ is not an independent set in $G$ then add the subproblem $(G, I, U \cup \{u\})$ to $S$.

5. If $S$ contains no more subproblems then the maximum independent set has size $bs$, otherwise go back to step 3.

This algorithms does a pure depth-first search to find the largest independent set in $G$.

#### 3.2 Problem

1. Implement the algorithm and run it on the random graphs.

2. Make a plot showing the size of the independent set found as a function of $n$ for each type of graph. For random graphs you should use at least 100 graphs for each value of $n$ that you use and take the average size of the independent sets found.

3. For how large values of $n$ can you run your program and stop within one minute for each type of graphs?
3.3 Algorithm 2: DFS branch and bound with constraint propagation

We will now make a small variation of the previous algorithm to include constraint propagation. The constraints in this problem is that if two vertices are connected by an edge then only one of them can belong to the same independent set. So if we have added a vertex to an independent set we can immediately draw the conclusion that none of its neighbours can be used.

Here we assume that our graph has $N$ vertices and that the vertices are named $1, \ldots, N$.

A subproblem here will be given by a triple $(G, I, A)$, where $G$ is a graph and $I$ and $A$ are sets of vertices. The set $A$ will contain the vertices which we are allowed to use in this subproblem.

1. Initialise a variable $bs = 0$. This variable will hold the size of the largest independent set found in $G$.

2. Start with list $S$ of subproblems containing only the pair $(G, \{\}, \{1, \ldots, N\})$.

3. Remove the most recent subproblem from $S$ and use that subproblem in the next step.

4. Let $u$ be the first vertex in $A$. If there is no such vertex the go to the next step otherwise do as follows:
   
   (a) If $I \cup \{u\}$ is an independent set in $G$ then add the two subproblems $(G, I, A_1)$ and $(G, I \cup \{u\}, A_2)$ to $S$. Here $A_1$ is $A$ with $u$ removed, and $A_2$ is constructed by removing both $u$ and all its neighbours from $A$.
   
   If $|I| + 1 > bs$ then set $bs = |I| + 1$

   (b) If $I \cup \{u\}$ is not an independent set in $G$ then add the subproblem $(G, I, A_1)$ to $S$. Here $A_1$ is $A$ with $u$ removed.

5. If $S$ contains no more subproblems then the maximum independent set has size $bs$, otherwise go back to step 3.

3.4 Problem

1. Implement the algorithm and run it on the random graphs.

2. Make a plot showing the size of the independent set found as a function of $n$ for each type of graphs. For random graphs you should use at least 100 graphs for each value of $n$ that you use and take the average size of the independent sets found.

3. For how large values of $n$ can you run your program and stop within one minute for each type of graphs?

3.5 Algorithm 3: Branch and bound with constraint propagation and bounds

We will now extend Algorithm 2 by making use of a bound as well. If we have constructed a independent set $I$ in $G$ but not yet looked at all vertices we might still be able to make $I$ larger. How much larger? It is hard to say exactly but a very simple bound is that it can not become larger than it’s current size plus all allowed vertices. Here we will also stop using the DFS strategy and instead use the subproblem with the best partial solution found so far.

Here we assume that our graph has $N$ vertices and that the vertices are named $1, \ldots, N$. A subproblem here will be given by a triple $(G, I, A)$, where $G$ is a graph and $I$ and $A$ are sets of vertices. The set $A$ will contain the vertices which we are allowed to use in this subproblem.

1. Run your greedy program and $bs$ to be the seize of the set found by the greedy program. This variable will hold the size of the largest independent set found in $G$.

2. Start with list $S$ of subproblems containing only the pair $(G, \{\}, \{1, \ldots, N\})$.

3. Remove the subproblem with the largest set $I$ from $S$ and use that subproblem in the next step.

4. Let $u$ be the first vertex in $A$. If there is no such vertex the go to the next step otherwise do as follows:
(a) If \( I \cup \{u\} \) is an independent set in \( G \) then
i. If \(|I| + 1 + |A_2| > bs\) then add the subproblem \((G, I \cup \{u\}, A_2)\) to \( S \). Here \( A_2 \) is constructed by removing both \( u \) and all its neighbours from \( A \). If \(|I| + 1 > bs\) then set \( bs = |I| + 1 \).
ii. If \(|I| + |A_1| > bs\) then add the subproblem \((G, I, A_1)\) to \( S \). Here \( A_1 \) is \( A \) with \( u \) removed.

(b) If \( I \cup \{u\} \) is not an independent set in \( G \) then add the subproblem \((G, I, A_1)\) to \( S \). Here \( A_1 \) is \( A \) with \( u \) removed.

5. If \( S \) contains no more subproblems then the maximum independent set has size \( bs \), otherwise go back to step 3.

3.6 Problem

1. Implement the algorithm and run it on the random graphs.

2. Make a plot showing the size of the independent set found as a function of \( n \) for each type of graphs. For random graphs you should use at least 100 graphs for each value of \( n \) that you use and take the average size of the independent sets found.

3. For how large values of \( n \) can you run your program and stop within one minute for each type of graphs?

4. Compare the size of the independent sets found with this algorithm with those found by the greedy algorithm. Plot the quotient between the two values for random graphs.

3.7 Algorithm 4: Branch and bound with constraint propagation and improved bounds

The bound used in the previous algorithm was very simple and by using a bit of graph theory we can derive a stronger bound. Given any graph with \( n \) vertices and \( m \) edges it is possible to prove that the graph cannot have an independent set larger than

\[
\frac{1}{2} \left( 1 + \sqrt{1 - 8m - 4n + 4n^2} \right).
\]

So if the graph given by the allowed vertices has \( n \) vertices and \( m \) edges we can use this value instead of \( n \), as we did in algorithm 3.

Whether this algorithm is faster or slower than Algorithm 3 will depend on how you have implemented your graphs. If the cost for keeping track of the number of edges is small this algorithm will be faster than Algorithm 3. If the cost for counting the number of edges is too large this will result in a slow-down.

3.8 Problem

1. Modify Algorithm 3 to use this improved bound instead, and run it on the random graphs.

2. Make a plot showing the size of the independent set found as a function of \( n \) for each type of graphs. For random graphs you should use at least 100 graphs for each value of \( n \) that you use and take the average size of the independent sets found.

3. For how large values of \( n \) can you run your program and stop within one minute for each type of graphs?

4. Compare the size of the independent sets found with this algorithm with those found by the greedy algorithm. Plot the quotient between the two values for the random graphs.
3.9 Algorithm 5: Branch and bound with constraint propagation and independent subproblems

If a graph has several components, i.e. it is not connected, a maximum independent set in the whole graph will be a union of maximum independent sets from each components. This means that each component can be treated as an independent subproblem.

3.10 Problem

1. Modify Algorithm 3 so that it computes the maximum independent set for the whole graph by computing the maximum independent set size for each component of the input graph and then adding their sizes. Run the program on the random graphs.

2. Make a plot showing the size of the independent set found as a function of \( n \) for each type of graphs. For random graphs you should use at least 100 graphs for each value of \( n \) that you use and take the average size of the independent sets found.

3. For how large values of \( n \) can you run your program and stop within one minute for each type of graphs?

4. Compare the size of the independent sets found with this algorithm with those found by the greedy algorithm. Plot the quotient between the two values for the random graphs.

4 Error detecting codes

One of the fundamental applications of the maximum independent set problem is the construction of error detecting and error correcting codes. In this section you will evaluate your code for the maximum independent set problem by finding efficient codes for code words of different lengths.

The set up here is that we have the set of all binary code words of length \( k \), these are simply all strings of length \( k \) of 0s and 1s. We now create a graph \( G_k \) with these strings as it’s vertices and two strings are neighbours if they differ in at most 2 positions. This corresponds to a situation where we want to be able to detect that up to two bit have been incorrectly transmitted and if at most one bit error has occurred we can also correct it.

Thus the graph \( G_k \) has \( 2^k \) vertices and each vertex will has degree \( \binom{k}{2} \).

4.1 Problem

1. Write a program which constructs the code graph for any \( k \).

2. Use you independent set program to find the maximum independent set in \( G_k \) for as large \( k \) as possible. Plot the running time as a function of \( k \) and the logarithm of the independent set size as a function of \( k \).

3. As a voluntary extra problem you can try to come up with and implement some additional modification of the code which makes it faster on these graphs, using some of the methods mentioned n the lectures.